Compositionality and Functorial Invariants in Machine Learning

Dan Shiebler

Kellogg College
University of Oxford

A thesis submitted in fulfillment for the degree of
Doctor of Philosophy

January 26, 2023

Abstract

The objective of this thesis is to show that studying the underlying compositional and functorial structure in machine learning systems allows us to better understand them. In order to do this, we explore category theoretic formulations of many subareas of machine learning, including optimization, probability, unsupervised learning, and supervised learning.

We begin with an investigation of how various optimization algorithms behave when we replace the gradient with a generic category theoretic structure. We prove that the key properties of these algorithms hold under very relaxed assumptions, and demonstrate this result through numerical experiments. We also explore a category theoretic perspective on dynamical systems that enables us to build powerful optimizers from the composition of simple operations.

Next, we take a category theoretic perspective on the relationship between probabilistic modeling and gradient based optimization. We use this perspective to study how maximum likelihood estimation preserves certain key structures in the transformation from a statistical model to a supervised learning algorithm.

Next, we take a functorial perspective on unsupervised learning. We develop taxonomies of unsupervised learning algorithms based on the category theoretic properties of their functorial representations, and demonstrate that these taxonomies are predictive of algorithm behavior. We use this perspective to derive a host of new unsupervised learning algorithms for clustering and manifold learning, and demonstrate that these new algorithms can outperform commonly used alternatives on real world data. We also use these tools to prove new results on the behavior and limitations of popular unsupervised learning algorithms, including refinement bounds and stability in the face of noise.

Finally, we turn to supervised learning and demonstrate that many of the most common problems in data science and machine learning can be expressed as Kan extensions. We use this perspective to derive novel classification and supervised clustering algorithms. We also explore the performance of these algorithms on real data.

Acknowledgements

I am very grateful to my advisors Jeremy Gibbons and Cezar Ionescu whose thoughtful and wise guidance shaped the development of my research throughout the last several years.

I am also grateful to everyone who collaborated with me on my research, including Bruno Gavranović, Paul Wilson, Alexis Toumi, and Mehrnoosh Sadrzadeh. Our discussions helped my research reach new levels.

I also want to thank all of the awesome researchers who generously took the time to share feedback with me over the years. Leland McInnes, Luis Scoccola, and Jared Culbertson responded to my cold emails with compelling and thoughtful input. Noson Yanofsky has given me great feedback and a forum to present my research. Many anonymous conference and journal reviewers took the time to read my papers and share their thoughts. Thank you for your time and help.

I would also like to thank my DPhil examiners Sam Staton and Vitaliy Kurlin for volunteering their time and providing a careful examination.

I am grateful to Kellogg College, the Department of Continuing Education, the Department of Computer Science, and the wonderful staff for supporting me through the administrative aspects of this process.

I would also like to thank Twitter and the incredibly smart people who I have worked with over the last several years. I strongly believe that having one foot in academia and one in industry has made me both a stronger researcher and a stronger engineer.

I want to particularly thank my close friends and family who supported me as I navigated my DPhil journey. They have been patient with my focus being elsewhere for long periods of time and tolerant of my enthusiasm for esoteric mathematics. My deepest gratitude is for my girlfriend Stephanie Diacovo and my parents Lori Goldstein and George Shiebler, who shared my excitement throughout every step of this process. I could not have done this without them.

Statement of Originality

The work in this dissertation is largely adapted from several papers that I have written over the last several years. The core insights and ideas in all of these papers is the product of many discussions with many people. One paper, roughly mapping to Chapter 3, was co-authored with Paul Wilson and Bruno Gavranović. In all other papers and Sections the writing is entirely my own.

Publications

Much of the work in this thesis has been adapted from the following publications:

- Dan Shiebler. Functorial clustering via simplicial complexes. *Topological Data Analysis and Beyond at NeurIPS 2020*, 2020c. URL https://openreview.net/pdf?id=ZkDLcXCP5sV
- Dan Shiebler. Categorical stochastic processes and likelihood. *Compositionality*, 3, April 2021c. ISSN 2631-4444. doi: 10.32408/compositionality-3-1
- Dan Shiebler. Flattening multiparameter hierarchical clustering functors. *International Conference on Geometric Science of Information*, 2021d. URL https://arxiv.org/pdf/2104.14734.pdf
- Dan Shiebler. Generalized optimization: A first step towards category theoretic learning theory. In *Intelligent Computing & Optimization*, pages 525–535. Springer International Publishing, 2022a. ISBN 978-3-030-93247-3
- Dan Shiebler. Functorial manifold learning. *Applied Category Theory*, 2021e. URL https://arxiv.org/abs/2011.07435
- Dan Shiebler, Bruno Gavranovic, and Paul W. Wilson. Category theory in machine learning. *Applied Category Theory*, 2021. URL https://arxiv.org/abs/2106.07032
- Dan Shiebler. Kan extensions in data science and machine learning. 2022b. URL https://arxiv.org/abs/2203.09018 (Submission to TMLR in progress)

Over the course of my DPhil I have also coauthored the following papers, but did not incorporate them into my thesis.

- Dan Shiebler, Alexis Toumi, and Mehrnoosh Sadrzadeh. Incremental monoidal grammars. 2020. URL https://arxiv.org/abs/2001.02296
- Alim Virani, Jay Baxter, Dan Shiebler, et al. Lessons learned addressing dataset bias in model-based candidate generation at Twitter. *Industrial Recommendation Systems at 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020
- Benjamin P. Chamberlain, Emanuele Rossi, Dan Shiebler, Suvash Sedhain, and Michael M. Bronstein. Tuning word2vec for large scale recommendation systems. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 732–737, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375832. doi: 10.1145/3383313.3418486

Contents

1 Introduction

1.1 My Thesis

The objective of this thesis is to show that studying the underlying compositional and functorial structure in machine learning systems allows us to better understand them. We aim to answer the following key questions:

Question 1. *Can we derive new, assumption light representations of the process of fitting a model on data?*

Answering Question 1 will help us understand how to adapt our learning procedures to new domains. This will also help us develop systems for optimization and model fitting that support new kinds of restrictions.

Question 2. *How does the structure of a model trained on a dataset mirror the structure of that dataset?*

Answering Question 2 will unveil what kind of adaptations we can make to a machine learning system without eroding the structural patterns that make it effective. This helps us predict the failure modes of existing machine learning systems and derive new machine learning algorithms from constraints.

Question 3. *Can we identify common structures that underlie seemingly different machine learning systems?*

Answering Question 3 will help us bridge apparently disparate fields of study within machine learning and transfer insights between them. This can also help us derive better course plans for machine learning that emphasize shared structure.

1.2 Understanding Machine Learning

Over the last several decades machine learning has evolved into one of the most important aspects of modern software. As computation and information have become increasingly embedded into society, the volume of data that we produce and the computational power of the machines we use have increased as well. Since the effectiveness of machine learning scales directly with the quantity of data and computational power, these increases have enabled machine learning solutions to outperform traditional techniques in a myriad of applied areas.

Machine learning algorithms now play a role in almost every aspect of everyday life, from the tools we use to the food we eat. Signal processing, visual reasoning, language understanding, content recommendation and forecasting problems can all be solved with machine learning. Furthermore, dozens of subfields of machine learning have become fields of study in their own right, including computational learning theory, deep learning, Bayesian inference, normalizing flows, clustering, reinforcement learning, and meta learning.

Any application of machine learning is powered by data. At its core, machine learning functions by using data generated by some process in order to make inferences about that process. In the most general case, we know little to nothing about the data generating process, and the data itself is just a string of numbers or values, devoid of inherent structure or meaning. However, once we make assumptions about this process or what

the data represents, we can build on our data to develop models with impressive predictive power.

Different assumptions are common in different applications, sometimes for principled reasons and sometimes not. In practice, the assumptions we make, problem statements we derive, and models we use are very loosely connected. There are few rigorous frameworks for reasoning about the relationships between these components. This makes it challenging to deeply understand the systems we use.

Furthermore, as machine learning grows in importance the systems that rely on it grow more complex as well. It is increasingly common for software systems to consist of multiple machine learning components interacting with each other. Predicting the behavior of such systems requires a deep understanding of how the pieces compose and how these compositions respond to changes.

1.3 Why Applied Category Theory?

One strategy for developing an understanding of such a complex system is to study it in its simplest form. By removing everything other than the minimal assumptions and structures, we can derive insights into the system's fundamental structure.

A strategy for doing this that has been growing in popularity is applied category theory. The abstractions present in category theory are useful for representing structures and patterns in complex systems. This enables applied category theoreticians to extract and predict common behaviors between systems that appear very different on the surface. As a result, category theory is becoming a unifying force in mathematics and physics. Over the last decade researchers have made major strides in the application of category theory to chemistry, statistics, game theory, causality, and database theory.

There are many different patterns that category theory can help represent and generalize. The two that we primarily focus on in this thesis are compositionality and functoriality.

1.3.1 Why Compositionality?

A compositional system is one in which larger or more complex pieces are made up of smaller or simpler ones. When we study a compositional system we focus on the ways in which we can effectively combine smaller pieces and understand their compositional hierarchy. Most modern machine learning applications, both at the system level and the model level, are inherently compositional. By developing a framework to study the composition of these systems, models and algorithms we may be able to better understand, constrain and guide their behaviors.

Today, the construction of most nontrivial machine learning systems is largely guided by heuristics about what works well in practice. While the individual components of complex models are generally well developed mathematically, their composition and combinations tend to be poorly understood. By studying the compositional behavior of statistical models and other system components we can define more intelligent rules around their combinations.

1.3.2 Why Functoriality?

A functor is a mapping between categories that preserves identity morphisms and morphism composition. Underlying this technical definition is a powerful concept: the func-

toriality of a transformation is a blueprint for its structure, expressed in terms of the invariants it preserves. If a given transformation is functorial over some pair of categories, then the transformation preserves the structure represented in those categories' morphisms. One way to develop a deeper understanding of a particular transformation or class of transformations is to identify the most interesting settings under which it is functorial. Once we have done this we can derive extensions or alternate versions of this transformation that preserve this functoriality, as well as modifications that break it. We summarize this with the following recipe:

1. Identify a transformation of interest.
2. Find the most interesting categories over which this transformation is functorial.
3. Figure out what interesting extensions we can make to the transformation such that it remains functorial.

Many of the most crucial questions we have about a machine learning system can be expressed as questions about transformation invariance and equivariance. If this data changes, what will happen to my model? If I apply this transformation to my training procedure, how will my model respond? What kind of data or model changes will damage the well behavedness conditions of my system?

Functoriality enables us to rigorously address these questions and therefore develop a deeper understanding of the relationships between the components of machine learning systems.

1.4 Overview of this Dissertation

In Chapter 3 we explore some of the previous research applying category theory to machine learning. Since many modern machine learning systems are inherently compositional it is unsurprising that a number of authors have begun to study them through the lens of category theory. We survey category theoretic perspectives on three areas:

- Gradient based methods: Building from the foundations of automatic differentiation to neural network architectures, loss functions and model updates.
- Probabilistic methods: Building from the foundations of probability to simple Bayesian models.
- Invariant and Equivariant Learning: Characterizing the invariances and equivariances of unsupervised and supervised learning algorithms.

In Chapter 4 we aim to answer Question 1. First, in Section 4.1 we use the Cartesian reverse derivative, a categorical generalization of reverse mode automatic differentiation, to generalize several optimization algorithms. We study straightforward generalizations of gradient descent and momentum as well as a novel generalization of Newton's method. We then explore which properties of these algorithms are preserved in this generalized setting. We show that the transformation equivariances of these algorithms are preserved: generalized Newton's method is equivariant under all invertible linear transformations, generalized gradient descent and generalized momentum are equivariant under orthogonal linear transformations. Next, we show that we can express the change in loss of generalized gradient descent with an expression that is similar to an inner product, thereby

generalizing the non-increasing and convergence properties of the gradient descent optimization flow. Finally, we include several numerical experiments to illustrate these ideas and demonstrate how we can use them to optimize polynomial functions over an ordered ring.

Next, in Section 4.2 we explore the dynamical systems defined by the application of algorithms like gradient descent and momentum to optimize a parametric function. We use a category theoretic framework to study the composition and recombinations of the dynamical systems defined by these algorithms. In particular, we explore how we can leverage the composition of dynamical systems to construct complex optimization algorithms from simpler components. This enables us to construct momentum and stochastic momentum from the composition of gradient descent with simple data passing operations.

In Chapter 5 we continue to tackle Question 1. We take a category theoretic perspective on the relationship between probabilistic modeling and gradient based optimization. We define two extensions of function composition to stochastic process subordination: one based on a co-Kleisli category and one based on the parameterization of a category with a Lawvere theory. We show how these extensions relate to the category of Markov kernels through a pushforward procedure. We then extend stochastic processes to parametric statistical models and their likelihood functions. Finally, we demonstrate how the maximum likelihood estimation procedure defines a family of identity-on-objects functors from categories of statistical models to the category of supervised learning algorithms.

In Chapter 6 we focus on answering Question 2 and Question 3. First, in Section 6.1 we characterize a class of hierarchical overlapping clustering algorithms as functors that factor through a category of simplicial complexes. We first develop a pair of adjoint functors that map between simplicial complexes and the outputs of clustering algorithms. Next, we introduce the maximal and single linkage clustering algorithms as the respective composition of the flagification and connected components functors with a finite singular set functor. We then demonstrate that all other hierarchical overlapping clustering functors are refined by maximal linkage and refine single linkage.

Next, in Section 6.2 we expand this perspective to study multiparameter hierarchical clustering. We begin by introducing a procedure for flattening multiparameter hierarchical clusterings. We also include empirical results demonstrating its effectiveness. Next, we introduce a Bayesian update algorithm for learning clustering parameters from data. We demonstrate that the composition of this algorithm with our flattening procedure satisfies a consistency property.

Next, in Section 6.3 we develop a functorial perspective on manifold learning, also known as nonlinear dimensionality reduction. We first characterize manifold learning algorithms as functors that map metric spaces to optimization objectives and that factor through hierarchical clustering functors. We then use this characterization to prove refinement bounds on manifold learning loss functions and construct a hierarchy of manifold learning algorithms based on their equivariants. We express several popular manifold learning algorithms as functors at different levels of this hierarchy, including metric multidimensional scaling, isomap, and UMAP. Next, we use interleaving distance to study the stability of a broad class of manifold learning algorithms. We present bounds on how closely the embeddings these algorithms produce from noisy data approximate the embeddings they would learn from noiseless data. Finally, we use our framework to derive a set of novel manifold learning algorithms, which we experimentally demonstrate are competitive with traditional approaches.

Next, in Chapter 7 we continue to tackle answering Question 3. We investigate how

the Kan extension can represent disparate applications within supervised learning. In particular, we explore the general problem: "use this function defined over this small set to generate predictions over that larger set." First, we derive a simple classification algorithm as a Kan extension and experiment with this algorithm on real data. Next, we use the Kan extension to derive a procedure for learning clustering algorithms from labels and explore the performance of this procedure on real data. We then investigate how Kan extensions can be used to learn a general mapping from datasets of labeled examples to functions and to approximate a function in one class with a function in another class

1.5 Overview of Results

In this section we share some of our primary results. Lines marked with $*$ are supported by computational experiments:

- (Chapter 4, Theorem 4.25 and Theorem 4.28) Several optimization algorithms, including gradient descent, momentum, and Newton's method, maintain their transformation equivariance properties when we generalize derivatives to Cartesian derivatives.
- (Chapter 4, Theorem 4.35)* There exist optimization domains other than the domain of differentiable functions between Euclidean spaces that support gradient based optimization. For example, the domain of polynomials over ordered rings.
- (Chapter 4, Proposition 4.48 and Proposition 4.44) It is possible to construct the momentum and stochastic momentum optimization algorithms in terms of the composition and parallel products of categorical lenses.
- (Chapter 5, Theorem 5.7)* There are at least two meaningfully different ways to compose stochastic processes: one based on parametric function composition and one based on the co-Kleisli category of the product comonad. The pushforward procedure that transforms stochastic processes into Markov kernels is functorial when we use parametric function composition, but not functorial when we use co-Kleisli composition.
- (Chapter 5, Theorem 5.21 and Definition 5.22) The maximum likelihood procedure is a functor from a category of parametric statistical models to a category of learning algorithms.
- (Chapter 6, Table 4)* It is possible to outperform choosing the optimal hyperparameter value for a multiparameter hierarchical clustering algorithm by instead flattening the algorithm's output with a binary integer program.
- (Chapter 6, Theorem 6.35)* The composition of a novel Bayesian algorithm for learning a distribution over clustering hyperparameters from data with the algorithm for flattening a multiparameter hierarchical clustering is consistent.
- (Chapter 6, Theorem 6.51) We can project many manifold learning algorithms along a spectrum based on the criterion by which learned embeddings are penalized for being too close together or too far apart. We can derive new manifold learning algorithms by moving existing algorithms along this spectrum.

- (Chapter 6, Section 6.3.4 and Section 6.3.5) We can predict the behaviors of many common manifold learning algorithms by studying the categories over which they are functorial.
- (Chapter 6, Theorem 6.65) We can use the interleaving distance of functors to bound the difference between the embeddings that certain manifold learning algorithms construct from noisy and noiseless data.
- (Chapter 6, Section 6.3.7)* We can recombine functors to construct novel well behaved manifold learning algorithms or improve the performance of manifold learning algorithms on certain tasks.
- (Chapter 6, Proposition 6.69) We can extrapolate the output of a clustering algorithm to a larger set by taking a colimit.
- (Chapter 7, Theorem 7.1 and Theorem 7.9)* We can derive novel classification and clustering algorithms by taking the Kan extension of a functor relating training samples to labels. These algorithms can learn to cluster or classify images.
- (Chapter 7, Proposition 7.14 and Proposition 7.20) We can use Kan extensions to derive collections of supervised learning algorithms from pairs of labeled datasets and trained functions or to approximate functions in one class with functions in another class.

2 Preliminaries

In this Chapter we review a few key definitions to fix notation. Although none of these definitions are fully novel, some have been reframed in slightly different language than in previous work to ease their usage throughout this dissertation.

2.1 Category Theory

We assume that readers have a basic familiarity with Category Theory at the level of "Basic Category Theory" (Leinster, 2016) or "Seven Sketches in Compositionality" (Fong and Spivak, 2019). We exclusively consider small categories (categories where the collections of objects and morphisms are sets) in this work.

Definition 2.1. *Given the categories* $\mathbf{C}, \mathbf{D}$ *the functors:*

$$F : \mathbf{D} \to \mathbf{C} \qquad G : \mathbf{C} \to \mathbf{D}$$

form an adjunction:

$$F \dashv G$$

if for any objects $c \in \mathbf{C}, d \in \mathbf{D}$ *we have that there exists a natural isomorphism:*

$$\mathbf{C}[F(d), c] \simeq \mathbf{D}[d, G(c)]$$

We call F *the left adjoint of* G *and we call* G *the right adjoint of* F*.*

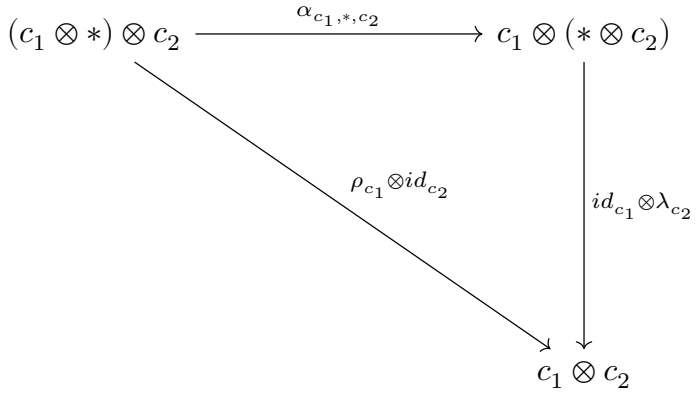
Intuitively, an adjunction between the categories $\mathbf{C}, \mathbf{D}$ describes how the structure in the categories are similar.

Definition 2.2. *A monoidal category is a category equipped with a functor* $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ *which we call the monoidal tensor, a unit object* $*$*, an associator natural transformation*

$$\alpha_{c_1,c_2,c_3} : (c_1 \otimes c_2) \otimes c_3 \to c_1 \otimes (c_2 \otimes c_3)$$

a left unitor natural transformation $\lambda_c : * \otimes c \to c$*, and a right unitor natural transformation* $\rho_c : c \otimes * \to c$ *such that the following diagrams commute:*

$$\begin{array}{ccccc}
((c_1 \otimes c_2) \otimes c_3) \otimes c_4 & \xrightarrow{\alpha_{c_1 \otimes c_2, c_3, c_4}} & (c_1 \otimes c_2) \otimes (c_3 \otimes c_4) & \xrightarrow{\alpha_{c_1, c_2, c_3 \otimes c_4}} & (c_1 \otimes (c_2 \otimes (c_3 \otimes c_4))) \\
\Big\downarrow{\scriptstyle \alpha_{c_1,c_2,c_3} \otimes id_{c_4}} & & & & \Big\uparrow{\scriptstyle id_{c_1} \otimes \alpha_{c_2,c_3,c_4}} \\
(c_1 \otimes (c_2 \otimes c_3)) \otimes c_4 & & \xrightarrow{\alpha_{c_1, c_2 \otimes c_3, c_4}} & & c_1 \otimes ((c_2 \otimes c_3) \otimes c_4)
\end{array}$$



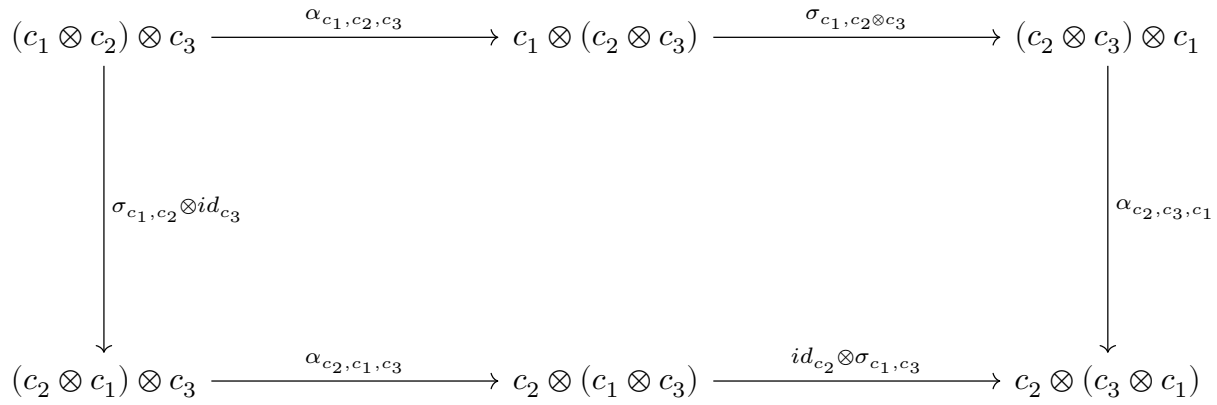
Monoidal categories will appear repeatedly over the course of this dissertation. Given an object c or morphism f in a monoidal category $\mathbf{C}$, we write $c^k = c \otimes c \otimes \cdots \otimes c$ and $f^k = f \otimes f \otimes \cdots \otimes f$ to respectively denote c and f tensored with themselves k times.

Definition 2.3. *A strict monoidal category* $\mathbf{C}$ *is a monoidal category in which the associators* α_{c_1,c_2,c_3} *and unitors* ρ_c, λ_c *are identity morphisms.*

Definition 2.4. *A symmetric monoidal category* $\mathbf{C}$ *is a monoidal category further equipped with a symmetric swap natural transformation:*

$$\sigma_{c_1,c_2} : c_1 \otimes c_2 \to c_2 \otimes c_1$$

such that the following diagram commutes:



and we have:

$$\sigma_{c_2,c_1} \circ \sigma_{c_1,c_2} = id_{c_1 \otimes c_2}$$

A particularly important kind of symmetric monoidal category is a Cartesian monoidal category:

Definition 2.5. *A Cartesian monoidal category is a monoidal category in which the monoidal tensor is the categorical product* $\times$ *and the unit object* $*$ *is the terminal object of* $\mathbf{C}$*.*

We write the projection maps in a Cartesian category $\mathbf{C}$ as:

$$\pi_1 : c_1 \times c_2 \to c_1 \qquad \pi_2 : c_1 \times c_2 \to c_2$$

and for any morphisms:

$$f_1 : c_1 \to c_2 \qquad f_2 : c_1 \to c_3$$

in $\mathbf{C}$ we use the notation $\langle f_1, f_2 \rangle : c_1 \to c_2 \times c_3$ to express the unique morphism such that:

$$\pi_1 \circ \langle f_1, f_2 \rangle = f_1 \qquad \pi_2 \circ \langle f_1, f_2 \rangle = f_2$$

One Cartesian category we will use as an example throughout this dissertation is the following:

Proposition 2.6. *We can define a Cartesian monoidal category* **Euc** *in which objects are natural numbers $a \in \mathbb{N}$, the morphisms from a to b are infinitely differentiable maps from $\mathbb{R}^a$ to $\mathbb{R}^b$, and the product of the objects a and b is $a + b$. The monoidal unit (and terminal object) in* **Euc** *is* 0.

Proof. It is easy to see that **Euc** is a category since infinite differentiability is preserved on function composition and the identity function is infinitely differentiable.

We can see that $a + b$ is a categorical product because the projection maps from $\mathbb{R}^{a+b}$ to $\mathbb{R}^a$ and $\mathbb{R}^b$ are universal. 0 is trivially the terminal object because for any $\mathbb{R}^a$ the map $\mathbb{R}^a \to \{*\}$ is unique. Therefore **Euc** is Cartesian monoidal. □

2.1.1 Lenses

Lenses are a general construction that we can use to represent pairs of processes that move in opposite directions.

Definition 2.7. *Given a Cartesian category* $\mathbf{C}$ *we define a* $\mathbf{C}$*-lens:*

$$\binom{A'}{A} \xrightarrow{(f_g, f_p)} \binom{B'}{B}$$

to be a pair of morphisms (f_g, f_p) in $\mathbf{C}$*:*

$$f_g : A \to B$$
$$f_p : A \times B' \to A'$$

We sometimes call f_g the "get" map and f_p the "put" map.

For more details on lenses see "Categories of Optics" (Riley, 2018).

Lenses are powerful because many computations can be expressed in terms of the combination of multiple lenses. The simplest way to combine lenses is to stack them in parallel:

Definition 2.8. *Given the lenses:*

$$\binom{A'}{A} \xrightarrow{(f_g, f_p)} \binom{B'}{B}$$
$$\binom{C'}{C} \xrightarrow{(g_g, g_p)} \binom{D'}{D}$$

we define their parallel product to be the lens:

$$\binom{A' \times C'}{A \times C} \xrightarrow{(h_g, h_p)} \binom{B' \times D'}{B \times D}$$

where:

$$h_g = f_g \times g_g$$
$$h_p = \langle f_p \circ (\pi_0 \times \pi_0), (g_p \circ (\pi_1 \times \pi_1) \rangle$$

We can draw h_p as:

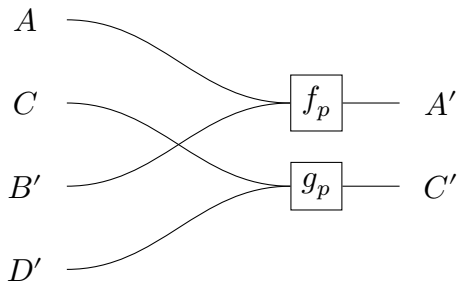


Figure 1: *The put map in a parallel product of lenses*

We can also compose lenses directly:

Definition 2.9. *Given the lenses:*

$$\binom{A'}{A} \xrightarrow{(f_g, f_p)} \binom{B'}{B} \qquad \binom{B'}{B} \xrightarrow{(g_g, g_p)} \binom{C'}{C}$$

we define their composition to be the lens:

$$\binom{A'}{A} \xrightarrow{(h_g, h_p)} \binom{C'}{C}$$

where:

$$h_g = g_g \circ f_g$$
$$h_p = f_p \circ \langle \pi_0, (g_p \circ \langle f_g \circ \pi_0, \pi_1 \rangle) \rangle$$

We can equivalently write h_p as:

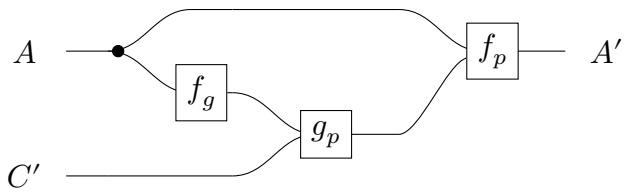
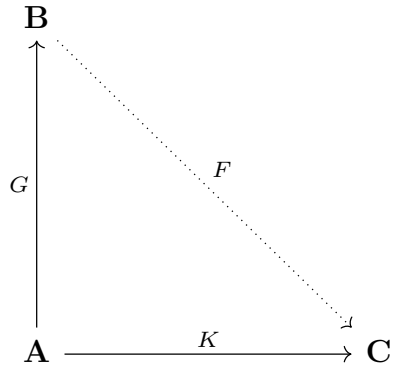


Figure 2: *Composition of put maps*

2.1.2 Kan Extensions

Suppose we have three categories $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and two functors $G : \mathbf{A} \to \mathbf{B}, K : \mathbf{A} \to \mathbf{C}$ and we would like to derive the "best" functor $F : \mathbf{B} \to \mathbf{C}$:



There are two canonical ways that we can do this:

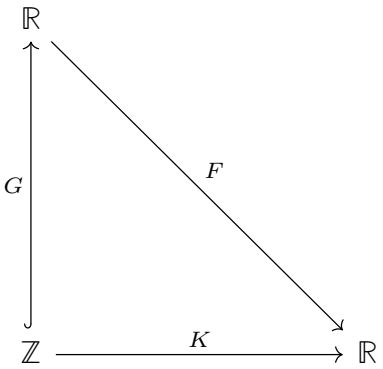
Definition 2.10. *The left Kan extension of K along G is the universal pair of the functor $Lan_G K : \mathbf{B} \to \mathbf{C}$ and natural transformation $\mu : K \to (Lan_G K \circ G)$ such that for any pair of a functor $M : \mathbf{B} \to \mathbf{C}$ and natural transformation $\lambda : K \to (M \circ G)$ there exists a unique natural transformation $\sigma : Lan_G K \to M$ such that $\lambda = \sigma_G \circ \mu$ (where $\sigma_G(a) = \sigma(Ga)$).*

Definition 2.11. *The right Kan extension of K along G is the universal pair of the functor $Ran_G K : \mathbf{B} \to \mathbf{C}$ and natural transformation $\mu : (Ran_G K \circ G) \to K$ such that for any pair of a functor $M : \mathbf{B} \to \mathbf{C}$ and natural transformation $\lambda : (M \circ G) \to K$ there exists a unique natural transformation $\sigma : M \to Ran_G K$ such that $\lambda = \mu \circ \sigma_G$ (where $\sigma_G(a) = \sigma(Ga)$).*

For more details on Kan extensions see "Categories for the Working Mathematician" (MacLane, 1978).

Intuitively, if we treat G as an inclusion of $\mathbf{A}$ into $\mathbf{B}$ then the Kan extensions of K along G act as extrapolations of K from $\mathbf{A}$ to all of $\mathbf{B}$. If $\mathbf{C}$ is a preorder then the left and right Kan extensions respectively behave as the least upper bound and greatest lower bounds of K.

For example, suppose we want to interpolate a monotonic function $K : \mathbb{Z} \to \mathbb{R}$ to a monotonic function $F : \mathbb{R} \to \mathbb{R}$ such that $F \circ G = K$ where $G : \mathbb{Z} \hookrightarrow \mathbb{R}$ is the inclusion map (morphisms in $\mathbb{Z}, \mathbb{R}$ are $\leq$).



We have that $Lan_G K : \mathbb{R} \to \mathbb{R}$ is simply $K \circ floor$ and $Ran_G K : \mathbb{R} \to \mathbb{R}$ is simply $K \circ ceil$, where $floor, ceil$ are the rounding down and rounding up functions respectively.

2.2 Optimization in Machine Learning

Many machine learning problems reduce to optimization problems. That is, we start with some parameterized function:

$$f : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$

some finite training dataset:

$$S = \{(x_{a_i}, y_i) \mid x_{a_i} \in \mathbb{R}^a, y_i \in \mathbb{R}^b\}$$

and some loss function:

$$l : \mathbb{R}^b \times \mathbb{R}^b \to \mathbb{R}$$

and we want to find a parameter vector $x_p \in \mathbb{R}^p$ to minimize:

$$l_f : \mathbb{R}^p \to \mathbb{R}$$
$$l_f(x_p) = \sum_{(x_{a_i}, y_i) \in S} l(f(x_p, x_{a_i}), y_i)$$

2.2.1 Gradient Based Optimization

If f and l are both differentiable we can minimize l_f with the gradient descent algorithm:

Definition 2.12. *Suppose $l_f : \mathbb{R}^p \to \mathbb{R}$ is a differentiable function, $x_p \in \mathbb{R}^p$ is the starting point, and $\alpha \in (0, 1]$ is the learning rate. Then the gradient descent algorithm is defined as follows:*

1: ***procedure*** GradientDescent(l_f, x_p, α)
2: *Repeat until convergence:*
3: $x_p \leftarrow x_p - \alpha \nabla l_f(x_p)$
4: *Return* x_p

If l_f is a convex function and α is small enough, then gradient descent is guaranteed to converge to the global minimum of l_f (Boyd and Vandenberghe, 2004). Even when l_f is not convex, gradient descent is often still useful. For example, under relatively mild conditions we can show that taking small enough gradient descent steps will never increase the value of any differentiable $l_f : \mathbb{R}^p \to \mathbb{R}$ (Boyd and Vandenberghe, 2004). The modern field of deep learning consists largely of applying gradient descent and other algorithms that can be efficiently computed with reverse mode automatic differentiation to optimize non-convex functions (LeCun et al., 2015).

When l_f is heavily nonlinear, it can be the case that gradient descent can overshoot minima or require an extremely small α to consistently move in the right direction. In these cases it can be helpful to rescale the gradient based on the local curvature of the function before taking the gradient step (Boyd and Vandenberghe, 2004). A particularly powerful way to do this is to use the inverse Hessian.

Definition 2.13. *Suppose $l_f : \mathbb{R}^p \to \mathbb{R}$ is a smooth function and $x_p \in \mathbb{R}^p$ is the starting point. Then the Newton's method algorithm is defined as follows:*

1: ***procedure*** NewtonsMethod(l_f, x_p)
2: *Repeat until convergence:*

3: $x_p \leftarrow x_p - \nabla^2(l_f(x_p))^{-1}\nabla l_f(x_p)$

4: *Return* x_p

$\nabla^2(l_f(x_p))^{-1}$ *is the inverse Hessian of* l_f *at* x_p. *Recall that the Hessian matrix of* $l_f : \mathbb{R}^p \to \mathbb{R}$ *at the point* $x_p \in \mathbb{R}^p$ *is the following square* $p \times p$ *matrix:*

$$\nabla^2 l_f(x_p) = \begin{bmatrix} \frac{\partial^2 l_f}{\partial x[1]\partial x[1]}(x_p) & \dots & \frac{\partial^2 l_f}{\partial x[1]\partial x[p]}(x_p) \\ \vdots & \ddots & \\ \frac{\partial^2 l_f}{\partial x[p]\partial x[1]}(x_p) & & \frac{\partial^2 l_f}{\partial x[p]\partial x[p]} \end{bmatrix}$$

where $\frac{\partial^2 l_f}{\partial x[i]\partial x[j]}$ *is the mixed partial derivative of the function* l_f *in the* i*th and* j*th components of its input vector.*

One of the challenges with using Newton's method at scale is that inverting the Hessian can be an expensive computation. This is especially true when l_f is a large function with many parameters, such as a deep neural network. In these cases there are other strategies that we can utilize to smooth the trajectory of gradient descent.

For example, it can be helpful to track the previous gradient steps and use them in the computation of the next step. One of the simplest ways to do this is the momentum algorithm, which uses a placeholder variable to track the value of the previous update steps (Polyak, 1964):

Definition 2.14. *Suppose* $l_f : \mathbb{R}^p \to \mathbb{R}$ *is a differentiable function,* $x_p \in \mathbb{R}^p$ *is the starting point,* $\alpha \in (0, 1]$ *is the learning rate, and* $\beta \in (0, 1]$ *is the momentum parameter. Then the momentum algorithm is defined as follows:*

1: ***procedure*** MOMENTUM(l_f, x_p, α, β)
2: $\quad x'_p \leftarrow 0$
3: $\quad$ *Repeat until convergence:*
4: $\quad\quad x'_p \leftarrow x'_p - \beta x'_p - \beta\nabla l_f(x_p)$
5: $\quad\quad x_p \leftarrow x_p + \alpha x'_p$
6: $\quad$ *Return* x_p

We can visualize this algorithm as simulating the momentum of a ball rolling down a hill. We use α for both the learning rate and momentum parameter.

While momentum is a powerful tool for learning from noisy data, we often see even better performance by tracking previous updates at the individual parameter level. For example, the Adagrad algorithm uses a placeholder variable to reweight updates based on the magnitude of previous updates (Duchi et al., 2011):

Definition 2.15. *Suppose* $l_f : \mathbb{R}^p \to \mathbb{R}$ *is a differentiable function,* $x_p \in \mathbb{R}^p$ *is the starting point, and* $\alpha \in (0, 1]$ *is the learning rate. Then the Adagrad algorithm is defined as follows:*

1: ***procedure*** ADAGRAD(l_f, x_p, α)
2: $\quad x'_p \leftarrow 0$
3: $\quad$ *Repeat until convergence:*
4: $\quad\quad x'_p \leftarrow x'_p + \nabla l_f(x_p)^2$
5: $\quad\quad x_p \leftarrow x_p - \alpha\frac{\nabla l_f(x_p)}{\sqrt{x'_p}}$
6: $\quad$ *Return* x_p

The division in $\frac{\nabla l_f(x_p)}{\sqrt{x'_p}}$ *is performed elementwise.*

2.2.2 Efficient Computation of the Gradient

Each of the optimization approaches we discuss in Section 2.2.1 optimize a function $l_f : \mathbb{R}^p \to \mathbb{R}$ by repeatedly computing its gradient $\nabla l_f : \mathbb{R}^p \to \mathbb{R}^p$. In order to use these strategies to optimize a neural network we need a strategy to compute this gradient efficiently. In the case that:

$$l_f(x_p) = \sum_{(x_{a_i}, y_i) \in S} l(f(x_p, x_{a_i}), y_i)$$

we have that:

$$\begin{aligned}
&\nabla l_f(x_p) \\
=& \quad \{\text{Definition of } l_f\} \\
&\nabla \sum_{(x_{a_i}, y_i) \in S} l(f(x_p, x_{a_i}), y_i) \\
=& \quad \{\text{Gradient distributes over addition}\} \\
&\sum_{(x_{a_i}, y_i) \in S} \nabla l(f(x_p, x_{a_i}), y_i)
\end{aligned}$$

Now suppose we have a parameter vector:

$$x_p = (x_{p_1}, x_{p_2}, \cdots, x_{p_{n-1}}, x_{p_n}) \in \mathbb{R}^p$$

and neural network:

$$\begin{aligned}
&f : \mathbb{R}^p \to \mathbb{R}^b \\
&f(x_p, x_a) = f_n(x_{p_n}, f_{n-1}(x_{p_{n-1}}, f_{n-2}(x_{p_{n-2}}, \cdots, f_1(x_{p_1}, x_a))))
\end{aligned}$$

where each:

$$f_k : \mathbb{R}^{p_k} \times \mathbb{R}^{a_k} \to \mathbb{R}^{a_{k+1}}$$

corresponds to a layer in f. In this case we can use the following strategy to compute the gradient of each layer's parameter vector efficiently.

Definition 2.16. *The backpropagation algorithm (Rumelhart et al., 1986) is a special case of reverse mode automatic differentiation (Rall, 1981) in which we efficiently compute the gradient of the loss of the n-layer neural network:*

$$l(f(x_p, x_a), y) = l(f_n(x_{p_n}, f_{n-1}(x_{p_{n-1}}, f_{n-2}(x_{p_{n-2}}, \cdots, f_1(x_{p_1}, x_a)))), y)$$

by repeatedly applying the chain rule to cache the computation of parameter gradients as we go along and compute the full neural network gradient in a single pass.

2.2.3 Stochastic Optimization

Given some set:

$$S = \{(x_{a_i}, y_i) \mid x_{a_i} \in \mathbb{R}^a, y_i \in \mathbb{R}^b\}$$

and functions:

$$l : \mathbb{R}^b \times \mathbb{R}^b \to \mathbb{R} \qquad f : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$

it may be that S is too large to compute the gradient of the following function:

$$l_f(x_p) = \sum_{(x_{a_i}, y_i) \in S} l(f(x_p, x_{a_i}), y_i)$$

and apply the algorithms described in Section 2.2.1. In this case we can apply a stochastic approach in which we iteratively update x_p based on the gradients of each $l(f(x_p, x_{a_i}), y_i)$. For example, consider the following algorithm:

Definition 2.17. *Suppose $l : \mathbb{R}^b \times \mathbb{R}^b \to \mathbb{R}$ and $f : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ are differentiable functions,*

$$S = \{(x_{a_i}, y_i) \mid x_{a_i} \in \mathbb{R}^a, y_i \in \mathbb{R}\}$$

is a set of points, $x_p \in \mathbb{R}^p$ is the starting point, and $\alpha \in (0, 1]$ is the learning rate. Then the stochastic gradient descent algorithm is defined as follows:

1: ***procedure*** *StochasticGradientDescent(l, f, S, x_p, α)*
2: *Repeat until convergence:*
3: $i \leftarrow mod(i + 1, |S|)$
4: $l_{f_i} \leftarrow l(f(\_, x_{a_i}), y_i)$
5: $x_p \leftarrow x_p - \alpha \nabla l_{f_i}(x_p)$
6: *Return* x_p

Definition 2.18. *Suppose $l : \mathbb{R}^b \times \mathbb{R}^b \to \mathbb{R}$ and $f : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ are differentiable functions,*

$$S = \{(x_{a_i}, y_i) \mid x_{a_i} \in \mathbb{R}^a, y_i \in \mathbb{R}\}$$

is a set of points, $x_p \in \mathbb{R}^p$ is the starting point, $\alpha \in (0, 1]$ is the learning rate, and $\beta \in (0, 1]$ is the momentum parameter. Then the stochastic momentum algorithm is defined as follows:

1: ***procedure*** *StochasticMomentum($l, f, S, x_p, \alpha, \beta$)*
2: $x'_p \leftarrow 0$
3: *Repeat until convergence:*
4: $i \leftarrow mod(i + 1, |S|)$
5: $l_{f_i} \leftarrow l(f(\_, x_{a_i}), y_i)$
6: $x'_p \leftarrow x'_p - \beta x'_p - \beta \nabla l_{f_i}(x_p)$
7: $x_p \leftarrow x_p + \alpha x'_p$
8: *Return* x_p

The other algorithms in Section 2.2.1 can be adapted similarly. For more on stochastic optimization see "Convex Optimization" (Boyd and Vandenberghe, 2004).

A major benefit of stochastic gradient descent is that we can optimize the function l_f without fitting S in memory. Furthermore, some recent research has also suggested that models trained with stochastic gradient descent may generalize better than models trained with gradient descent (Smith et al., 2020).

2.3 Probability

Definition 2.19. *A σ-algebra Σ on the set Ω is a set of subsets of Ω that includes Ω and is closed under (1) countable union (2) countable intersection (3) complement in Ω.*

Definition 2.20. *A measurable space is a pair (Ω, Σ) of a set Ω and a σ-algebra Σ on Ω.*

There is a natural notion of a morphism that we can define between measurable spaces.

Definition 2.21. *A function $f : A \to B$ from the measurable space (A, Σ_A) to the measurable space (B, Σ_B) is measurable if for any $\sigma_B \in \Sigma_B$, $f^{-1}(\sigma_B) \in \Sigma_A$.*

Given two measurable spaces we can take their product in a canonical way.

Definition 2.22. *Given measurable spaces (Ω, Σ_Ω), $(\Omega', \Sigma_{\Omega'})$ we can form a σ-algebra $\Sigma_\Omega \times \Sigma_{\Omega'}$ on the set $\Omega \times \Omega'$ by taking all countable unions and complements of subsets in $\{\sigma_\Omega \times \sigma_{\Omega'} \mid \sigma_\Omega \in \Sigma_\Omega, \sigma_{\Omega'} \in \Sigma_{\Omega'}\}$. We call $\Sigma_\Omega \times \Sigma_{\Omega'}$ the product σ-algebra of Σ_Ω and $\Sigma_{\Omega'}$ and we call the measurable space $(\Omega \times \Omega', \Sigma_\Omega \times \Sigma_{\Omega'})$ the product measurable space of $(\Omega, \Sigma_\Omega), (\Omega', \Sigma_{\Omega'})$.*

We can also form measurable spaces from topological spaces.

Definition 2.23. *Given the topological space Ω the Borel σ-algebra $\mathcal{B}(\Omega)$ of Ω is the σ-algebra generated by the collection of open subsets of Ω. We call the measurable space $(\Omega, \mathcal{B}(\Omega))$ a Borel measurable space.*

Said another way, the Borel σ-algebra of Ω is the smallest σ-algebra of Ω that contains all open sets in Ω.

The fundamental objects in measure-theoretic probability are the probability measure and probability space:

Definition 2.24. *A probability measure $\mu : \Sigma \to [0, 1]$ over the measurable space (Ω, Σ) is a countably additive function over the σ-algebra Σ that returns results in the unit interval $[0, 1]$ such that $\mu(\Omega) = 1, \mu(\emptyset) = 0$. Recall that Σ is a set of subsets of Ω.*

Definition 2.25. *A probability space is a triplet (Ω, Σ, μ) where (Ω, Σ) is a measurable space and μ is a probability measure over (Ω, Σ).*

Probability spaces are closed under products. This is, when (Ω, Σ, μ) and (Ω', Σ', μ') are probability spaces the product space $(\Omega \times \Omega', \Sigma \times \Sigma', \mu\mu')$ where $\mu\mu'(\omega) = \mu(\omega)\mu'(\omega)$ is also a probability space.

A particularly useful tool for working with measurable functions and probability spaces is Lebesgue integration.

Definition 2.26. *Suppose $f : X \to \mathbb{R}$ is a measurable and non-negative real-valued function and $\mu : \Sigma_X \to [0, 1]$ is a probability measure. For some $\sigma_x \in \Sigma_X$ we can define:*

$$f^{\sigma_x}(t) = \mu(\{x \in \sigma_x, f(x) > t\})$$

and we can define the Lebesgue integral $\int_{x \in \sigma_x} f(x) d\mu$ in terms of the improper Riemann integral:

$$\int_{x \in \sigma_x} f(x) d\mu = \int_0^\infty f^{\sigma_x}(t) dt$$

When f is not necessarily non-negative we can define its Lebesgue integral as the difference of its positive and negative parts.

2.3.1 Markov Kernels, Random Variables, and Stochastic Processes

We will frequently work with parameterized probability measures, which we call Markov kernels.

Definition 2.27. *A Markov kernel from the measurable space (A, Σ_A) to the measurable space (B, Σ_B) is a function $\mu : A \times \Sigma_B \to [0, 1]$ such that:*

- *For all $\sigma_b \in \Sigma_B$, the function $\mu(\_, \sigma_b) : A \to [0, 1]$ is measurable.*
- *For all $x_a \in A$, $\mu(x_a, \_) : \Sigma_B \to [0, 1]$ is a probability measure on (B, Σ_B). In particular:*

$$\mu(x_a, B) = 1 \qquad \mu(x_a, \emptyset) = 0.$$

For example, a Markov Kernel from the one-point set to the measurable space (A, Σ_A) is just a probability measure over (A, Σ_A).

Another foundational object in measure-theoretic probability is the random variable, which is ironically neither random nor a variable.

Definition 2.28. *A random variable defined on the probability space (Ω, Σ, μ) is a measurable function from (Ω, Σ) to $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$.*

We will sometimes use the term "random variable" to refer to measurable functions into $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ as well. These are also called multivariate random variables or random vectors. While some authors use uppercase letters like X to denote random variables, we will use lowercase letters like f, g to emphasize that random variables are functions.

Random variables and probability measures are closely related.

Definition 2.29. *Given a probability space (Ω, Σ, μ) and a random variable $f : \Omega \to \mathbb{R}$, the pushforward $f_*\mu$ of μ along f is a probability measure over $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ defined to be:*

$$f_*\mu : \mathcal{B}(\mathbb{R}) \to [0, 1]$$
$$f_*\mu(\sigma_{\mathbb{R}}) = \mu(f^{-1}(\sigma_{\mathbb{R}})).$$

Like probability measures, random variables have a parameterized extension.

Definition 2.30. *A stochastic process defined in the probability space $(\Omega, \Sigma_\Omega, \mu)$ is a family of random variables indexed by points in some measurable space (T, Σ_T). We can write a stochastic process as a function:*

$$f : \Omega \times T \to \mathbb{R}$$

In this work we exclusively consider stochastic processes that are measurable in the product σ-algebra $\Sigma_\Omega \times \Sigma_T$.

Stochastic processes and Markov kernels are closely related.

Definition 2.31. *Given a stochastic process:*

$$f : \Omega \times T \to \mathbb{R}$$

over the probability space $(\Omega, \Sigma_\Omega, \mu)$ we define the pushforward of μ along f to be the following Markov Kernel from (T, Σ_T) to (Ω, Σ_Ω):

$$f_*\mu : T \times \mathcal{B}(\mathbb{R}) \to [0, 1]$$
$$f_*\mu(x_t, \sigma_{\mathbb{R}}) = f(\_, x_t)_*\mu(\sigma_{\mathbb{R}}) = \mu(f(\_, x_t)^{-1}(\sigma_{\mathbb{R}})).$$

For more on Markov kernels, random variables, and stochastic processes see "Probability Theory: A Comprehensive Course" (Klenke, 2013).

2.3.2 Categories in Probability

Measurable spaces and measurable functions form a symmetric monoidal category as follows:

Definition 2.32. *The objects in* **Meas** *are pairs* (A, Σ_A)*, where* Σ_A *is a* σ*-algebra over* A*, and morphisms are measurable functions.*

The tensor product of the measurable spaces (A, Σ_A) *and* (B, Σ_B) *in* **Meas** *is the product measurable space* $(A \times B, \Sigma_A \times \Sigma_B)$ *(Definition 2.22) and the tensor product of the measurable functions* g, f *is the function* $(g \otimes f)(x, y) = (g(x), f(y))$*.*

Proposition 2.33. *The strict symmetric monoidal category* **EucMeas** *is a subcategory of* **Meas** *in which objects are restricted to be* $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ *for some* $n \in \mathbb{N}$*, the tensor of the objects* $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ *and* $(\mathbb{R}^m, \mathcal{B}(\mathbb{R}^m))$ *is* $(\mathbb{R}^{n+m}, \mathcal{B}(\mathbb{R}^{n+m}))$ *and morphisms are restricted to be infinitely differentiable.*

Proof. **EucMeas** contains all identities since the identity function $f(x_n) = x_n$ is infinitely differentiable for all n. Next, given two infinitely differentiable measurable maps g, f the composition $g \circ f$ and tensor $(g \otimes f)(x, y) = (g(x), f(y))$ are infinitely differentiable and measurable as well. Therefore morphisms in **EucMeas** are closed under composition and tensor. Next, since the tensor product of the measurable spaces $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ and $(\mathbb{R}^m, \mathcal{B}(\mathbb{R}^m))$ in **Meas** is the measurable spaces $(\mathbb{R}^{n+m}, \mathcal{B}(\mathbb{R}^{n+m}))$ we have that objects in **EucMeas** are closed under tensor as well.

Next, in order to show that **EucMeas** is strict monoidal we need to show that the associators and unitors in **EucMeas** are identities. First, since:

$$(\mathbb{R}^{n+(m+k)}, \mathcal{B}(\mathbb{R}^{n+(m+k)})) = (\mathbb{R}^{(n+m)+k}, \mathcal{B}(\mathbb{R}^{(n+m)+k}))$$

the associators in **EucMeas** are identities. Next, since:

$$(\mathbb{R}^{0+n}, \mathcal{B}(\mathbb{R}^{0+n})) = (\mathbb{R}^{n+0}, \mathcal{B}(\mathbb{R}^{n+0})) = (\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$$

the unitors in **EucMeas** are identities.

□

Another important category that we will consider is **Stoch** (Lawvere, 1962; Giry, 1982).

Definition 2.34. *In the symmetric monoidal category* **Stoch** *objects are measurable spaces and morphisms are Markov kernels. We define the composition of the Markov kernels* $\mu : A \times \Sigma_B \to [0, 1]$ *and* $\mu' : B \times \Sigma_C \to [0, 1]$ *to be the following, where* $x_a \in A$ *and* $\sigma_c \in \Sigma_C$*:*

$$(\mu' \circ \mu) : A \times \Sigma_C \to [0, 1]$$
$$(\mu' \circ \mu)(x_a, \sigma_c) = \int_{x_b \in B} \mu'(x_b, \sigma_c) d\mu(x_a, \_).$$

The identity morphism at (A, Σ_A) *is* δ *where for* $x_a \in A, \sigma_a \in \Sigma_A$*:*

$$\delta : A \times \Sigma_A \to [0, 1]$$
$$\delta(x_a, \sigma_a) = \begin{cases} 1 & x_a \in \sigma_a \\ 0 & x_a \notin \sigma_a \end{cases}.$$

The tensor product of the Markov Kernels $\mu : A \times \Sigma_B \to [0,1]$ *and* $\mu' : C \times \Sigma_D \to [0,1]$ *in* **Stoch** *is the Markov Kernel:*

$$(\mu \otimes \mu') : (A \times C) \times (\Sigma_B \times \Sigma_D) \to [0,1]$$

where $\Sigma_B \times \Sigma_D$ *is the product* σ*-algebra and for* $x_a \in A, x_c \in C, \sigma_b \in \Sigma_B, \sigma_d \in \Sigma_D$*:*

$$(\mu \otimes \mu')((x_a, x_c), \sigma_b \times \sigma_d) = \mu(x_a, \sigma_b)\mu'(x_c, \sigma_d).$$

Note that by the π-λ-theorem the action of $(\mu \otimes \mu')((x_a, x_c), \_)$ on any set in $\Sigma_B \times \Sigma_D$ is determined by its action on rectangles $\sigma_b \times \sigma_d$ (Fritz, 2020).

The objects in **Stoch** are also equipped with a commutative comonoidal structure that is compatible with the monoidal tensor in **Stoch**. That is, each object $X \in$ **Stoch** is equipped with a comultiplication map $\mathsf{cp} : X \to X \otimes X$ and a natural counit map $\mathsf{del} : X \to *$ that commute with the unitors and associators in **Stoch** and further satisfy:

$$\mathsf{cp}_{X \otimes Y} = (id_X \otimes \sigma_{X,Y} \otimes id_Y)(\mathsf{cp}_X \otimes \mathsf{cp}_Y),$$

where $\sigma_{X,Y} : X \times Y \to Y \times X$ is the symmetric swap map in **Stoch**.

One subcategory of **Stoch** that we will work closely with is the following:

Definition 2.35. EucStoch *is the subcategory of* **Stoch** *in which objects are restricted to be* $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ *for some* $n \in \mathbb{N}$ *and the tensor of the objects* $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ *and* $(\mathbb{R}^m, \mathcal{B}(\mathbb{R}^m))$ *is* $(\mathbb{R}^{n+m}, \mathcal{B}(\mathbb{R}^{n+m}))$.

For more details on categories in probability see (Fritz, 2020).

2.4 Structural Representations

Given a set X, there are many ways to represent the relationships between the points in X. We explore some of these structures in this section.

2.4.1 Metric Spaces

We can represent a dataset as a finite set of points and distances. There are two slightly different constructions that we use:

Definition 2.36. *A metric space* (X, d_X) *is a tuple of a set* X *and a function* $d_X : X \times X \to \mathbb{R}_{\geq 0}$ *such that:*

- $d_X(x_1, x_2) = d_X(x_2, x_1)$
- $d_X(x, x) = 0$
- $d_X(x_1, x_2) + d_X(x_2, x_3) \geq d_X(x_1, x_3)$
- $\forall x_1 \neq x_2, d_X(x_1, x_2) > 0$

Definition 2.37. *A ubermetric space* (X, d_X) *is a tuple of a set* X *and a function* $d_X : X \times X \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ *such that:*

- $d_X(x_1, x_2) = d_X(x_2, x_1)$
- $d_X(x, x) = 0$

- $d_X(x_1, x_2) + d_X(x_2, x_3) \geq d_X(x_1, x_3)$

where for all $x \in \mathbb{R}_{\geq 0}$ *we have:*

$$x \leq \infty \qquad \infty + x = x + \infty = \infty$$

Ubermetric spaces (Spivak, 2012) are a relaxation of metric spaces that admit infinite distances and nonidentical points with distance 0.

There is a natural notion of a morphism between these spaces.

Definition 2.38. *A nonexpansive map from* (X, d_X) *to* (Y, d_Y) *is a function* $f : X \to Y$ *such that:*

$$d_Y(f(x_1), f(x_2)) \leq d_X(x_1, x_2)$$

We can use nonexpansive maps to form categories of these spaces.

Definition 2.39. *In the category* **Met** *objects are finite metric spaces and morphisms are nonexpansive maps.*

Definition 2.40. *In the category* **UMet** *objects are finite ubermetric spaces and morphisms are nonexpansive maps.*

We will frequently consider the subcategories of **Met** and **UMet** in which morphisms are restricted to functions of a certain type. For example:

Definition 2.41. $\mathbf{Met}_{inj}$ *is the subcategory of* **Met** *in which morphisms are restricted to injective nonexpansive maps.*

Definition 2.42. $\mathbf{Met}_{sur}$ *is the subcategory of* **Met** *in which morphisms are restricted to surjective nonexpansive maps.*

Definition 2.43. $\mathbf{Met}_{bij}$ *is the subcategory of* **Met** *in which morphisms are restricted to bijective nonexpansive maps.*

Definition 2.44. $\mathbf{Met}_{isom}$ *is the subcategory of* **Met** *in which morphisms are restricted to isometries (distance preserving isomorphisms)*

$\mathbf{Met}_{isom}$ is a subcategory of $\mathbf{Met}_{bij}$, which is in turn a subcategory of both $\mathbf{Met}_{sur}$ and $\mathbf{Met}_{inj}$.

One more useful preorder subcategory of **Met** is as follows:

Definition 2.45. $\mathbf{Met}_{id}$ *is the subcategory of* **Met** *in which the morphisms from* (X, d_X) *to* (Y, d_Y) *are limited to inclusion functions* $\iota(x) = x$. $\mathbf{Met}_{id}$ *is a preorder and we write:*

$$(X, d_X) \leq_{\mathbf{Met}_{id}} (Y, d_Y)$$

to indicate that $X \subseteq Y$ *and that the inclusion map* $\iota : (X, d_X) \hookrightarrow (Y, d_Y)$ *is nonexpansive.*

If **D** is a subcategory of $\mathbf{Met}_{id}$ then we write

$$(X, d_X) \leq_{\mathbf{D}} (Y, d_Y)$$

to indicate that the inclusion map $\iota : (X, d_X) \hookrightarrow (Y, d_Y)$ is a morphism in **D**.

2.4.2 Simplicial Complexes

In this work we use the abstract definition of a simplicial complex. For more details on these definitions we recommend (Chazal and Michel, 2017).

Definition 2.46. *A simplicial complex is a family of finite sets that is closed under taking subsets.*

Definition 2.47. *The faces of a simplicial complex are the finite sets in that complex.*

Definition 2.48. *The n-simplices of a simplicial complex are its subcomplexes that contain all of the subsets of a single $(n+1)$-element face in the complex.*

Intuitively, 0-simplices are points, 1-simplices are pairs of points, 2-simplices are triples of intersecting 1-simplices that we can visualize as triangles, etc.

Definition 2.49. *A vertex of a simplicial complex is a 0-simplex in that complex. The underlying set X of a simplicial complex S_X is its set of vertices.*

Any n-simplex in a simplicial complex is completely determined by its set of vertices. For example, if S_X is the powerset of $\{1,2,3\}$ then the vertices in S_X are $1,2,3$ and the underlying set of S_X is $\{1,2,3\}$.

Definition 2.50. *A finite simplicial complex is a simplicial complex S_X with a finite underlying set X.*

Simplicial complexes are also equipped with a natural notion of a morphism:

Definition 2.51. *A simplicial map from a simplicial complex S_X with underlying set X to a simplicial complex S_Y with underlying set Y is a function $f : X \to Y$ such that if $x_1, x_2, \cdots, x_n$ span a face of S_X, then $f(x_1), f(x_2), \cdots, f(x_n)$ span a face of S_Y.*

We can build on this definition to construct a category of simplicial complexes:

Definition 2.52. *The category* **SCpx** *has finite simplicial complexes as objects and simplicial maps as morphisms.*

A particularly important class of simplicial complexes is the class determined by graphs:

Proposition 2.53. *Given a graph G with vertex set X, the collection of cliques in G is a simplicial complex with underlying set X. We call a simplicial complex that can be constructed in this way a flag complex.*

Proof. Define S_X to be the collection of cliques in G and consider any clique $S \in S_X$. By the definition of a clique, for any subset $S' \subseteq S$ there must be an edge between each pair of vertices in S'. Therefore S' is a clique as well. Since S_X is a family of finite sets that is closed under taking subsets it is therefore a simplicial complex. □

Given a flag complex generated by the graph G, the 0-simplices in the complex are the vertices in G, the 1-simplices are the pairs of vertices connected by an edge in G and the n-simplices are the n-element cliques in G.

Given a finite metric space (X, d_X), there are a number of ways that we can form a finite simplicial complex with underlying set X. For example:

Definition 2.54. *Given a finite metric space (X, d_X) and a choice of $\delta \in \mathbb{R}_{\geq 0}$, the δ-Vietoris-Rips complex is the simplicial complex whose faces are the subsets of X with all pairwise distances no greater than δ.*

If $\delta \leq \delta'$, then the set of n-simplices of the δ-Vietoris-Rips complex is a subset of the set of n-simplices of δ'-Vietoris-Rips complex. This invites the following definition:

Definition 2.55. *A filtration of simplicial complexes is a sequence of complexes:*

$$S_{X_1} \subseteq S_{X_2} \subseteq \cdots$$

where $S_{X_i} \subseteq S_{X_j}$ implies that all of the simplices in S_{X_i} are also simplices in S_{X_j}.

Given a finite metric space (X, d_X), any sequence $\delta_1 \leq \delta_2 \leq \cdots$ therefore induces a filtration of Vietoris-Rips complexes.

It is often more convenient to work with points $a \in (0, 1]$ rather than $\delta \in \mathbb{R}_{\geq 0}$.

Definition 2.56. *The category $(0, 1]^{op}$ is the total order in which objects are $a \in (0, 1]$ and morphisms are defined by the relation $\geq$.*

In order to convert distances $\delta \in \mathbb{R}_{\geq 0}$ to points $a \in (0, 1]^{\text{op}}$ we can use the functor:

$$-\log : (0, 1]^{\text{op}} \to \mathbb{R}_{\geq 0}$$

Now given a finite metric space (X, d_X), any sequence $a_1 \geq a_2 \geq \cdots$ of points in $(0, 1]^{\text{op}}$ therefore induces a filtration of Vietoris-Rips complexes under the transformation $-\log(a_1) \leq -\log(a_2) \leq \cdots$

Definition 2.57. *We say that a functor:*

$$F_X : (0, 1]^{op} \to \mathbf{C}$$

commutes with the functor $U : \mathbf{C} \to \mathbf{Set}$ if there exists some set X such that for any $a \in (0, 1]^{op}$ we have:

$$(U \circ F_X)(a) = X$$

and for any $a' \geq a \in (0, 1]^{op}$ each component of $(U \circ F_X)(a' \geq a)$ is the identity function $f(x) = x$ on the set X.

We can express a filtration indexed by the set $(0, 1]^{\text{op}}$ as follows:

Definition 2.58. *A fuzzy simplicial complex with underlying set X is a functor*

$$F_X : (0, 1]^{op} \to \mathbf{SCpx}$$

where F_X commutes with the forgetful functor $U : \mathbf{SCpx} \to \mathbf{Set}$ and $(U \circ F_X)(a) = X$.

An example of a fuzzy simplicial complex is a functor that maps each $a \in (0, 0.5]$ to the powerset of $\{1, 2, 3\}$ and maps each $a \in (0.5, 1]$ to the simplicial complex $\{\{1\}, \{2\}, \{3\}\}$.

Fuzzy simplicial complexes are closely related to Spivak's fuzzy simplicial sets (Spivak, 2012). Note also that in a fuzzy simplicial complex F_X for all $a \in (0, 1]^{\text{op}}$ the underlying set of the simplicial complex $F_X(a)$ is the underlying set of F_X.

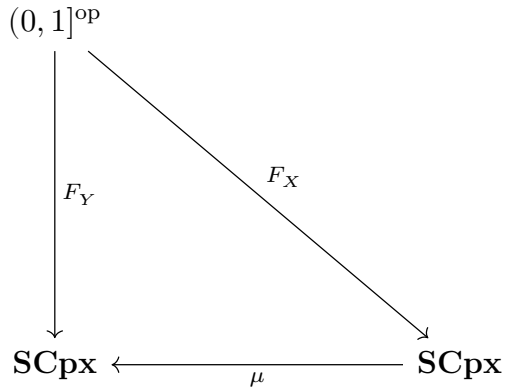
The functoriality of F_X then implies that for $a, a' \in (0, 1]$ such that $a' \geq a$, if the n-simplex σ is in $F_X(a')$ then σ is also in $F_X(a)$.

Definition 2.59. *The strength of the n-simplex σ in the fuzzy simplicial complex F_X is the largest a in $(0,1]$ such that σ is in $F_X(a)$. If no such a exists then the strength of σ is 0.*

We can build on these definitions to form a category of fuzzy simplicial complexes:

Definition 2.60. *The objects in the category* **FSCpx** *are fuzzy simplicial complexes (i.e. functors). The morphisms in* **FSCpx** *are natural transformations.*

The components of the natural transformations in **FSCpx** are simplicial maps (Definition 2.51) and for any such natural transformation μ from the fuzzy simplicial complex $F_X : (0,1]^{\text{op}} \to \mathbf{SCpx}$ to the fuzzy simplicial complex $F_Y : (0,1]^{\text{op}} \to \mathbf{SCpx}$ the following diagram commutes:



That is, μ is defined by a function from the underlying set of F_X to the underlying set of F_Y.

2.4.3 Partitions and Covers

There are two other types of subset structures that are important to introduce: partitions and nonnested flag covers.

Definition 2.61. *A partition $\mathbb{P}_X$ of the set X is a nonoverlapping cover of X.*

Definition 2.62. *A nonnested flag cover $\mathcal{C}_X$ of the set X is a cover of X such that:*

- *If $S_1, S_2 \in \mathcal{C}_X$ and $S_1 \subseteq S_2$, then $S_1 = S_2$*
- *There exists a flag complex with underlying set X and faces all finite subsets of the sets in $\mathcal{C}_X$.*

All partitions are also nonnested flag covers. However, unlike in a partition the sets in a nonnested flag cover $\mathcal{C}_X$ may overlap.

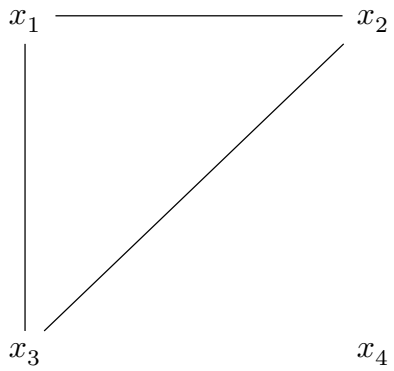
We can always upgrade any cover of a finite set to a nonnested flag cover of that set:

Definition 2.63. *Given a cover $\mathcal{C}_X$ of a finite set X, the flagification of $\mathcal{C}_X$ is the nonnested flag cover we get by iteratively adjoining to $\mathcal{C}_X$ any clusters mandated by the flag condition, and then removing all the nonmaximal ones (Culbertson et al., 2016).*

For example, consider the following set and cover:

$$X = \{x_1, x_2, x_3, x_4\}$$
$$\mathcal{C}_X = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_1, x_3\}, \{x_4\}\}$$

$\mathcal{C}_X$ is nonnested, but it is not a flag cover. To see this, consider the graph whose edges are defined by the pairs of points that share a cluster in $\mathcal{C}_X$:



This graph has a clique $\{x_1, x_2, x_3\}$ that is not a cluster in $\mathcal{C}_X$. The flagification of $\mathcal{C}_X$ is therefore:

$$\{\{x_1, x_2, x_3\}, \{x_4\}\}$$

which we obtain by adjoining $\{x_1, x_2, x_3\}$ to $\mathcal{C}_X$ and then removing the nonmaximal clusters $\{x_1, x_2\}, \{x_2, x_3\}, \{x_1, x_3\}$.

Nonnested flag covers and partitions are equipped with a natural notion of a morphism:

Definition 2.64. *A consistent map from the nonnested flag cover $\mathcal{C}_X$ of X to the nonnested flag cover $\mathcal{C}_Y$ of Y is a function $f : X \to Y$ such that for any set $S_X \in \mathcal{C}_X$ there exists some set $S_Y \in \mathcal{C}_Y$ such that $f(S_X) \subseteq S_Y$ (Culbertson et al., 2016).*

Consistent maps are not exactly the same thing as graph homomorphisms. The map that sends all vertices in X to a single point y in Y is always consistent since there must exist some set $S_Y \in \mathcal{C}_Y$ that contains y. However, this map is not necessarily a graph homomorphism unless the graphs we construct are assumed to be equipped with loops at each point.

We can build on this definition to form categories of nonnested flag covers and partitions.

Definition 2.65. *In the category* **Cov***, objects are tuples $(X, \mathcal{C}_X)$ where $\mathcal{C}_X$ is a nonnested flag cover of the finite set X. The morphisms in* **Cov** *from $(X, \mathcal{C}_X)$ to $(Y, \mathcal{C}_Y)$ are consistent maps $f : X \to Y$.*

For example, the tuples $(\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}\})$ and $(\{a, b\}, \{\{a, b\}\})$ are objects in **Cov**, and the function:

$$f(1) = a \qquad f(2) = b \qquad f(3) = b$$

is a morphism between them.

Definition 2.66. *In the category* **Part** *objects are tuples* $(X, \mathbb{P}_X)$ *where* $\mathbb{P}_X$ *is a partition of the finite set* X*. The morphisms in* **Part** *from* $(X, \mathbb{P}_X)$ *to* $(Y, \mathbb{P}_Y)$ *are consistent maps* $f : X \to Y$*.*

Part is a subcategory of **Cov**. One particularly useful subcategory of **Part** is as follows:

Definition 2.67. $\mathbf{Part}_{id}$ *is the subcategory of* **Part** *in which the morphisms from* $(X, \mathbb{P}_X)$ *to* $(Y, \mathbb{P}_Y)$ *are limited to inclusion functions* $\iota(x) = x$*.* $\mathbf{Part}_{id}$ *is a preorder and we write:*

$$(X, \mathbb{P}_X) \leq_{\mathbf{Part}_{id}} (Y, \mathbb{P}_Y)$$

to indicate that $X \subseteq Y$ *and that the inclusion map* $\iota : (X, \mathbb{P}_X) \hookrightarrow (Y, \mathbb{P}_Y)$ *is consistent.*

We often need to reason about structure that exists at multiple scales. In this case it can be helpful to work with fuzzy nonnested flag covers:

Definition 2.68. *A fuzzy nonnested flag cover of the set* X *is a functor:*

$$F_X : (0, 1]^{op} \to \mathbf{Cov}$$

where F_X *commutes with the forgetful functor* $U : \mathbf{Cov} \to \mathbf{Set}$ *and* $(U \circ F_X)(a) = X$*.*

Fuzzy nonnested flag covers generalize nonnested flag covers, since we can represent a nonnested flag cover $\mathcal{C}_X$ of X with a fuzzy nonnested flag cover F_X such that $F_X(a) = \mathcal{C}_X$ for all a.

We can also form a fuzzy analog of partitions:

Definition 2.69. *A fuzzy partition of the set* X *is a functor:*

$$F_X : (0, 1]^{op} \to \mathbf{Part}$$

where F_X *commutes with the forgetful functor* $U : \mathbf{Part} \to \mathbf{Set}$ *and* $(U \circ F_X)(a) = X$*.*

Intuitively, a fuzzy nonnested flag cover or fuzzy partition of X represents how the points in X may be grouped together with different strengths.

Definition 2.70. *The strength of a subset* $S \subseteq X$ *in the fuzzy nonnested flag cover or fuzzy partition* F_X *of the set* X *is the largest* $a \in (0, 1]$ *such that there exists some set* S' *in* $F_X(a)$ *where* $S \subseteq S'$*. If no such* a *exists then the strength of* S *is* 0*.*

We can form categories of fuzzy nonnested flag covers and fuzzy partitions.

Definition 2.71. *The objects in the category* **FCov** *are fuzzy nonnested flag covers (i.e. functors). The morphisms in* **FCov** *are natural transformations.*

Definition 2.72. *The objects in the category* **FPart** *are fuzzy partitions (i.e. functors). The morphisms in* **FPart** *are natural transformations.*

The components of the morphisms in **FPart** and **FCov** are consistent maps.

2.5 Experimental Background

Most of the work in this dissertation is theoretical. However, several results are accompanied by illustrative experiments. In this section we describe some of the datasets and metrics we use.

2.5.1 Datasets

Many of the experiments in this dissertation are performed over simulated data. However, some experiments use publicly available datasets.

The Fashion MNIST dataset contains about $60,000$ images of clothing. Each image is represented as a 28×28 grayscale image (784 features). Each image is also bucketed into one of 10 possible classes, including "shirt", "pants", etc.

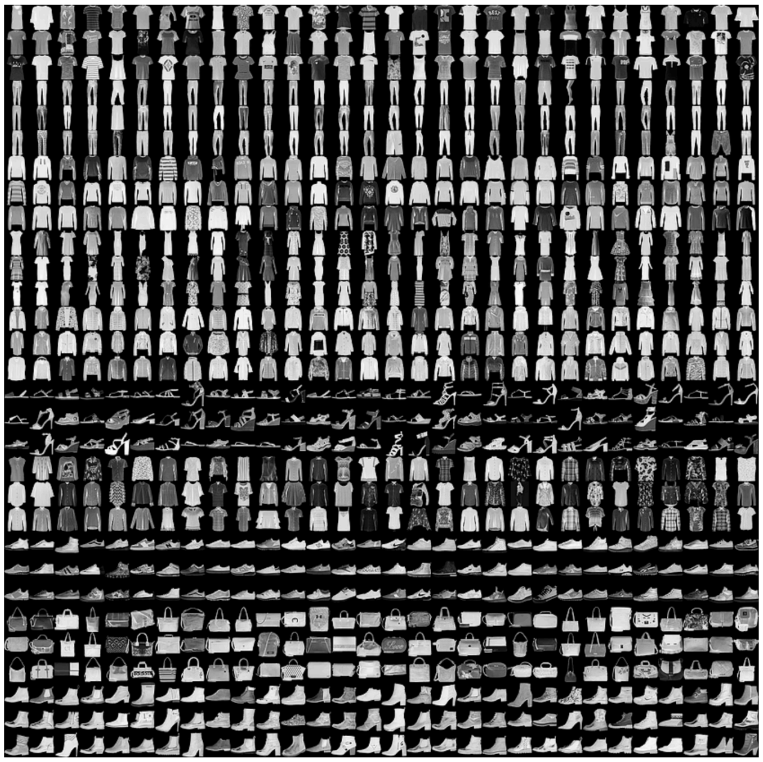


Figure 3: *The Fashion MNIST dataset (Xiao et al., 2017)*

The 20 Newsgroups (Lang, 1995) dataset contains about $18,000$ newsgroups posts, each of which is bucketed into one of 20 topics. We use the scikit-learn (Pedregosa et al., 2011) version of this dataset.

2.5.2 Metrics

Given a set X in which each sample has a label in $\{\text{false}, \text{true}\}$ we can evaluate the performance of a classifier:

$$f : X \to \{\text{false}, \text{true}\}$$

with the following metrics:

Definition 2.73. *The true positive rate of a classifier is the proportion of all true samples which the classifier correctly labels as true. This is also known as recall or sensitivity. The true positive rate is* 1.0 *if and only if there are no false negatives.*

Definition 2.74. *The true negative rate of a classifier is the proportion of all false samples which the classifier correctly labels as false. This is also known as specificity. The true negative rate is* 1.0 *if and only if there are no false positives.*

Next, given two partitions of a finite set X we often want to measure how well they agree. In order to do this we can use the Rand score:

Definition 2.75. *The Rand score (Rand, 1971; Pedregosa et al., 2011) between the partitions $\mathbb{P}_X, \mathbb{P}'_X$ of the set X is a number between 0 and 1 defined to be the ratio:*

$$RI(\mathbb{P}_X, \mathbb{P}'_X) = \frac{|both(\mathbb{P}_X, \mathbb{P}'_X)| + |neither(\mathbb{P}_X, \mathbb{P}'_X)|}{|X|^2}$$

where:

$$both(\mathbb{P}_X, \mathbb{P}'_X) = \{x_i, x_j \mid \exists s_X \in \mathbb{P}_X, x_i, x_j \in s_X \wedge \exists s'_X \in \mathbb{P}'_X, x_i, x_j \in s'_X\}$$
$$neither(\mathbb{P}_X, \mathbb{P}'_X) = \{x_i, x_j \mid \nexists s_X \in \mathbb{P}_X, x_i, x_j \in s_X \wedge \nexists s'_X \in \mathbb{P}'_X, x_i, x_j \in s'_X\}$$

The value of the Rand score is heavily dependent on the number of clusters (sets in the partition), which can make it difficult to interpret. Therefore, in practice we usually work with an adjusted variant of the Rand score that is close to 0 for a random partition and is exactly 1 for identical partitions.

Definition 2.76. *Suppose $\mathbf{P}^2_X$ is the set of all pairs of partitions of the set X and $\mu_{\mathbf{P}^2_X}$ is the uniform distribution over $\mathbf{P}^2_X$. Then the adjusted Rand score (Hubert and Arabie, 1985; Pedregosa et al., 2011) between the partitions $\mathbb{P}_X, \mathbb{P}'_X$ of the set X is the ratio:*

$$ARI(\mathbb{P}_X, \mathbb{P}'_X) = \frac{RI(\mathbb{P}_X, \mathbb{P}'_X) - E_{\mu_{\mathbf{P}^2_X}}[RI]}{\max_{\mathbf{P}^2_X}(RI) - E_{\mu_{\mathbf{P}^2_X}}[RI]}$$

where $RI(\mathbb{P}_X, \mathbb{P}'_X)$ is the Rand score between the partitions $\mathbb{P}_X, \mathbb{P}'_X$.

The adjusted Rand score is a number between 0 and 1. This score tracks the number of pairs of samples that are assigned to the same or different clusters in $\mathbb{P}_X$ and $\mathbb{P}'_X$. An adjusted Rand score close to 0 means that the cluster assignments are no more similar than random assignments would be and an adjusted Rand score of 1 means that the cluster assignments are identical up to permutation.

3 Literature Review

3.1 Gradient Based Learning

One of the subareas of machine learning that lends itself most nicely to categorification is gradient based learning. Gradients have a number of interesting compositional structures that allows them to feature prominently in a wide variety of machine learning applications.

Category theoretic gradient based learning researchers generally represent machine learning models as morphisms $f : P \otimes A \to B$ in some monoidal category $(\mathbf{C}, \otimes, *)$. The objects P, A, B respectively represent a space of parameters, observed data, or model predictions. For example, if our goal is to train a 1000 parameter model that classifies 28×28 pixel images into two classes, we might have $P = \mathbb{R}^{1000}$, $A = \mathbb{R}^{28\times28}$ and $B = [0, 1]$, with the latter representing a probability [1].

Training such a model consists of finding a specific parameter value $\theta : * \to P$ (a 1000 element vector in our image classification example), such that we can form the trained model $f_\theta : A \to B = f \circ (\theta \otimes id_A)$.

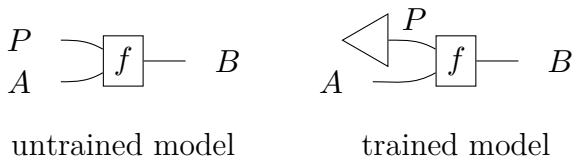


Figure 4: *Trained and untrained model information flow*

In essence, the parameters P serve to index a collection of maps, and so searching for a map $f \circ (\theta \otimes id_A) : A \to B$ reduces to searching for a value $\theta : * \to P$. Much of the research in category theoretic gradient based learning comes from one of the following perspectives:

- **Computing the Gradient**: Gradient based optimization is ubiquitous in machine learning–especially neural networks–so computing the gradient efficiently is key. In this section, we discuss categorical approaches to this computation.
- **Parameterized Maps and Lenses**: Lenses can represent the forward 'predictive' and backward 'update' behaviors of learning.
- **Updates and Learning**: Formalizing the update step allows us to formalize the process of learning.

In this section we outline each of these in turn.

3.1.1 Computing the Gradient

Gradient descent is an ubiquitous approach for training machine learning models where one views learning a model $f : P \otimes A \to B$ as iteratively improving some initial guess of parameters $\theta : * \to P$ in order to minimize some choice of 'loss' function. The gradient of this loss function is interpreted as the direction of steepest ascent: gradient descent

[1] Since pixels are not actually real-valued, we may instead use $A = F^{28\times28}$ or $A = U^{28\times28}$ where F is the set of all floating point numbers and U is the set of all unsigned integers.

makes repeated steps in the negative direction of the gradient to converge on a local minimum. It is important that the computation of the gradient is efficient, since it must be recomputed at each of the many iterations made by gradient descent.

A particularly important algorithm for efficiently computing the gradient of a loss function with respect to some parameters is backpropagation (Definition 2.16). The first examination of backpropagation in a categorical setting is the "Backprop as functor" paper by Fong et al. (2019), which we will explore in more detail in Section 3.1.3.

Instead, we begin in Section 3.1.1.1 with a discussion of Cartesian differential categories, which categorify the notion of a differential operator. However, we will see that this is not quite what we need to train a machine learning model via gradient descent: for this, we need a reverse differential operator, which we discuss in Section 3.1.1.2.

3.1.1.1 Cartesian Differential Categories

Some authors have begun to explore axiomatic perspectives on differentiation. For example, Blute et al. (2006) explore how an additive symmetric monoidal category with a coalgebra modality and differential combinator that satisfies a collection of coherence conditions can generalize the notion of derivatives of smooth maps. Blute et al. (2009) further adapt this construction to form Cartesian differential categories, which explicitly characterize the notion of an infinitely differentiable category. Cartesian differential categories are equipped with a differential combinator D which sends a map $f : A \to B$ to a generalized derivative map $D[f] : A \times A \to B$.

Cartesian differential categories are defined in terms of left-additive structure, which we first recall.

Definition 3.1. *A Cartesian left additive category is a Cartesian category* $\mathbf{C}$ *in which the hom-set of each pair of objects* B, C *is a commutative monoid, with addition operation* $+$ *and zero maps (additive identities)* $0_{BC} : B \to C$*, such that:*

- *For any morphism* $h : A \to B$ *and morphism* $f, g : B \to C$ *we have:*

$$(f + g) \circ h = (f \circ h) + (g \circ h) : A \to C$$
$$0_{BC} \circ h = 0_{AC} : A \to C$$

- *For any object* $C = C_1 \times C_2 \times \cdots \times C_n$*, projection map* $\pi_i : C \to C_i$ *and morphisms* $f, g : B \to C$ *we have:*

$$\pi_i \circ (f + g) = (\pi_i \circ f) + (\pi_i \circ g) : B \to C_i$$
$$\pi_i \circ 0_{BC} = 0_{BC_i} : B \to C_i$$

We write 0_A *for the additive identity of the hom-set* $\mathbf{C}[*, A]$*.*

Intuitively, in a Cartesian left additive category we can add morphisms in a way that is compatible with postcomposition and the Cartesian structure. Certain Cartesian left additive categories are equipped with additional structure that behaves similarly to derivatives:

Definition 3.2. *A Cartesian differential category* $\mathbf{C}$ *is a Cartesian left-additive category equipped with a Cartesian differential combinator* D *that assigns to each morphism* $f : A \to B$ *in* $\mathbf{C}$ *a morphism* $D[f] : A \times A \to B$ *in* $\mathbf{C}$ *such that* D *satisfies the following equations (Definition 4 in Cockett et al. (2019), adapted from Definition 2.1.1 in (Blute et al., 2009)):*

CDC.1 $D[f+g] = D[f] + D[g]$ *and* $D[0] = 0$*;*

CDC.2 $D[f] \circ \langle a, b+c\rangle = D[f] \circ \langle a, b\rangle + D[f] \circ \langle a, c\rangle$ *and* $D[f] \circ \langle a, 0\rangle = 0$*;*

CDC.3 $D[id] = \pi_1, D[\pi_0] = \pi_0 \circ \pi_1$ *and* $D[\pi_1] = \pi_1 \circ \pi_1$*;*

CDC.4 $D[\langle f, g\rangle] = \langle D[f], D[g]\rangle$*;*

CDC.5 $D[g \circ f] = D[g] \circ \langle f \circ \pi_0, D[f]\rangle$*;*

CDC.6 $D[D[f]] \circ \langle\langle a, b\rangle, \langle 0, c\rangle\rangle = D[f] \circ \langle a, c\rangle$*;*

CDC.7 $D[D[f]] \circ \langle\langle a, b\rangle, \langle c, d\rangle\rangle = D[D[f]] \circ \langle\langle a, c\rangle, \langle b, d\rangle\rangle$.

CDC.5 is the chain rule that defines the differential operation on composite maps. This has the following information flow:

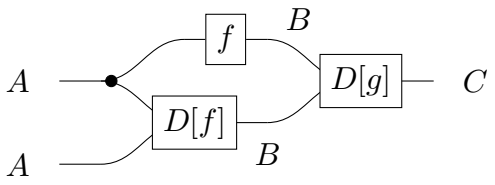


Figure 5: *The chain rule in a Cartesian differential category*

A familiar example of a Cartesian differential category is the category **Euc** (Proposition 2.6) of infinitely differentiable maps between Euclidean spaces:

For a map $f : \mathbb{R}^a \to \mathbb{R}^b$ in **Euc**, the Cartesian differential $D[f] : \mathbb{R}^a \times \mathbb{R}^a \to \mathbb{R}^b$ is the following smooth function:

$$D[f](x_a, x'_a) = J_f(x_a) \cdot x'_a = \begin{bmatrix} \frac{\partial f_1}{\partial x[1]}(x_a) & \cdots & \frac{\partial f_1}{\partial x[a]}(x_a) \\ \vdots & \ddots & \\ \frac{\partial f_b}{\partial x[1]}(x_a) & & \frac{\partial f_b}{\partial x[a]}(x_a) \end{bmatrix} \cdot x'_a$$

$D[f]$ is linear in its second argument, and it maps a vector of coefficients $x_a \in \mathbb{R}^a$ and a vector of values $x'_a \in \mathbb{R}^a$ to the projection of x'_a along the Jacobian of f at x_a.

Intuitively $D[f]$ describes the following linear approximation of f:

$$f(x_a + x'_a) \approx f(x_a) + D[f](x_a, x'_a)$$

Since the Jacobian of a linear function $l(x) = Ax$ is simply the matrix A, we have the following for any linear function l in **Euc**:

$$l(x + x') = l(x) + D[l](x, x')$$

Blute et al. (2009) use this property to generalize the notion of linearity:

Definition 3.3. *A morphism $f : A \to B$ in the Cartesian differential category* **C** *is linear when $D[f] = f \circ \pi_1$, where π_1 is the right projection map.*

This is a generalization of the idea that the derivative of a linear map is the map itself. Since linear maps are preserved under composition and tensor, any Cartesian differential category contains a subcategory of linear maps.

3.1.1.2 Reverse Derivative Categories

Although Cartesian differential categories give a suitably generalized definition of the derivative, they do not provide quite what we need for gradient based learning. Consider the following supervised learning scenario, where we have:

- A parameterized model $f : P \times A \to B$
- A training example $(a, b) : * \to A \times B$
- A choice of parameters $\theta : * \to P$

Intuitively speaking, we would like to use our training data (a, b) to compute some new parameter $\hat{\theta}$ such that $f(\hat{\theta}, a)$ is closer to b than $f(\theta, a)$. The Cartesian differential of f gives us a morphism:

$$D[f] : (P \times A) \times (P \times A) \to B$$

but what we actually want is a morphism of type:

$$(P \times A) \times B \to P \times A$$

This is precisely the type of the reverse derivative combinator introduced by Cockett et al. (2019):

Definition 3.4. *A Cartesian reverse derivative category is a Cartesian left-additive category* $\mathbf{C}$ *equipped with a Cartesian reverse derivative combinator* R *that assigns to each morphism* $f : A \to B$ *in* $\mathbf{C}$ *a morphism* $R[f] : A \times B \to A$ *in* $\mathbf{C}$ *such that* R *satisfies the following equations (Definition 13 of (Cockett et al., 2019)):*

RD.1 $R[f + g] = R[f] + R[g]$ *and* $R[0] = 0$*;*

RD.2 $R[f] \circ \langle a, b + c \rangle = R[f] \circ \langle a, b \rangle + R[f] \circ \langle a, c \rangle$ *and* $R[f] \circ \langle a, 0 \rangle = 0$*;*

RD.3 $R[id] = \pi_1, R[\pi_0] = \iota_0 \circ \pi_1$ *and* $R[\pi_1] = \iota_1 \circ \pi_1$*;*

RD.4 $R[\langle f, g \rangle] = R[f] \circ (id_A \times \pi_0) + R[g] \circ (id_A \times \pi_1)$ *and* $R[!_A] = 0$*;*

RD.5 $R[g \circ f] = R[f] \circ (id_A \times R[g]) \circ \langle \pi_0, \langle f \circ \pi_0, \pi_1 \rangle \rangle$*;*

RD.6 $\pi_1 \circ R[R[R[f]]] \circ (\iota_0 \times id_B) \circ \langle id_A \times \pi_0, 0_A \times \pi_1 \rangle = R[f] \circ (id_A \times \pi_1)$*;*

RD.7

$$\pi_1 \circ R[R[\pi_1 \circ R[R[f]] \circ (\iota_0 \times id_B)]] \circ (\iota_0 \times id_B) =$$
$$\langle \pi_0 \times \pi_0, \pi_1 \times \pi_1 \rangle \circ \pi_1 \circ R[R[\pi_1 \circ R[R[f]] \circ (\iota_0 \times id_B)]] \circ (\iota_0 \times id_B)$$

where:

$$\iota_0 : A \to A \times B$$
$$\iota_0 = \langle id_A, 0_B \rangle$$

and:

$$\iota_1 :: B \to A \times B$$
$$\iota_1 = \langle 0_A, id_B \rangle$$

are the Cartesian injection maps (Cockett et al., 2019).

The conditions **RD.1** to **RD.7** mirror the properties of the derivative operation. For example, R must distribute over addition (**RD.1**) and compose according to a chain rule (**RD.5**) that has the following information flow:

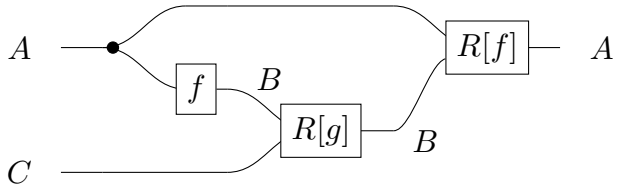


Figure 6: *The chain rule in a Cartesian reverse derivative category*

Intuitively, **RD.5** captures the flow of information of backpropagation: given a point A and output C, information flows backwards through $R[g]$ and then $R[f]$ to compute a change in the initial inputs, A.

Returning to our previous example, **Euc** also forms a reverse derivative category. For a map $f : \mathbb{R}^a \to \mathbb{R}^b$ in **Euc**, the reverse derivative $R[f] : \mathbb{R}^a \times \mathbb{R}^b \to \mathbb{R}^a$ is the following smooth function:

$$R[f](x_a, x'_b) = J_f(x_a)^T \cdot x'_b = \begin{bmatrix} \frac{\partial f_1}{\partial x[1]}(x_a) & \cdots & \frac{\partial f_b}{\partial x[1]}(x_a) \\ \vdots & \ddots & \\ \frac{\partial f_1}{\partial x[a]}(x_a) & & \frac{\partial f_b}{\partial x[a]}(x_a) \end{bmatrix} \cdot x'_b$$

Like the differential combinator $D[f]$, note that $R[f]$ is also linear in its second argument and describes the following linear approximation:

$$f(x_a) + x'_b \approx f(x_a + R[f](x_a, x'_b))$$

Another important example of Cartesian reverse derivative categories are polynomial ring categories.

Definition 3.5. *Given a commutative ring r we can form the category $\mathbf{Poly}_r$ in which objects are natural numbers and the morphisms from n to m are tuples of m polynomials with n variables and coefficients in r. That is, a morphism $P : n \to m$ is a map $P(x) = (p_1(x), \cdots, p_m(x))$ where $x = (x[1], \cdots, x[n])$ and $p_i(x)$ is a polynomial in x.*

The composition of morphisms in $\mathbf{Poly}_r$ is given by the standard composition of polynomials and the tensor of objects is given by addition (Cockett et al., 2019).

$\mathbf{Poly}_r$ is a Cartesian reverse derivative category in which the terminal object $*$ is 0 and the reverse derivative of:

$$P : n \to m$$
$$P(x) = (p_1(x), \cdots, p_m(x))$$

is:

$$R[P] : n \times m \to n$$
$$R[P](x, y') = \left(\sum_{i=1}^{m} \frac{\partial p_i}{\partial x[1]}(x) y'[i], \cdots, \sum_{i=1}^{m} \frac{\partial p_i}{\partial x[n]}(x) y'[i] \right)$$

where $x[i]$ is the ith component of the vector x and $\frac{\partial p_i}{\partial x[j]}(x)$ is the formal derivative of the polynomial p_i taken with respect to the jth component $x[j]$ of the vector x (Cockett et al., 2019).

By Theorem 16 in (Cockett et al., 2019), every Cartesian reverse derivative category $\mathbf{C}$ is also a Cartesian differential category where for any morphism $f : A \to B$ in $\mathbf{C}$:

$$D[f] : A \times A \to B$$
$$D[f] = \pi_1 \circ R[R[f]] \circ (\langle id_A, 0_{AB} \rangle \times id_A)$$

Therefore, we can construct the linear maps in a Cartesian reverse derivative category from the linear maps in the associated Cartesian differential category.

In a Cartesian reverse derivative categories these linear maps have additional structure that can be useful to explore. In particular, the linear maps of $\mathbf{C}$ form a subcategory $\mathbf{L}$ of $\mathbf{C}$ equipped with a stationary on objects involution:

$$(\_)^\dagger : \mathbf{L}^{op} \to \mathbf{L}$$

such that for any linear map f in $\mathbf{L}$ we have $R[f] = f^\dagger \circ \pi_1$ (Cockett et al., 2019).

For example, the linear maps in **Euc** are exactly the linear maps in the traditional sense, and given a linear map $f : \mathbb{R}^a \to \mathbb{R}^b$ in **Euc** where $f(x) = Mx$ we have $f^\dagger : \mathbb{R}^b \to \mathbb{R}^a$ where $f^\dagger(x) = M^T x$. That is, $\dagger$ is a generalization of the transpose. Similarly, the linear maps in $\mathbf{Poly}_r$ are those that can be expressed as:

$$P(x) = (p_1(x), p_2(x), \cdots, p_m(x))$$

where:

$$p_i(x) = \sum_{i=1}^{n} r_i x[i]$$

for $r_i \in r$, where $x[i]$ is the ith component of the vector x (Cockett et al., 2019).

The flow of information in Cartesian reverse derivative categories is very close to the flow of information in lenses (2.1.1). For example, recall the flow of information in the composition of lens put maps (Section 2.1.1):

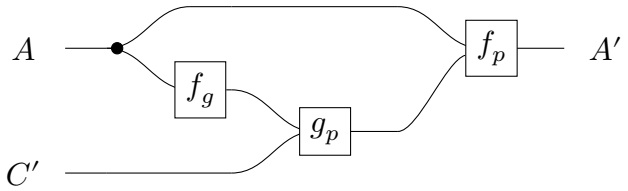


Figure 7: *Composition of put maps*

This is remarkably similar to the flow of information in reverse derivative composition (axiom **RD.5** of reverse differential categories, Figure 6).

3.1.1.3 Automatic Differentiation

Elliott (2018) also takes a generalized perspective on differentiation, but from a different angle. Whereas Cockett et al. (2019) explicitly generalize the information flow in differentiation and then demonstrate that backpropagation fits into their framework, Elliott explores the automatic differentiation algorithm itself.

Elliott (2018) writes from a functional programming perspective, and the author's generalization of automatic differentiation is essentially a program specification. His central construction is a framework for differentiable functional programming that relies on a typed higher-order partial function that tracks and computes derivatives of programs. Elliott (2018) defines efficient implementations of derivatives by using a program transformation strategy.

Elliott (2018) writes $a \multimap b$ to indicate the type of linear maps from a to b and then defines a derivative to be an operator of the form:

$$\mathsf{D} : (a \to b) \to (a \to (a \multimap b))$$

that satisfies:

- The chain rule: for all functions f, g we have: $\mathsf{D}(g \circ f)a = \mathsf{D}g(fa) \circ \mathsf{D}fa$
- The cross rule: for all functions f, g we have: $\mathsf{D}(f \times g)(a, b) = \mathsf{D}fa \times \mathsf{D}gb$
- The linear rule: for all linear functions f we have, $\mathsf{D}fa = f$

All of these theorems are satisfied by the classical notion of derivatives, and Elliott demonstrates the generality of his construction by relying only on these properties. In order to enable the composition of derivatives, he also defines the operator:

$$\mathsf{D}^+ f = (f, \mathsf{D}f)$$

which acts as an endofunctor over a category of differentiable functions.

From the perspective of Elliott's construction, the difference between reverse mode and forward mode automatic differentiation reduces to the difference between left/right associated compositions of derivatives. This allows Elliott to define an implementation of reverse mode differentiation from a continuation-passing style (Wand, 1980) rather than dealing with a gradient tape (an additional data structure to store intermediate gradient computations). Furthermore, because the differentiation operator decorates the inference function, the construction by Elliott (2018) explicitly specifies how computation is shared in reverse mode automatic differentiation. This is a key aspect of why reverse mode automatic differentiation is particularly efficient for gradient based learning.

3.1.2 Parameterized Maps and Lenses

In practice, gradient based learning is often used to optimize the parameters of a machine learning model. A categorical tool that has risen in prominence for representing these parameters is the **Para** operator. This operator has several presentations (Capucci et al., 2021; Fong et al., 2019; Gavranovic, 2019; Cruttwell et al., 2021). A simple definition is as follows:

Definition 3.6. *Let* $\mathbf{C}$ *be a strict symmetric monoidal category. Then* $\mathbf{Para}(\mathbf{C})$ *is a category with the same objects as* $\mathbf{C}$*. A morphism* $A \to B$ *in* $\mathbf{Para}(\mathbf{C})$ *is a pair* (P, f) *where* P *is an object of* $\mathbf{C}$ *and:*

$$f : P \otimes A \to B$$

is a morphism in $\mathbf{C}$*. The composition of morphisms:*

$$f : P \otimes A \to B \qquad g : Q \otimes B \to C$$

is given by $(Q \otimes P, g \circ (id_Q \otimes f))$*:*

$$(Q \otimes P) \otimes A \xrightarrow{=} Q \otimes (P \otimes A) \xrightarrow{id_Q \otimes f} Q \otimes B \xrightarrow{g} C$$

Consider a morphism $(P, f) : A \to B$ in $\mathbf{Para}(\mathbf{C})$. This is a map from A to B with an extra input P that has to be accounted for. The composition of two parameterized maps $f : P \otimes A \to B$ and $g : Q \otimes B \to C$ collects the parameters into the monoidal tensor, as shown in the figure below:

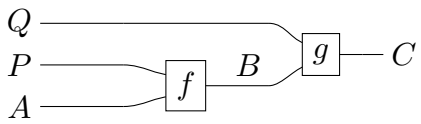


Figure 8: *Composition of arrows in* $\mathbf{Para}(\mathbf{C})$

The **Para** construction captures the idea that the P inputs of a machine learning model $f : P \otimes A \to B$ are the values to be learned. In the next subsection we explore how authors have built on **Para** to describe these machine learning models.

3.1.2.1 Learners

Much of the categorical perspective on neural networks derives from ideas first discussed by Fong et al. (2019). The authors demonstrate that machine learning models, which they call learners, can be explicitly constructed as the morphisms in a category. This construction is quite concrete and built on top of the category **Set**, and it is therefore somewhat independent from the generalized categorical constructions related to differentiation outlined in the previous sections.

Fong et al. (2019) start by introducing the learner, or a tuple $A \xrightarrow{(P,I,U,r)} B$ where P is a set (the parameter space) and I, U, and r are functions with types:

$$I : P \times A \to B$$
$$U : P \times A \times B \to P$$
$$r : P \times A \times B \to A$$

The maps I, U, and r are called the implementation, update, and request maps, respectively.

The authors also define the category **Learn** in which objects are sets, morphisms are learners, and the composition of the learners:

$$(P, I, U, r) : A \to B$$
$$(Q, J, V, s) : B \to C$$

is:

$$(P \times Q, J \circ_{\mathbf{Para}(\mathbf{Euc})} I, V \circ_{\mathbf{Para}(\mathbf{Euc})} U, r \circ_{\mathbf{Para}(\mathbf{Euc})} s) : A \to C$$

where $\circ_{\mathbf{Para}(\mathbf{Euc})}$ is composition in $\mathbf{Para}(\mathbf{Euc})$ and therefore:

$$(J \circ_{\mathbf{Para}(\mathbf{Euc})} I)(p, q, a) = J(q, I(p, a))$$
$$(V \circ_{\mathbf{Para}(\mathbf{Euc})} U)(p, q, a, c) = U(p, a, s(q, I(p, a), c)), V(q, I(p, a), c)$$
$$(r \circ_{\mathbf{Para}(\mathbf{Euc})} s)(p, q, a, c) = r(p, a, s(q, I(p, a), c))$$

Fong et al. (2019) use this structure to describe the backpropagation algorithm as a functor from a slight variation of $\mathbf{Para}(\mathbf{Euc})$ to **Learn**.

3.1.2.2 Generalizing Learners with Lenses

Fong and Johnson (2019) note that a simple way to cast a learner:

$$A \xrightarrow{(P,I,U,r)} B$$

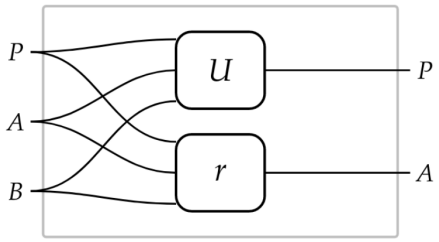
as a lens:

$$\binom{A}{A} \xrightarrow{(f_g, f_p)} \binom{B}{B}$$

is to write:

$$f_g = I \qquad f_p = \langle U, r \rangle$$

where $\langle U, r \rangle$ is the combined update-request morphism:



However, this transformation does not preserve composition. In order to mitigate this Fong and Johnson (2019) turn to symmetric lenses, or equivalence classes of spans of lenses:

$$A \xleftarrow{(f_g, f_p)} S_1 \xrightarrow{(f'_g, f'_p)} B$$

and demonstrate that we can define a faithful identity-on-objects monoidal functor from **Learn** to the category of symmetric lenses.

One of the benefits of this perspective is that we can apply lens theory to the study of machine learning. For example, two canonical properties that a lens (f_g, f_p) in **Set** could satisfy are the GetPut and PutGet laws:

- A lens $\binom{A}{A} \xrightarrow{(f_g, f_p)} \binom{B}{B}$ in **Set** satisfies GetPut when for $a \in A$:

  $$f_p(a, f_g(a)) = a$$

- A lens $\binom{A}{A} \xrightarrow{(f_g, f_p)} \binom{B}{B}$ in **Set** satisfies PutGet when for $a \in A, b \in B$:

$$f_g(f_p(a, b)) = b$$

Intuitively, we can think of (f_g, f_p) as a pair of a database get function f_g and a database put function f_p. In this case PutGet states that putting data into the database and then taking it out does not change the data. Similarly, GetPut states that taking data out of the database and then putting it back in does not change anything.

Both laws can be interpreted in the machine learning context as well. If the lens associated with a learner satisfies GetPut, then that learner's update function will not change the parameter values when the learner correctly classifies an example. If the lens associated with a learner satisfies PutGet, then that learner will output the "correct" value for a sample after it sees it. This is unrealistic for most neural network models, but is a common property of models with infinite VC dimension (Vapnik and Chervonenkis, 2015), such as non-parametric algorithms like KNN (K-nearest neighbors).

Cruttwell et al. (2021) take a different perspective and use reverse derivative categories to generalize the learners of Fong et al. (2019) as lenses. Given a reverse derivative category $\mathbf{C}$ the authors introduce a category $\mathbf{Lens}(\mathbf{C})$ in which morphisms are $\mathbf{C}$-lenses. They then demonstrate that there exists a functor $R : \mathbf{C} \to \mathbf{Lens}(\mathbf{C})$ that sends the morphism f in $\mathbf{C}$ to the lens $(f, R[f])$ in $\mathbf{Lens}(\mathbf{C})$.

Cruttwell et al. (2021) then demonstrate that the application of **Para** to $\mathbf{Lens}(\mathbf{C})$ yields a category $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ in which the morphisms, which the authors call parametric lenses, are a generalization of learners. Furthermore, the functor $R : \mathbf{C} \to \mathbf{Lens}(\mathbf{C})$ can be lifted to a functor:

$$\mathbf{Para}(R) : \mathbf{Para}(\mathbf{C}) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$$

3.1.3 Updates and Learning

Machine learning in the wild consists of a diverse set of techniques for training models. In this section we discuss how category theory has been applied to give a theoretical foundation for some of these techniques.

3.1.3.1 Delayed Trace and Backpropagation-Through-Time

Sprunger and Katsumata (2019) explore a categorical perspective on training models with a notion of state. That is, suppose instead of training a model $f : P \times A \to B$ from examples $A \times B$, we instead wish to learn from time series data, where we have sequences of training points $\mathsf{List}(A \times B)$.

A recurrent neural network uses a stateful approach to model time series data. A useful component in such a stateful model is the delay gate, represented with i in Figure 9. Intuitively, the delay gate delays the model input until the next model iteration.

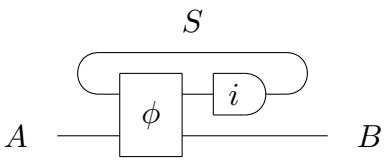


Figure 9: *The delay gate. A is the inputs, B is the outputs, and S is the state.*

Sprunger and Katsumata (2019) construct a category of these stateful models in which the Cartesian differential operator shares the same structure as the backpropagation through time algorithm that is used to train recurrent neural networks (Mozer, 1989). The authors represent stateful morphisms as commuting squares in a strict Cartesian category **C**. By vertically composing these commuting squares the authors construct stateful morphism sequences. They use these sequences to define a category in which objects are $\mathbb{N}$-indexed families of **C**-objects and morphisms are equivalence classes of stateful morphism sequences.

3.1.3.2 Learners' Languages

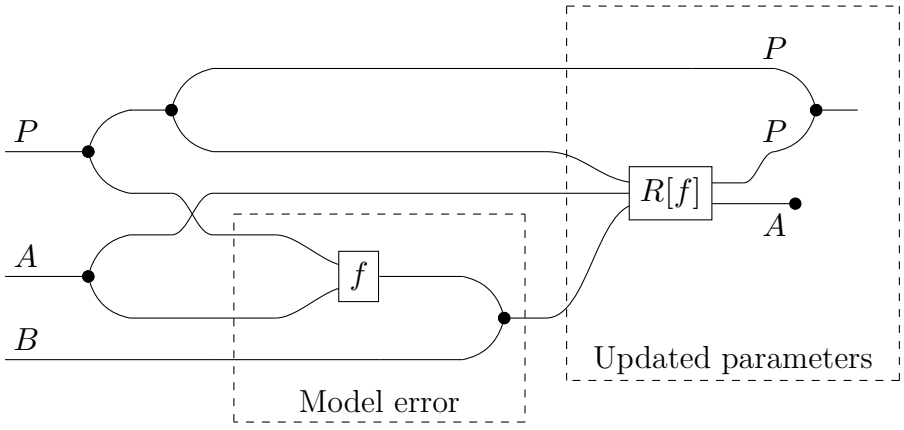
Spivak (2020) explores the connections between **Learn** and **Poly**, a category of polynomial functors in one variable. Spivak (2020) first shows that **Learn** is a subcategory of **Poly** and then uses this perspective to interpret learners as dynamical systems.

Spivak (2020) goes on to show that the space of learners between objects A and B forms a topos. This perspective makes it possible to consider logical propositions that can be stated in a learner's internal language. For example, this perspective makes it possible to formulate the response of the learner to receiving new data - taking a gradient descent update step - in categorical language.

3.1.3.3 Generalized Optimization Algorithms

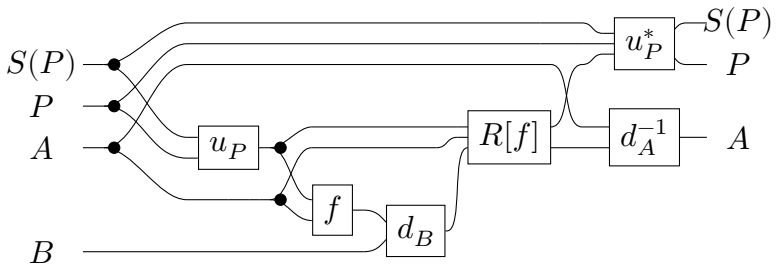
Some authors have begun to extend this generalized perspective on differentiation and derive new optimization algorithms built on top of the reverse derivative.

First, Wilson and Zanasi (2021) exploit the generality of the categorical reverse derivative (Section 3.1.1.2) to define an analogue of gradient based methods. Although their procedure has general applications, they focus on the special case of learning Boolean circuits. For a model $f : P \times A \to B$, they give the following map to update the model's parameters:



Next, the framework for categorical gradient based learning by Cruttwell et al. (2021)

extends this work. Their more general update step is as follows:



where d_A, d_B are error functions, which the authors call displacement maps. While the model f and reverse derivative $R[f]$ components are similar between the two works, the framework by Cruttwell et al. (2021) generalizes:

- The update maps that define how to update parameters given the gradient
- The displacement maps that define the error or distance between a model prediction and the true label

3.2 Categorical Probability

The research in this section focuses on understanding how the random processes underlying the data we model and the algorithms we use can be characterized and implemented in machine learning.

Unlike the research in Section 3.1.3 in which most learning is performed over categories with appropriate differential structure, in this section learning is performed over categories with appropriate probabilistic structure.

3.2.1 Background

The use of category theory in probabilistic machine learning can be divided into two areas. The first attempts to reformulate the core constructions of probability theory, including random variables, in categorical language. The second attempts to use this formalization to describe how learning works in a categorical setting.

The former has been extensively studied, going back to (Lawvere, 1962) and spanning dozens of papers. The latter is less popular, but has seen a number of papers in recent years studying causality (Fong, 2013), conjugate priors (Jacobs, 2017) and general aspects of Bayesian learning (Culbertson and Sturtz, 2014, 2013).

In this section we do not aim to examine all papers related to probability theory and category theory. Instead, we focus on papers that provide general principles most relevant to learning. This includes:

- **Synthetic Probability** The systemization of basic concepts from probability theory into an axiomatic framework. This enables understanding of joint distributions, marginalization, conditioning, Bayesian inverses, and more (Fritz, 2020; Fritz et al., 2020; Cho and Jacobs, 2019) purely in terms of the interactions of morphisms.
- **Probabilistic Machine Learning.** The study of updating a distribution with samples from a dataset and reasoning about uncertainty arising from noisy measurements (Culbertson and Sturtz, 2014, 2013; Jacobs, 2017).

3.2.2 Categorical Probability

The field of categorical probability aims to reformulate core components of probability theory on top of category theoretic constructions. This includes things such as composition of probabilistic maps, formation of joint probability distributions, marginalization, disintegration, independence and conditionals.

Several authors, including Cho and Jacobs (2019) and Fritz (2020) claim that categories with similar monoidal and comonoidal structures serve as the right abstraction for reasoning about probability. Each of these authors introduce frameworks within which we can describe common manipulations of joint probability distributions in terms of category theoretic operations. These constructions allow us to reason about complex probabilistic relationships in terms of algebraic axioms rather than low level analytical machinery.

One of the most important constructions in this group of work is the Markov category (Fritz, 2020), which generalizes a number of existing constructions:

Definition 3.7. *A Markov category is a symmetric monoidal category* $(\mathbf{C}, \otimes, 1)$ *in which every object* X *is equipped with a commutative comonoid structure (a comultiplication map* $\mathsf{cp} : X \to X \otimes X$ *and a counit map* $\mathsf{del} : X \to 1$*) satisfying coherence laws with the monoidal structure and where additionally the* del *map is natural with respect to every morphism* f*.*

This seemingly opaque definition will be dissected in the following subsections. We will see how each part plays an important role which is directly related to some aspect of the concept of randomness.

Before we move on however, let us take a moment to inspect a few characteristics of this definition. First, the naturality of del makes the monoidal unit terminal. Markov categories are therefore semicartesian. This is in contrast with CD-categories (Cho and Jacobs, 2019), which are identical other than this requirement. That is, CD-categories are a slightly more general construction, and Cho and Jacobs (2019) dub Markov categories affine CD-categories. Cho and Jacobs (2019) demonstrate that dropping the naturality condition permits a meaningful investigation of scalars (endomorphisms on the terminal object) and effects (morphisms into the terminal object). In contrast, the naturality of del tells us that there is only one way to delete information in Markov categories.

However, regardless of this condition the cp map is not necessarily natural with respect to every morphism in a Markov category. We remark on this in Section 3.2.2.2, where the naturality of a morphism with respect to cp implies the morphism is deterministic.

3.2.2.1 Distributions, Channels, Joint Distributions, and Marginalization

Markov categories are a synthetic framework for reasoning about random mappings. A morphism $p : * \to X$ from the monoidal unit in a Markov category can be thought of as distribution $p(x)$ over some object X, or simply as a random element of X. To understand how this map p truly acts as a distribution, or a random element, we need to understand how it interacts with the rest of the Markov category structure.

The fundamental component of a Markov category is a channel, or a morphism $f : X \to Y$. Authors often use the suggestive notation $f(y \mid x)$ to depict a channel, hinting at the role of the channel as a generalization of a probability distribution over Y parameterized by X. The composition of the channels $p : * \to X$ and f yields a morphism $f \circ p : * \to Y$ which can be interpreted causally: first, a random element of X

is generated, which is subsequently fed into f, outputting an element of Y. This allows us to interpret $f \circ p$ as a random element of Y.

A morphism $h : * \to X \otimes Y$ from the monoidal unit into the tensor of two objects gives us the notion of a joint distribution over X and Y. This joint distribution can then be marginalized over Y to produce a distribution over X. This is done with the help of the delete map, i.e. by composition of h with $id_X \otimes \mathsf{del}_Y : X \otimes Y \to X$. That is, we can say that the X-marginal over the channel $h : * \to X \otimes Y$ is:

$$(id_X \otimes \mathsf{del}_Y) \circ h : * \to X$$

Although composing the joint distribution h individually with the two projections:

$$(id_X \otimes \mathsf{del}_Y) : X \otimes Y \to X$$
$$(\mathsf{del}_X \otimes id_Y) : X \otimes Y \to Y$$

does give us two marginal distributions:

$$(id_X \otimes \mathsf{del}_Y) \circ h : * \to X$$
$$(\mathsf{del}_X \otimes id_Y) \circ h : * \to Y$$

this is a process that cannot generally be inverted. This makes intuitive sense: the space of joint distributions contains more information than the product of their marginals.

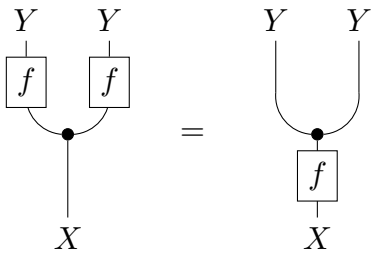
3.2.2.2 Deterministic Morphisms

We can specify what it means for a morphism to be deterministic simply by describing how it interacts with other morphisms.

Formally, Fritz (2020) defines a morphism $f : X \to Y$ in a Markov category to be deterministic when the following equality holds:

$$(f \otimes f) \circ \mathsf{cp}_X = \mathsf{cp}_Y \circ f$$

Fritz (2020) draw this as follows:



An intuitive way to see how f not respecting the cp structure gives rise to randomness is to think of f like rolling the dice. The process of rolling the dice and then copying the number we see on the top is not the same process as rolling two dice.

Now note that any deterministic morphism f must be a comonoid homomorphism, since del is already natural in a Markov category. A Markov category in which every morphism is deterministic is therefore a Cartesian category. However, many important Markov categories are not Cartesian, and this lack of determinism gives Markov categories their rich structure.

3.2.2.3 Conditional Probabilities

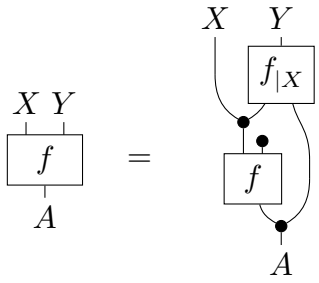
Given a joint distribution $p(x, y)$, we are often interested in computing the probability of y, *given that* x *occurred*. In other words, when conditioning y on x we are are computing the channel $p(y \mid x)$ such that it agrees with the information in $p(x, y)$. This is usually written as $p(x, y) = p(y \mid x)p(x)$.

Conditionals can be formulated in a Markov category, although not every Markov category has conditionals. Cho and Jacobs (2019) refer to CD-categories with conditionals as admitting disintegration.

Formally, given $f : A \to X \otimes Y$ in the Markov category $\mathbf{C}$, Fritz (2020) defines the morphism $f_{|X} : X \otimes A \to Y$ in $\mathbf{C}$ to be a conditional of f with respect to X if the following equation holds:

$$f = (id_X \otimes f_{|X}) \circ (((\mathsf{cp}_X \otimes \mathsf{del}_Y) \circ f) \otimes id_A) \circ \mathsf{cp}_A$$

Fritz (2020) draw this as:



and says that the Markov category $\mathbf{C}$ has conditionals if for all objects $A, X, Y \in \mathbf{C}$ and morphisms $f : A \to X \otimes Y$ in $\mathbf{C}$ there exists a conditional $f_{|X}$ of f with respect to X.

3.2.2.4 Independence

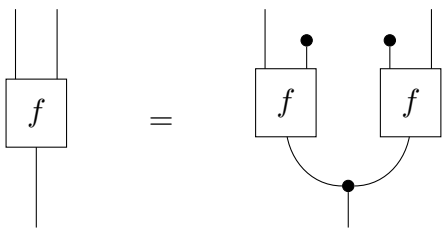
One of the most important concepts in probability theory is independence. Independence enables us to reason about the relationships between distributions or processes, and basically every nontrivial probabilistic computation requires the application of independence assumptions. Although independence is typically defined in terms of random variables or measurable functions out of a probability space, several authors have begun to explore categorical characterizations of independence.

For example Fritz (2020) introduce two equivalent characterizations of conditional independence in a Markov category $\mathbf{C}$:

- The morphism $f : A \to X \otimes Y$ satisfies:

  $$f = (id_X \otimes \mathsf{del}_Y \otimes \mathsf{del}_X \otimes id_Y) \circ (f \otimes f) \circ \mathsf{cp}_A$$

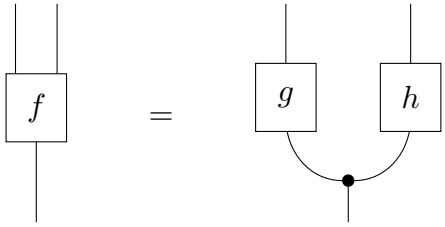
  which Fritz (2020) draws as:



- There are $g : A \to X$ and $h : A \to Y$ such that:

  $$f = (g \otimes h) \circ \mathsf{cp}_A$$

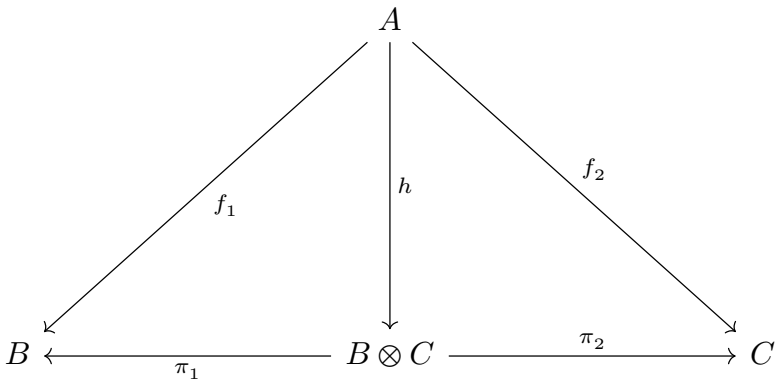
  which Fritz (2020) draws as:



That is, $f : A \to X \otimes Y$ displays conditional independence if we get the same result when we:

- Generate $X \otimes Y$ from A
- Independently generate X and Y from A and then tensor them together

Franz (2002) proposes a similar definition of independence for any semicartesian monoidal category $\mathbf{C}$. Consider the objects A, B, C and morphisms $f_1 : A \to B, f_2 : A \to C$ in $\mathbf{C}$. The morphisms f_1, f_2 are independent if there exists some morphism $h : A \to B \otimes C$ such that the following diagram commutes. π_1, π_2 are the projections of the tensor product.



Simpson (2018) explores independence from a slightly different perspective. He defines an independence structure on a category $\mathbf{C}$ to be a collection of multispans, or tuples $(A, \{f_i : A \to B_i\})$, that form a multicategory (contain identities and closed under composition), satisfy a number of coherence properties, and contain every singleton family $(A, \{f : A \to B\})$.

3.2.2.5 Cartesian Closedness and Quasi-Borel Spaces

It is often desirable to reason about spaces of probabilistic mappings. For example, defining distributions over these spaces is critical for writing and reasoning about probabilistic programs. However, while **Meas** is symmetric monoidal, it is not Cartesian closed, since the space of measurable maps into spaces of measurable maps is not always measurable. Culbertson and Sturtz (2013) attempt to address this problem by equipping **Meas** with a different monoidal tensor to form a symmetric monoidal closed category.

Heunen et al. (2017) use a different strategy and instead introduce a generalization of measurable spaces, which they call quasi-Borel spaces. There are a number of ways to characterize quasi-Borel spaces, including as structured measurable spaces or as a conservative extension of standard Borel spaces that supports simple type theory (products, coproducts and function spaces).

Heunen et al. (2017) show that the category of quasi-Borel spaces is Cartesian closed. They use this insight to describe a compositional procedure for Bayesian linear regression. They also demonstrate that de Finetti's theorem can be generalized to quasi-Borel spaces.

3.2.3 Causality and Bayesian Updates

There are a wide variety of category theoretic approaches to simple Bayesian modeling. In this section we explore a collection of abstract and concrete approaches.

3.2.3.1 Bayes Law; Abstractly

Fong (2013) uses a graphical framework to generalize Bayesian networks. He introduces the notion of a causal theory, or a strict symmetric monoidal category generated by a directed acyclic graph and equipped with a comonoidal structure on each object.

Fong (2013) demonstrates that for any causal theory $\mathbf{C}$ generated by a directed acyclic graph with vertex set V a Bayesian network can be derived as a functor $F : \mathbf{C} \to \mathbf{Stoch}$. Such a network maps $f : A \to B$ in the causal theory $\mathbf{C}$ to a Markov Kernel $F(f)$ that encodes the probabilistic relationship between A and B. Forward (predictive) inference is then mediated by the compositions of morphisms in the DAG and their images under F. Fong (2013) does not describe how causal theories could be used for backwards inference or how they can be learned from data.

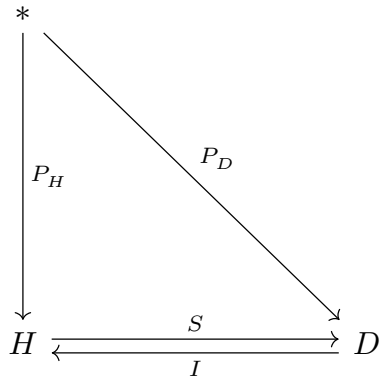
It is easy to see how this construction could be generalized by replacing **Stoch** with any other Markov category. Jacobs (2018) builds on this perspective to describe a generalized framework for Bayesian updates. He shows that by replacing **Stoch** with different categories we can represent both observations and models as functors out of a structure similar to the causal theories of Fong (2013). He then shows that the process of updating the parameters of a model based on observations can be represented as a natural transformation from the functor representing an observation to the functor representing a model. Jacobs (2018) characterizes both frequentist and Bayesian learning procedures in this way.

These works also hint at how categorical probability can serve as a basis for a structural perspective on machine learning. Cho and Jacobs (2019) use their generalized string diagram formulation of Bayesian inversion to step through a small example of naive Bayes classification: they first disintegrate channels from a table of data, and then invert the combined channels to yield a classification. It's worth noting how different this formulation is from the lens-based perspective on learning that we explored in Section 3.1. Whereas the lens papers abstract away the objective function and focus on the mechanics of model updates, Cho and Jacobs' objective is inextricably tied to the marginalization, inversions, and disintegrations in the learning procedure.

3.2.3.2 Bayes Law; Concretely

Culbertson and Sturtz (2013, 2014) take a more concrete perspective on Bayes law and

exclusively work in a subcategory of **Stoch** that has conditionals. They define a Bayesian model to be a diagram that consists of the following components:



- Hypothesis space: A measurable space H that serves as the space of potential models.
- Data space: A measurable space D that serves as the space of possible experiment outcomes.
- Prior probability measure: An arrow $P_H : * \to H$.
- Sampling distribution: An arrow $S : H \to D$ that relates the models in H to distributions over the data space D.
- Inference Map: An arrow $I : D \to H$ that relates new data observations to posterior probabilities over H.

The authors construct I such that for $\sigma_D \in D, \sigma_H \in H$:

$$\int_{x_D \in \sigma_D} I(x_D, \sigma_H) dP_D = \int_{x_H \in \sigma_H} S(x_H, \sigma_D) dP_H$$

The authors represent a new data observation with a probability measure μ over D, or an arrow $\mu : * \to D$. Given such an observation the inference map I defines the posterior probability over H to be:

$$\hat{P_H}(\sigma_H) = I \circ \mu = \int_{x_D \in D} I(x_D, \sigma_H) d\mu$$

The authors' construction then iterates the following process for each new data point $\mu : * \to D$:

1. Set $\hat{P_H}(\sigma_H) = I \circ \mu$.
2. Set P_H to $\hat{P_H}$.
3. Define I from P_H and S.

In a machine learning application we would expect that the elements of the hypothesis space H are maps over the data space D. The cleanest way to represent such maps would be as exponential objects. However, **Meas** is not closed over the Cartesian product. In order to solve this, Culbertson and Sturtz (2013) define a new monoidal tensor such that **Meas** becomes closed. This allows the authors to define Bayesian models over function spaces, including Gaussian processes and Kalman filters.

The solution by Culbertson and Sturtz (2013) to redefine the monoidal tensor contrasts with the solution by Heunen et al. (2017), who instead replace **Meas** with a Cartesian closed category.

3.2.4 Lenses for Probability

Some authors have also begun to study lenses in probability. For example, Smithe (2020) leverages a synthetic approach to Bayesian inversion to demonstrate that Bayesian updates compose with a lens pattern. Since his construction relies on the Cartesian-closedness of the base category, Smithe's primary example base category is the category of quasi-Borel spaces by (Heunen et al., 2017) rather than the category **Meas** of measurable spaces.

Smithe (2020) builds on this category to define a category of lenses in which the get map is a morphism in a CD-category (Cho and Jacobs, 2019) and the put map is its inverse. Smithe (2020) uses this category to demonstrate that Bayesian inversion is almost-everywhere preserved upon lens composition. One benefit of this perspective is that Smithe (2020) can study the lawfulness of Bayesian lenses, similarly to the exploration of the lawfulness of learners as lenses by Fong and Johnson (2019). Due to the inherent uncertainty in the processes that Bayesian lenses model, it is not surprising that none of the three laws (GetPut, PutGet, PutPut) hold in general.

3.3 Invariant and Equivariant Learning

The research in this section focuses on the symmetry preserving properties of machine learning algorithms. The authors whom we cite use a variety of tools for exploring the relationships between transformations of datasets and the outputs of machine learning models that run on those datasets. Many of these tools are explicitly category theoretic, such as functors and natural transformations.

The authors in this section focus on a variety of different kinds of machine learning algorithms that are used widely in practice:

- **Clustering** (Section 3.3.1.1): Clustering algorithms group points in the same dataset together. Social networks use clustering algorithms to identify users who share interests or social connections (Satuluri et al., 2020).
- **Manifold Learning** (Section 3.3.1.7): Manifold learning algorithms map the points in a dataset to low-dimensional embeddings in $\mathbb{R}^n$, which can then be used as features for machine learning models. Manifold learning algorithms are also commonly used along with nearest neighbor search algorithms to power recommendation engines (Virani et al., 2020).
- **Convolutional Neural Networks** (Section 3.3.4): Convolutional neural networks process image data in a way that exploits the position invariance inherent to most

Algorithm	Can produce overlapping clusters?	Is functor from **Met**?	Is functor from $\mathbf{Met}_{inj}$?
Single Linkage	No	Yes	Yes
Robust Single Linkage	No	No	Yes
Maximal Linkage	Yes	Yes	Yes
K-Means	No	No	No

Table 1: *Clustering algorithms*

image processing tasks. These are used in most modern image processing applications (Linsley et al., 2018).

- **Graph Neural Networks** (Section 3.3.4): Graph neural networks process graph data in a way that exploits both node features and internode connectivities. Graph neural networks are used widely for network understanding (Wu et al., 2021).

3.3.1 Functorial Unsupervised Learning

An unsupervised learning algorithm is any algorithm that aims to extract insights from data without explicit supervision. The class of unsupervised algorithms is much more general than that of supervised algorithms, but most unsupervised algorithms operate by doing one or both of the following:

- Estimating a low dimensional manifold that the data is assumed to lie upon.
- Determining the shape of the probability distribution that the data was drawn from

In order for an unsupervised algorithm to be useful, the properties of the lower dimensional manifold or probability distribution that the algorithm uses to describe the observed data must be somewhat in line with the structure of that data. One way to formalize this is to cast these algorithms as various kinds of functors and characterize this property in term of functoriality.

3.3.1.1 Nonoverlapping Flat Clustering

One of the most common classes of unsupervised learning algorithms is clustering algorithms. Suppose we have a finite dataset X that has been sampled from some larger space $\mathbf{X}$ according to a probability measure $\mu_{\mathbf{X}}$ over $\mathbf{X}$. Clustering algorithms group points in X together. These groups may be regions of high density in $\mu_{\mathbf{X}}$, path components in the support of $\mu_{\mathbf{X}}$, or simply collections of points in X that share some common trait (e.g. type of flower, class of image). From another perspective, a clustering of a metric space (X, d_X) is essentially a partition of X such that the points x, x' are more likely to be in the same cluster if $d_X(x, x')$ is small.

By casting clustering algorithms as functors we can find commonalities between different algorithms, derive extensions of algorithms that preserve functoriality and identify modifications that break it.

Carlsson and Mémoli (2008, 2013) describe clustering algorithms as functors from a category of metric spaces to a category of partitions. An example clustering functor is the following:

Definition 3.8. *The δ-single linkage functor maps a metric space (X, d_X) to the partition of X in which the points x_1, x_n are in the same partition if and only if there exists some sequence of points:*

$$x_1, x_2, \cdots, x_{n-1}, x_n$$

such that for all (x_i, x_{i+1}) in this sequence we have:

$$d_X(x_i, x_{i+1}) \leq \delta$$

The clusters defined by single linkage clustering are the connected components of the Vietoris-Rips complex (Definition 2.54), and can be viewed as a discrete approximation of the path components in a topological space.

Carlsson and Mémoli (2013) note that there are many uninteresting clustering functors, so they define an additional property to restrict to a more interesting class of maps:

Definition 3.9. *An excisive clustering functor is one that is idempotent in that applying the functor to any of the partitions it forms will not further subpartition that partition.*

A variety of clustering algorithms can be expressed as excisive clustering functors over $\mathbf{Met}_{isom}$. Theorem 6.1 by Carlsson and Mémoli (2013) shows that any excisive clustering algorithm that can be defined exclusively in terms of its actions on isometry classes is an excisive clustering functor over $\mathbf{Met}_{isom}$. This includes agglomerative clustering algorithms like average linkage.

In contrast, a much smaller set of algorithms can be expressed as excisive clustering functors over $\mathbf{Met}_{inj}$ (described in detail below), and single linkage clustering is the only excisive clustering functor over all of $\mathbf{Met}$.

However, there are a number of major issues with single linkage clustering that make it perform poorly in practice. For example, it is extremely sensitive to noise points, which can fall between distinct clusters and cause them to be erroneously bridged. For this reason, practical applications of single linkage clustering generally include pre/post-processing stages that rescale distances or remove points to reduce the impact of noise.

For example, Wishart (1969) proposes first defining a density estimate over the space and then removing points that appear in low-density regions, and Chaudhuri and Dasgupta (2010) instead describe the following approach:

Definition 3.10. *Given $\gamma \in (0, 1]$, $\delta \in \mathbb{R}_{\geq 0}$ the (γ, δ)-robust single linkage algorithm maps the metric space (X, d_X) to the partition of X in which the distinct points x_1, x_n are in the same partition if and only if there exists some chain of points:*

$$x_1, x_2, \cdots, x_{n-1}, x_n$$

such that for all (x_i, x_{i+1}) in this sequence we have:

$$d_X^{\gamma}(x_i, x_{i+1}) \leq \delta$$

where:

$$d_X^{\gamma}(x_i, x_{i+1}) = max(d_X(x_i, x_{i+1}), \mu_{X_\gamma}(x_i), \mu_{X_\gamma}(x_{i+1}))$$

and $\mu_{X_\gamma}(x_i)$ is the distance from x_i to its $\lfloor \gamma * |X| \rfloor$th nearest neighbor.

Intuitively, (γ, δ)-robust single linkage reduces the impact of dataset noise by increasing distances in sparse regions of the space. Note that d_X^γ is not a metric on X since $d_X^\gamma(x, x)$ is not guaranteed to be 0 for $x \in X$.

Carlsson and Mémoli (2013) also explore how we can define a broad family of clustering algorithms in $\mathbf{Met}_{inj}$ that factor through single linkage clustering. Since maps in $\mathbf{Met}_{inj}$ must be injective, clustering functors over $\mathbf{Met}_{inj}$ can be sensitive to the number of points in a region, such as the robust single linkage algorithm (Chaudhuri and Dasgupta, 2010). Such a density sensitive mapping would not be functorial over $\mathbf{Met}$ because a morphism in $\mathbf{Met}$ may collapse multiple points into the same point.

Carlsson and Mémoli (2013) use this insight to define an explicit generative modeling framework for excisive clustering functors out of $\mathbf{Met}_{inj}$. This framework can represent any such functor as a finite collection of finite metric spaces Ω. The represented functor then maps the points x_1, x_{k+1} in the metric space X to the same partition if and only if there exist:

- A sequence of $k+1$ points $x_1, x_2, \cdots, x_k, x_{k+1} \in X$
- A sequence of k metric spaces $\omega_1, \cdots, \omega_k \in \Omega$
- A sequence of k morphisms $f_1, \cdots, f_k$ in $\mathbf{Met}_{inj}$ where $f_i : \omega_i \to X$ such that there exist points $(\alpha_i, \beta_i) \in \omega_i$ where $f_i(\alpha_i) = x_i, f_i(\beta_i) = x_{i+1}$

The δ-single linkage functor (Definition 3.8) is then represented by the collection $\Omega = \{\Delta_2(\delta)\}$ where $\Delta_2(\delta)$ is the finite 2-point metric space whose points are separated by distance δ.

This functor is fundamental in the sense that any clustering functor F expressible by this framework factors through the δ-single linkage functor. That is, there exists some δ such that F is equivalent to the composition of the δ-single linkage functor with a functor $G : \mathbf{Met}_{inj} \to \mathbf{Met}_{inj}$ that commutes with the forgetful functor from metric spaces to $\mathbf{Set}$ (i.e. G maps the metric space (X, d_X) to (X, d'_X)). It is worth noting that "transform the distance metric and then apply single linkage clustering" is a powerful recipe for noise-resistant clustering algorithms, such as DBSCAN (Ester et al., 1996).

3.3.1.2 Nonoverlapping Hierarchical Clustering

A particularly important result in clustering theory is the Kleinberg Impossibility Theorem (Kleinberg, 2003), which states that it is impossible to define a clustering algorithm that satisfies all three of the following properties:

- Scale Invariance: The clustering function should not be sensitive to the units of measurement. Any algorithm that uses a distance hyperparameter (such as single linkage clustering) fails to satisfy scale invariance.
- Richness/Surjectivity: The clustering function should be able to produce any clustering, given an appropriate distance function. Intuitively, a clustering algorithm that satisfies this condition is morally similar to a classification algorithm that can shatter any set of points (i.e. infinite VC dimension (Vapnik and Chervonenkis, 2015)). Any algorithm that requires a pre-set number of clusters (such as K-means) fails to satisfy this condition.

- Consistency: Shrinking the distances between points in the same cluster and increasing the distances between points in different clusters does not change the clustering. Any centroid-based algorithm (such as mixture of Gaussians) fails to satisfy this condition.

One strategy for getting around this restriction is to define hierarchical clustering algorithms that generate a series of clusterings, each at a different scale (thereby relaxing scale invariance). This is similar to the strategy used in persistent homology (Chazal and Michel, 2017).

There are a number of formalizations of hierarchical clustering algorithms. For example, Carlsson and Mémoli (2008) introduce a category in which objects are persistent sets, or pairs (X, θ_X) of a finite set X and a function θ_X from the nonnegative real line to the set of partitions of X where:

1. If $r \leq s$ then $\theta_X(r)$ refines $\theta_X(s)$.
2. For any r, there is a number $\epsilon > 0$ so that $\theta_X(r') = \theta_X(r)$ for all $r' \in [r, r + \epsilon]$.

The morphisms in their category are persistence preserving maps, or functions $f : (X, \theta_X) \to (Y, \theta_Y)$ such that for any $r \in [0, \infty)$ the partition $\theta_X(r)$ of X is a refinement of the partition $f^{-1}(\theta_Y(r))$ of X. They define a hierarchical clustering algorithm to be a functor from a category of metric spaces into this category.

The authors' category is similar to the category **FPart** (Definition 2.72) under the transformation $-\log : (0, 1]^{\text{op}} \to [0, \infty)$. For example, we can define the hierarchical analog of the δ-single linkage functor (Definition 3.8) as a functor into **FPart** as follows:

Definition 3.11. *The single linkage functor* $\mathcal{SL} : \mathbf{UMet} \to \mathbf{FPart}$ *maps a metric space* (X, d_X) *to the functor:*

$$\mathcal{SL}(X, d_X) : (0, 1]^{op} \to \mathbf{Part}$$

where for some $a \in (0, 1]$*, the points* x_1, x_n *are in the same partition in* $\mathcal{SL}(X, d_X)(a)$ *if and only if there exists some chain of points:*

$$x_1, x_2, \cdots, x_{n-1}, x_n$$

such that for all (x_i, x_{i+1}) *in this sequence we have:*

$$d_X(x_i, x_{i+1}) \leq -\log(a)$$

3.3.1.3 Flat Overlapping Clustering

Culbertson et al. (2016) use a different approach from Carlsson and Mémoli (2013, 2008) and study clustering algorithms that can produce overlapping clusters. This produces another lever to mitigate the negative impact of chaining that occurs in single linkage clustering: since the relation that groups points into clusters does not need to be transitive, the algorithm can enforce maximum distance constraints on the points in the same clusters.

Their construction is based on the insight that the clusters in a nonoverlapping clustering are just the fibers $\{y \mid x \, R \, y\}$ of a symmetric, reflexive, transitive relation R (a partition or equivalence relation). Relaxing the nonoverlapping condition requires simply

weakening the transitivity property of this relation. This admits a more general class of covers than just partitions.

In particular, their construction replaces partitions with nonnested flag covers (Definition 2.62). Recall that a nonnested flag cover is a nonnested cover whose associated simplicial complex is a flag complex (Definition 2.53).

A particularly important class of flag covers is the class formed from the maximally linked subsets of a relation:

Definition 3.12. *Given a symmetric, reflexive relation R on the set X, the set of maximally linked subsets of X consists of all $S \subseteq X$ where (1) $x\,R\,y$ for all $x, y \in S$ and (2) S is not properly contained in any subset of X satisfying (1).*

Culbertson et al. (2016) define overlapping clustering functors to map from a category of metric spaces and nonexpansive mappings (such as **Met**, $\mathbf{Met}_{inj}$, or $\mathbf{Met}_{isom}$) to the category **Cov** of nonnested flag covers and consistent maps between them (Definition 2.65). They then define a generative model for representing nonoverlapping clustering algorithms in terms of finite sets of finite metric spaces.

Formally, given a set of finite metric spaces $\mathcal{T}$, Culbertson et al. (2016) construct the clustering functor:

$$\mathbf{ML}^{\mathcal{T}} : \mathbf{Met} \to \mathbf{Cov}$$

to map the metric space (X, d_X) to the flag cover formed from the maximally linked subsets of the relation R such that $x\,R\,x'$ if for some $(T, d_T) \in \mathcal{T}$ there exists a morphism (nonexpansive map) $t : (T, d_T) \to (X, d_X)$ such that both x and x' are in the image of t.

The main difference between this construction and the generative model by Carlsson and Mémoli (2013) is the lack of transitivity. A clustering functor characterized by Carlsson and Memoli's construction maps the points x, x' to the same partition if there exists a sequence of metric spaces in Ω whose images connect them. In contrast, a clustering functor generated by this construction will connect x and x' if there is a morphism from a single metric space in $\mathcal{T}$ that maps onto both of them.

To be specific, consider the clustering functor characterized by the collection $\mathcal{T} = \{\Delta_2(\delta)\}$ where $\Delta_2(\delta)$ is the finite 2-point metric space where the points are separated by distance δ. According to the generative model by Carlsson and Mémoli (2013), this functor is the δ-single linkage functor. However, according to this generative model, this is instead the δ-maximal linkage functor, defined as follows:

Definition 3.13. *The δ-maximal linkage functor maps a metric space (X, d_X) to the set of maximally linked subsets of the relation R where $x_1\,R\,x_2$ when $d_X(x_1, x_2) \leq \delta$.*

In this generative model the δ-single linkage functor is instead formed from the collection:

$$\mathcal{T} = \{\Delta_0(\delta), \Delta_1(\delta), \Delta_2(\delta)), \Delta_3(\delta), \cdots\}$$

where $\Delta_n(\delta)$ is the finite n-point metric space where each pair of points is separated by distance δ.

3.3.1.4 Hierarchical Overlapping Clustering

In a follow-up paper, Culbertson et al. (2018) extend their overlapping clustering construction to handle hierarchical overlapping clustering algorithms. They generalize persistent sets (Carlsson and Mémoli, 2008) to persistent covers by replacing the partitions in the codomain with nonnested flag covers. They primarily work with a restriction of persistent covers which they call sieves: a sieve is a persistent cover (X, θ_X) for which exists some $r \in [0, \infty)$ where $\theta_X(r) = \{X\}$.

Carlsson and Mémoli (2008) define the category **Sieve** to have sieves as objects and persistence preserving maps as morphisms, and they build their hierarchical clustering algorithms as functors into **Sieve**.

Formally, Carlsson and Mémoli (2008) define a stationary sieving functor to be a functor F from the category of finite weight spaces to **Sieve** where F commutes with the forgetful functor to **Set** and given the functor $\mathcal{J}$ that maps the sieve (X, θ_X) to (X, d_X) where:

$$d_X(x, x') = \min\{t \mid \exists S_X \in \theta_X(t), x \in S_X, x' \in S_X\}$$

the composition $\mathcal{J} \circ F$ is idempotent and nonexpansive.

Like Carlsson and Mémoli (2013), Culbertson et al. (2018) demonstrate that there exists a universal sieving functor through which every stationary sieving functor factors. However, unlike the construction by Carlsson and Mémoli (2013), this functor is the hierarchical maximal linkage functor (which they also call the Rips Sieving functor) rather than the single linkage functor. This highlights a fundamental difference between overlapping and nonoverlapping clustering functors: maximal linkage is universal for overlapping clustering, whereas single linkage is universal for nonoverlapping clustering.

The category **Sieve** is similar to the category **FCov** (Definition 2.71) under the transformation:

$$-\log : (0, 1]^{\text{op}} \to [0, \infty)$$

For example, we can define the hierarchical analog of the δ-maximal linkage functor (Definition 3.13) as a functor into **FCov** as follows:

Definition 3.14. *The maximal linkage functor $\mathcal{ML} : \mathbf{UMet} \to \mathbf{FCov}$ maps a metric space (X, d_X) to the functor:*

$$\mathcal{ML}(X, d_X) : (0, 1]^{op} \to \mathbf{Cov}$$

where for some $a \in (0, 1]$ the points x_1, x_2 are in the same partition in $\mathcal{ML}(X, d_X)(a)$ if and only if:

$$d_X(x_1, x_2) \leq -\log(a)$$

3.3.1.5 Flattening and Stability

In practice, it can be difficult to analyze a hierarchical clustering of a dataset. As a result, one of the most common operations to perform on a hierarchical clustering is to flatten it, or convert it to a flat clustering.

A core insight of topological data analysis is that structures that exist at multiple scales are particularly important descriptors of a dataset (Chazal and Michel, 2017). For

example, the important connected components of a filtration are those which have large differences between the indices of the simplicial complexes in which they first and last appear. McInnes and Healy (2017) and Chazal et al. (2013) use this insight to define tree-based strategies to flatten a hierarchical clustering.

Rolle and Scoccola (2020) use interleaving distance (Chazal et al., 2009, 2014) to explore the stability of these flattening algorithms.

Definition 3.15. *Suppose $\mathbb{R}_{\geq 0}$ is the total order on the nonnegative reals. For some category $\mathbf{C}$ an ϵ-interleaving between the functors $F : \mathbb{R}_{\geq 0} \to \mathbf{C}$ and $G : \mathbb{R}_{\geq 0} \to \mathbf{C}$ is a collection of commuting natural transformations:*

$$\alpha : F \to G^\epsilon$$
$$\beta : G \to F^\epsilon$$

where:

$$G^\epsilon(r) = G(r + \epsilon)$$
$$F^\epsilon(r) = F(r + \epsilon)$$

and we have:

$$\beta(r + \epsilon) \circ \alpha(r) = F(r \leq r + 2\epsilon)$$
$$\alpha(r + \epsilon) \circ \beta(r) = G(r \leq r + 2\epsilon)$$

Definition 3.16. *For some category $\mathbf{C}$ the interleaving distance between the functors $F : \mathbb{R}_{\geq 0} \to \mathbf{C}$ and $G : \mathbb{R}_{\geq 0} \to \mathbf{C}$ is the infimum of the set of all $\epsilon \geq 0$ such that an ϵ-interleaving exists between F and G.*

Scoccola (2020) explore how we can prove that an algorithm is stable by casting it as a series of transformations between functors out of $(\mathbb{R}, \leq)$ and proving that each transformation is uniformly continuous with respect to the interleaving distance.

3.3.1.6 Multiparameter Hierarchical Clustering

Some clustering algorithms accept multiple hyperparameters. Carlsson and Mémoli (2010) explore how we can represent these algorithms as functors in a way that captures the relationship between the values of hyperparameters and the algorithm output.

The authors define a persistent structure Q_X on the set X to be a map that sends pairs of points $x, x' \in X$ to the regions of the hyperparameter space in which x, x' are in the same cluster. They define a category in which objects are pairs of sets and persistent structures (X, Q_X) and morphisms:

$$f : (X, Q_X) \to (Y, Q_Y)$$

are functions $f : X \to Y$ such that for all $x, x' \in X$ we have:

$$Q_X(x, x') \subseteq Q_Y(f(x), f(x'))$$

Carlsson and Mémoli (2010) then define multiparameter hierarchical clustering functors as functors from **Met** into this category. They use this construction to extend the uniqueness theorem of single linkage clustering (Carlsson and Mémoli, 2008) to 2-dimensional hierarchical clustering functors.

McInnes and Healy (2017) explore multiparameter clustering algorithms that compute persistence across both scale and density parameters. They use HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) as an example. Their framework is very similar to the multiparameter clustering system that Carlsson and Mémoli (2010) describe.

One challenge with using multiparameter hierarchical clustering algorithms is that the most common procedures for flattening hierarchical clusterings (as in Section 3.3.1.5) rely on the possibility of representing the clustering with a merge tree or dendrogram (Chazal et al., 2013; McInnes and Healy, 2017). When the hierarchical structure is indexed by a partial order like $(\mathbb{R}^n, \leq)$ rather than a total order like $(\mathbb{R}, \leq)$, this is no longer an option. Given a hierarchical clustering functor H and the vectors:

$$(a[1], a[2], \cdots, a[n]) \in \mathbb{R}^n$$
$$(a'[1], a'[2], \cdots, a'[n]) \in \mathbb{R}^n$$

it is possible that there are two clusters:

$$c \in H(a[1], a[2], \cdots, a[n])$$
$$c' \in H(a'[1], a'[2], \cdots, a'[n])$$

such that c and c' are neither disjoint nor in a containment relationship. This is closely related to the phenomenon that unlike single parameter persistence modules, multiparameter persistence modules cannot be decomposed into a direct sum of simple modules (Lesnick and Wright, 2015).

In order to get around this limitation, some authors parameterize persistent structures along affine slices of $(\mathbb{R}^n, \leq)$. For example, Rolle and Scoccola (2020) define a hierarchical clustering algorithm that is indexed by a curve γ in $(\mathbb{R}^n, \leq)$, rather than all of $(\mathbb{R}^n, \leq)$. In this way the clustering represents structure across different values of each parameter.

3.3.1.7 Manifold Learning

Some authors have also begun exploring functorial frameworks for manifold learning. McInnes et al. (2018) build a dimensionality reduction algorithm, UMAP (Uniform Manifold Approximation and Projection), that uses simplicial complexes and filtrations to define a coherent way to combine multiple local approximations of geodesic distance.

UMAP exploits the property that we can locally approximate the geodesic distance between points that lie on a manifold that is embedded within $\mathbb{R}^n$. In order to extend these local approximations into an approximation of the geodesic distance between any point x and its neighbors, UMAP defines a custom distance metric for x such that the data is uniformly distributed with respect to this metric. This metric defines the distance from x to any point y to be $\frac{1}{r} d_{\mathbb{R}^n}(x, y)$, where $d_{\mathbb{R}^n}(x, y)$ is the $\mathbb{R}^n$ distance from x to y and r is the distance from x to its nearest neighbor. This technique is similar to the strategies used in HDBSCAN (Campello et al.) and robust single linkage (Chaudhuri and Dasgupta, 2010), but in the opposite direction. Rather than expand distances in regions of low density, UMAP contracts them. This effectively normalizes all regions to uniform density.

This rescaling creates a family of distinct metric spaces (one for each data point x). In order to combine these spaces into a coherent structure, UMAP builds a Vietoris-Rips complex for each metric space and then combines the complexes with a fuzzy set union.

The resulting combined simplicial complex acts as a representation of the local and global connectivity of the dataset. This representation is used to learn an $n \times d$ matrix A of d dimensional embeddings for the n elements of (X, d_X) by applying stochastic gradient descent to minimize the cross entropy between the reference simplicial complex and the Vietoris-Rips complex of the metric space defined by the embeddings in A.

3.3.2 Functorial Supervised Learning and Neural Networks

Some authors have begun to use similar techniques to those described in Section 3.3.1 to characterize the invariances of supervised learning algorithms and neural networks.

Harris (2019) builds a framework in which learning algorithms are natural transformations from a training dataset functor D to a prediction model functor P. Given a category $\mathbf{X}$ of input spaces, a category $\mathbf{Y}$ of output spaces, and an index category $\mathbf{I}$:

- $D : \mathbf{X} \times \mathbf{Y} \times \mathbf{I}^{\text{op}} \to \mathbf{Set}$ maps an (input, output, index) tuple to the set of all possible training datasets $\{(x_i, y_i) \mid i \in I\}$.
- $P : \mathbf{X}^{op} \times \mathbf{Y} \times \mathbf{I}^{\text{op}} \to \mathbf{Set}$ maps an (input, output, index) tuple to the set of all possible prediction functions $f : \mathbf{X} \to \mathbf{Y}$.

Harris (2019) defines learning algorithms as transformations $\mu : D \to P$. Intuitively such an algorithm defines a collection of functions, each of which maps a set of training datasets to a set of prediction functions.

Harris (2019) then builds on this definition to characterize learning algorithms in terms of the categories $\mathbf{X}, \mathbf{Y}, \mathbf{I}$ over which they satisfy a naturality condition. For example, the linear regression algorithm is natural when $\mathbf{X}$ is the category of finite dimensional vector spaces and invertible linear maps, $\mathbf{Y}$ is the category of finite dimensional vector spaces and linear maps, and $\mathbf{I}$ is the category of Euclidean spaces and monomorphisms. However, linear regression is not natural when we upgrade $\mathbf{X}$ to the full category of finite dimensional vector spaces and all linear maps between them. This is because the normal equations require inverting the data matrix.

As another example, Healy (2000) models the relationship between "concepts", which he represents as theories in formal logic, and components of a cognitive neural network that learns these concepts. He starts by defining a category that has concepts as objects and subconcept relationships as morphisms and demonstrating how colimits in this category can formalize the construction of complex concepts from simpler ones.

Healy (2000) also defines a category with neural network architectures as objects and sets of priming states, which represent how one part of the network can activate another, as morphisms. He builds on this construction to define functors from the category of concepts to the category of architectures that model how cognitive neural networks pass information about concepts.

Gavranović (2019) takes a slightly different perspective and focuses directly on the relationship between a neural network architecture and the task we are using it to solve. Typically this task is defined in terms of an objective function over data samples that are assumed to be drawn from some probability distribution.

One of the challenges that Gavranović (2019) discusses is the difficulty of defining a functor from neural network architectures to tasks. If such a functor F exists, then it must be the case that for architectures a_1, a_2, the functor F maps $a_2 \circ a_1$ to $Fa_2 \circ Fa_1$. This is very limiting: consider training the networks $a_2 \circ a_1$ and $a_3 \circ a_1$ on the same task.

There is no reason to believe that the best weights for a_1 within $a_2 \circ a_1$ are the same as the best weights for a_1 within $a_3 \circ a_1$.

This problem sheds light on one of the main challenges with using categorical models of optimization and neural networks to reason about machine learning: relating an optimization system to the task it is trained with. There are machine learning tasks for which the best weights for a_1 within $a_2 \circ a_1$ are the same as the best weights for a_1 within $a_3 \circ a_1$, but specifying these tasks to be compatible with our specifications for machine learning systems is challenging.

3.3.3 Functorial Statistics

Some authors have also used functoriality to explore how the different pieces of statistical learning systems can fit together. McCullagh (2002) attempts to unearth the invariants at the heart of statistical modeling. He does so by forming categories over which we can define statistical designs and model parameters as functors such that statistical models form natural transformations between them. The commutativity of these natural transformations enforces that parameter maps maintain their structure independent of sample space transformations and that the meaning of a parameter does not change in response to modifications of the statistical design.

Furthermore, Allison (2003) defines a system for representing components and categories of statistical models as types and classes. He uses this system to identify and factor out the common components between different types of algorithms, such as mixture modeling and decision trees. He also describes how recombinations of these common components can lead to new algorithms.

3.3.4 Equivariant Neural Networks

A particularly important stream of machine learning research focuses on the equivariance and invariance properties of neural network architectures. This work has heavy category theoretic undertones, which has been bubbling to the surface in recent years.

A function is equivariant under a transformation if applying that transformation to its inputs results in an equivalent transformation of its outputs. Invariance is a special case of equivariance in which any transformation of the inputs results in an identity transformation of the outputs. For example, the weight-sharing regimes of CNNs and RNNs make them equivariant under image translations and sequence position shifts respectively (up to edge effects).

For this reason, researchers are actively exploring neural network architectures with equivariance properies. Cohen and Welling (2016) generalize CNNs to G-CNNs, which use generalized convolution and pooling layers to exhibit equivariance to group transformations like rotations and reflections as well as translations. Intuitively, G-convolutions replace the position shifting in a discrete convolution operation with a general group of transformations. As group equivariance has grown in popularity, some authors have begun to dig deeper into its theoretical foundations. Kondor and Trivedi (2018) show that a neural network layer is G-equivariant only if it has convolution structure and Cohen et al. (2020) show that linear equivariant maps between feature spaces are in one-to-one correspondence with convolutions using equivariant kernels.

Cohen et al. (2018) build on this work to develop a spherical CNN that uses spherical convolutions powered by generalized Fourier transformations to exhibit equivariance to

sphere rotations. The authors demonstrate that their network is particularly effective at classifying 3D shapes that have been projected onto a sphere with ray casting.

Cohen et al. (2019) extend neural network equivariance beyond group symmetries. They develop a strategy to make neural networks that consume data on a general manifold dependent only on the intrinsic geometry of the manifold. In particular, the authors develop a convolution operator that is equivariant under the choice of gauge, or the tangent frame on the manifold relative to which the orientation of filters are defined. The authors implement gauge equivariant CNNs for a convenient type of manifold (the icosahedon) and demonstrate that this strategy outperforms standard approaches for learning spherical signals.

One data type with particularly complex symmetry properties is graph data. Maron et al. (2019) develop linear operators that are equivariant under graph permutations and de Haan et al. (2020) make explicit use of category theory to build more powerful neural networks that can exploit graph symmetries. They start by introducing the concept of a graph representation, which is essentially a functor from the category of graphs and graph isomorphisms to vector spaces and linear maps. They then generalize equivariant graph networks to natural graph networks. Each layer in a natural graph network is a natural transformation of graph representations. That is, it is a linear map that commutes with graph isomorphisms. This construction is both flexible and powerful: a layer in a natural graph network can apply different transformations to different graphs as long as the transformations commute with the chosen graph representation.

Mei et al. (2021) explore the precise statistical benefit of exploiting invariances in network architectures. They focus on functions that are invariant under actions by subgroups of the orthogonal group in d dimensions (elements are orthogonal matrices and composition is matrix multiplication). They show that increasing the size of the group under which a function is invariant has similar effects to increasing the training set size. They also demonstrate that under certain conditions incorporating group invariances to the model structure or augmenting the training dataset with group transformations are mathematically equivalent.

A related stream of work by Bronstein et al. (2016) aims to generalize equivariant and invariant neural network operators to non-Euclidean domains (e.g. graphs and manifolds). This geometric deep learning perspective largely focuses on the characterization and generalization of the equivariance properties of convolution, and therefore naturally benefits from the universality of the convolution as an invariant operator (Kondor and Trivedi, 2018; Cohen et al., 2020). Bronstein et al. (2016) illustrate that convolution commutes with the Laplacian operator, and they use this insight to define both spatially and spectrally motivated generalizations of convolution for non-Euclidean domains.

4 Optimization

In this Chapter we apply category theory to explore different aspects of the optimization process. In Section 4.1 we investigate how various optimization algorithms behave when we replace the gradient with the Cartesian reverse derivative. We prove that the key properties of these algorithms hold under very relaxed assumptions, and demonstrate this result through numerical experiments.

The constructions we explore in Section 4.1 optimize a parametric function by iterating dynamical systems whose states are function parameters. In Section 4.2 we build on this perspective through a category theoretic framework in which dynamical systems are a special case of lenses. We use this framework to express the dynamical systems defined by optimization algorithms as lenses. This allows us to compose simple dynamical systems to construct more complex optimization algorithms.

4.1 Generalized Optimization

In this section we use the Cartesian reverse derivative (Definition 3.4), a categorical generalization of reverse-mode automatic differentiation, to generalize several optimization algorithms. We study straightforward generalizations of gradient descent (Definition 2.12) and momentum (Definition 2.14) as well as a novel generalization of Newton's method (Definition 2.13). We then explore which properties of these algorithms are preserved under these generalizations.

First, we show that the transformation equivariances of these algorithms are preserved in this generalized setting: generalized Newton's method is equivariant under all invertible linear transformations, and generalized gradient descent and generalized momentum are equivariant under orthogonal linear transformations. Next, we show that we can express the change in loss of generalized gradient descent with an expression that is similar to an inner product, thereby generalizing the non-increasing and convergence properties of the gradient descent optimization flow. Finally, we include several numerical experiments to illustrate these ideas and demonstrate how we can use them to optimize polynomial functions over an ordered ring.

4.1.1 Background

Given a convex differentiable function $l : \mathbb{R}^n \to \mathbb{R}$, there are many algorithms that we can use to minimize it. For example, if we pick a step size α and a starting point $x_0 \in \mathbb{R}^n$ we can apply the gradient descent algorithm (Definition 2.12) in which we repeatedly iterate:

$$x_{t+1} = x_t - \alpha * \nabla l(x_t)$$

For small enough α this strategy is guaranteed to get close to the x that minimizes l (Boyd and Vandenberghe, 2004).

While gradient descent is particularly popular, it is not the only gradient based optimization algorithm that is widely used in practice. Both the momentum (Definition 2.14) and Adagrad (Definition 2.15) algorithms use placeholder variables that store information from previous gradient updates to improve robustness. Newton's method (Definition 2.13), which rescales the gradient with the inverse Hessian matrix, is popular for many applications but less commonly used in deep learning due to the difficulty of efficiently

implementing it with reverse-mode automatic differentiation. Each of these algorithms have different equivariance properties that affect their stability under dataset transformations: for example, Newton's method enjoys an equivariance to linear rescaling that gradient descent lacks.

Given the utility of these algorithms it is natural to explore when they can be generalized beyond differentiable functions. For example, if we replace the gradient in gradient descent with a representative of the subgradient, a simple generalization of the gradient for non-differentiable functions $l : \mathbb{R}^n \to \mathbb{R}$, the convergence properties of gradient descent are preserved (Boyd et al., 2003).

In this section we explore whether the convergence and equivariance properties of optimization algorithms built on top of the gradient and Hessian generalize to optimization algorithms built on top of the Cartesian reverse derivative (Cockett et al., 2019). In particular:

- We investigate the equivariance properties of several optimization algorithms, including gradient descent, momentum, and Newton's method.
- We use the Cartesian reverse derivative (Cockett et al., 2019) to define generalized analogs of several optimization algorithms, including a novel generalization of Newton's method.
- We derive novel results on the transformation equivariances of these generalized algorithms.
- We define the notion of an optimization domain over which we can apply these generalized algorithms and study their convergence properties.
- We show that the optimization domain of polynomials over ordered rings supports generalized gradient based optimization.
- We include numerical experiments to illustrate the above ideas and demonstrate how we can use them to optimize polynomial functions over an ordered ring. Code to run these experiments is on GitHub (Shiebler, 2021a).

4.1.2 Standard Optimization

As we described in Section 4.1.1, gradient descent optimizes an objective function $l : \mathbb{R}^n \to \mathbb{R}$ by starting at a point $x_0 \in \mathbb{R}^n$ and progressing the discrete dynamical system:

$$x_{t+1} = x_t - \alpha * \nabla l(x_t)$$

If we interpret the size of a time step as being α we can alternatively write:

$$x_{t+\alpha} = x_t - \alpha * \nabla l(x_t)$$

and taking the $lim_{\alpha \to 0}$ of this system yields the differential equation:

$$\frac{\partial x}{\partial t}(t) = -\nabla l(x(t))$$

which we can think of as the continuous limit of gradient descent.

4.1.2.1 Optimizers

Definition 4.1. *An optimizer for $l : \mathbb{R}^n \to \mathbb{R}$ with dimension k is a continuous function $d : \mathbb{R}^{kn} \to \mathbb{R}^{kn}$.*

Intuitively, an optimizer defines both a discrete and a continuous dynamical system

- Discrete System: $(x_{t+1}, y_{t+1}, \cdots) = (x_t, y_t, \cdots) + \alpha d(x_t, y_t, \cdots)$
- Continuous System: $\left(\frac{\partial x}{\partial t}(t), \frac{\partial y}{\partial t}(t), \cdots\right) = d(x(t), y(t), \cdots)$

The discrete dynamical system is the discretization of the continuous system by Euler's method.

The dimension of an optimizer determines its type signature. We can think of an optimizer with dimension $k > 1$ as using information beyond the previous value x_t to determine x_{t+1}.

In practice we usually work with optimizers that define dynamical systems in which $l(x(t))$ and $l(x_t)$ get closer to the minimum value of l as t increases. Given $l : \mathbb{R}^n \to \mathbb{R}$ we can construct the following examples:

Definition 4.2. *The gradient descent optimizer for $l : \mathbb{R}^n \to \mathbb{R}$ is:*

$$d : \mathbb{R}^n \to \mathbb{R}^n$$
$$d(x) = -\nabla l(x)$$

This optimizer has dimension 1, and is a representation of the gradient descent algorithm (Definition 2.12).

Definition 4.3. *Newton's optimizer for $l : \mathbb{R}^n \to \mathbb{R}$ is:*

$$d : \mathbb{R}^n \to \mathbb{R}^n$$
$$d(x) = -\nabla^2(l(x))^{-1}\nabla l(x)$$

This optimizer has dimension 1 and uses the inverse Hessian matrix to increase the stability of each update. This optimizer is a representation of Newton's method (Definition 2.13).

Definition 4.4. *The momentum optimizer for $l : \mathbb{R}^n \to \mathbb{R}$ is:*

$$d : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$$
$$d(x, y) = (y, -y - \nabla l(x))$$

This optimizer has dimension 2 and uses a placeholder variable to track the value of the previous update steps and simulate the momentum of a ball rolling down a hill. This optimizer is a representation of the momentum algorithm (Definition 2.14).

Definition 4.5. *The Adagrad optimizer for $l : \mathbb{R}^n \to \mathbb{R}$ is:*

$$d : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$$
$$d(x, y) = (-\nabla l(x)/\sqrt{y}, \nabla l(x)^2)$$

This optimizer has dimension 2 and uses a placeholder variable to reweight updates based on the magnitude of previous updates. This optimizer is a representation of the Adagrad algorithm (Definition 2.15).

4.1.2.2 Optimization Schemes

Definition 4.6. *An optimization scheme with dimension k:*

$$u : (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^{kn} \to \mathbb{R}^{kn})$$

is a family of functions (indexed by n) that maps objectives $l : \mathbb{R}^n \to \mathbb{R}$ to optimizers $d : \mathbb{R}^{kn} \to \mathbb{R}^{kn}$.

Each of the optimizers we introduce above can also be written more generally as optimization schemes. For example, the gradient descent optimization scheme is:

$$u : (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}^n)$$
$$u(l)(x) = -\nabla l(x)$$

and the momentum optimization scheme is:

$$u : (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^{2n} \to \mathbb{R}^{2n})$$
$$u(l)(x, y) = (y, -y - \nabla l(x))$$

In some situations we may be able to improve the convergence rate of the dynamical systems defined by optimization schemes by precomposing an invertible function $f : \mathbb{R}^m \to \mathbb{R}^n$. That is, rather than optimize the function $l : \mathbb{R}^n \to \mathbb{R}$ we optimize $l \circ f : \mathbb{R}^m \to \mathbb{R}$. However, for many optimization schemes there are classes of transformations under which they are equivariant: applying any such transformation to the data cannot change the trajectory.

Definition 4.7. *Suppose $f : \mathbb{R}^m \to \mathbb{R}^n$ is an invertible transformation and write f_k for the map:*

$$(f \times f \times \cdots) : \mathbb{R}^{km} \to \mathbb{R}^{kn}$$

The optimization scheme $u_n : (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^{kn} \to \mathbb{R}^{kn})$ is equivariant under f if for all $l : \mathbb{R}^n \to \mathbb{R}$ we have:

$$u_m(l \circ f) = f_k^{-1} \circ u_n(l) \circ f_k$$

The following proposition is a corollary of the affine invariance properties of the Newton step discussed in Chapter 9 of Boyd and Vandenberghe (2004).

Proposition 4.8. *Recall that an invertible linear transformation is a function $f(x) = Ax$ where the matrix A has an inverse A^{-1}. Recall also that an orthogonal linear transformation is an invertible linear transformation where $A^{-1} = A^T$. Newton's method is equivariant under all invertible linear transformations, whereas both gradient descent and momentum are equivariant under all orthogonal linear transformations.*

Proof. First, we will show that Newton's optimization scheme:

$$NEW(l)(x) = -(\nabla^2 l(x))^{-1} \nabla l(x)$$

is equivariant under invertible linear transformations. Consider any function $l : \mathbb{R}^m \to \mathbb{R}$ and function:

$$f : \mathbb{R}^n \to \mathbb{R}^m$$
$$f(x) = Ax$$

where A is invertible. We have:

$$
\begin{aligned}
& NEW(l \circ f) : \mathbb{R}^n \to \mathbb{R}^n \\
& NEW(l \circ f)(x) \\
= \quad & \{\text{Definition of } NEW\} \\
& -\left[(\nabla^2 (l \circ f)(x))^{-1}\right] \left[\nabla (l \circ f)(x)\right] \\
= \quad & \{\text{Chain rule with } f\} \\
& -\left[(A^T \nabla^2 l(Ax) A)^{-1}\right] \left[A^T \nabla l(Ax)\right] \\
= \quad & \{\text{Apply } (AB)^{-1} = B^{-1}A^{-1} \text{ twice}\} \\
& -\left[A^{-1} (\nabla^2 l(Ax))^{-1} A^{-T}\right] \left[A^T \nabla l(Ax)\right] \\
= \quad & \{A^{-T} \text{ and } A^T \text{ are inverses}\} \\
& -A^{-1} (\nabla^2 l(Ax))^{-1} \nabla l(Ax) \\
= \quad & \{\text{Definition of } f\} \\
& -f^{-1}((\nabla^2 l(f(x)))^{-1} \nabla l(f(x))) \\
= \quad & \{\text{Definition of } NEW\} \\
& f^{-1}(NEW(l)(f(x)))
\end{aligned}
$$

Next, we will show that the gradient descent optimization scheme:

$$GRAD(l)(x) = -\nabla l(x)$$

is equivariant under orthogonal linear transformations. Consider any function of the form $f(x) = Ax$ where A is an orthogonal matrix. Then the following holds when $A^T = A^{-1}$:

$$
\begin{aligned}
& GRAD(l \circ f) : \mathbb{R}^n \to \mathbb{R}^n \\
& GRAD(l \circ f)(x) \\
= \quad & \{\text{Definition of } GRAD\} \\
& -\nabla (l \circ f)(x) \\
= \quad & \{\text{Chain rule}\} \\
& -A^T (\nabla l(Ax)) \\
= \quad & \{\text{Apply } A^T = A^{-1}\} \\
& -A^{-1} (\nabla l(Ax)) \\
= \quad & \{\text{Definition of } GRAD\} \\
& f^{-1}(GRAD(l)(f(x)))
\end{aligned}
$$

Next, we will show that the momentum optimization scheme:

$$MOM(l)(x, y) = (y, -y - \nabla l(x))$$

is also equivariant under orthogonal linear transformations. Consider any function of the form $f(x) = Ax$ where A is an orthogonal matrix. Then the following holds when

$A^T = A^{-1}$:

$$
\begin{aligned}
& MOM(l \circ f) : \mathbb{R}^{2n} \to \mathbb{R}^{2n} \\
& MOM(l \circ f)(x, y)[0] \\
= \quad & \{\text{Definition of } MOM\} \\
& y \\
= \quad & \{A^{-1}A = I\} \\
& A^{-1}Ay \\
= \quad & \{\text{Definition of } MOM \text{ and definition of } f\} \\
& f^{-1}(MOM(l)(f(x), f(y))[0])
\end{aligned}
$$

and:

$$
\begin{aligned}
& MOM(l \circ f)(x, y)[1] \\
= \quad & \{\text{Definition of } MOM\} \\
& -y - \nabla(l \circ f)(x)) \\
= \quad & \{\text{Chain rule}\} \\
& -A^{-1}Ay - A^T \nabla l(Ax)) \\
= \quad & \{\text{Apply } A^T = A^{-1}\} \\
& A^{-1}\left(-Ay - \nabla l(Ax)\right) \\
= \quad & \{\text{Definition of } f\} \\
& f^{-1}\left(-f(y) - \nabla l(f(x))\right) \\
= \quad & \{\text{Definition of } MOM\} \\
& f^{-1}(MOM(l)(f(x), f(y))[1])
\end{aligned}
$$

$MOM(l \circ f)(x, y)[0]$ and $MOM(l \circ f)(x, y)[1]$ are respectively the first and second components of $MOM(l \circ f)(x, y)$. □

Adagrad is not equivariant under orthogonal linear transformations due to the fact that it tracks a nonlinear function of past gradients (sum of squares).

In order to interpret these equivariance properties it is helpful to consider how they affect the discrete dynamical system defined by an optimization scheme.

Proposition 4.9. *Given an objective function $l : \mathbb{R}^n \to \mathbb{R}$ and an optimization scheme:*

$$u : (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^{kn} \to \mathbb{R}^{kn})$$

that is equivariant under the invertible linear function $f : \mathbb{R}^m \to \mathbb{R}^n$, the discrete system defined by the optimizer $u(l \circ f)$:

$$y_{t+1} = y_t + \alpha u(l \circ f)(y_t)$$

cannot converge faster than the discrete system defined by the optimizer $u(l)$:

$$x_{t+1} = x_t + \alpha u(l)(x_t)$$

when $f_k(y_0) = x_0$.

Proof. Consider starting at some point $x_0 \in \mathbb{R}^{kn}$ and repeatedly taking steps:

$$x_{t+1} = x_t + \alpha u(l)(x_t)$$

Now suppose instead that we start at the point $y_0 = f_k^{-1} x_0$ and take steps:

$$y_{t+1} = y_t + \alpha u(l \circ f)(y_t)$$

We can prove by induction that $y_{t+1} = f_k^{-1}(x_{t+1})$, and therefore the two sequences converge at the same rate. The base case holds by definition and by induction we can see that:

$$\begin{aligned}
& y_{t+1} \\
= \quad & \{\text{Definition of } y_{t+1}\} \\
& y_t + \alpha u(l \circ f)(y_t) \\
= \quad & \{\text{Inductive step}\} \\
& f_k^{-1}(x_t) + \alpha f_k^{-1}(u(l)(x_t)) \\
= \quad & \{\text{Linearity of f and definition of } x_{t+1}\} \\
& f_k^{-1}(x_{t+1})
\end{aligned}$$

□

Propositions 4.8 and 4.9 together give some insight into why Newton's method can perform so much better than gradient descent for applications where both methods are computationally feasible (Boyd and Vandenberghe, 2004). When u is Newton's optimization scheme the convergence rate of the discrete system:

$$x_{t+1} = x_t + \alpha u(l)(x_t)$$

is not affected by invertible linear transformations of the data. We cannot make the same claim for the gradient descent optimization scheme.

4.1.3 Generalizing Optimization

In this section we will use Cartesian differential categories (Definition 3.2) and Cartesian reverse derivative categories (Definition 3.4) to generalize standard results on the behavior of gradient descent as well as the results in Section 4.1.2.

4.1.3.1 Optimization Domain

Definition 4.10. *An optimization domain is a tuple* $(\mathbf{Base}, X)$ *such that each morphism* $f : A \to B$ *in the Cartesian reverse derivative category* **Base** *has an additive inverse* $-f$ *where* $f + (-f) = 0_{AB}$ *and when* f *is linear then for any morphism* $g : C \to A$ *in* **Base** *we have* $f \circ -g = -(f \circ g)$.

Each homset $\mathbf{Base}[*, A]$ *out of the terminal object* $*$ *is further equipped with a multiplication operation* fg *and a multiplicative identity map* $1_A : * \to A$ *to form a commutative ring with the left additive structure* $+$.

X *is an object in* **Base** *such that the homset* $f \in \mathbf{Base}[*, X]$ *is further equipped with a total order* $f \leq g$ *to form an ordered commutative ring.*

Recall that an ordered commutative ring (ord, 2022) is a commutative ring r equipped with a total order $\leq$ where for a, b, c in r:

- if $a \leq b$ then $a + c \leq b + c$
- if $a \leq b$ and $0 \leq c$ then $ca \leq cb$

We abbreviate the map:

$$1_B \circ !_A : A \to B$$

as 1_{AB}, where $!_A : A \to *$ is the unique map into the terminal object $*$. Given an optimization domain $(\mathbf{Base}, X)$ the object X represents the space of objective values to optimize.

Definition 4.11. *An objective in the optimization domain* $(\mathbf{Base}, X)$ *is a morphism into* X *in* **Base**.

Section 4.1.2 implicitly focuses on the following optimization domain:

Definition 4.12. *The standard domain is the pair* $(\mathbf{Euc}, 1)$. *The objectives in the standard domain are infinitely differentiable functions* $l : \mathbb{R}^n \to \mathbb{R}$.

We will also build on Definition 3.5 to explore the following family of optimization domains:

Proposition 4.13. *Given an ordered commutative ring* r *the pair* $(\mathbf{Poly}_r, 1)$ *is an optimization domain, which we call the* r*-polynomial domain.*

The objectives in the r*-polynomial domain are the* r*-polynomials* $l_P : n \to 1$.

Proof. To start, note that $\mathbf{Poly}_r$ is a Cartesian reverse derivative category by Cockett et al. (2019).

Next, for any r-polynomial $f : n \to m$ we can form the additive inverse $-f : n \to m$ by replacing each coefficient in f with its additive inverse in r.

Next, since the set of polynomials with coefficients in the commutative ring r is itself a commutative ring, each homset $\mathbf{Poly}_r[*, n]$ is an n-tuple of commutative rings, which is itself a commutative ring.

Finally, the morphisms in the homset $\mathbf{Poly}_r[0, 1]$ are 0-variable polynomials with coefficients in r, which are just elements of r. Since r is an ordered commutative ring, this homset is an ordered commutative ring as well. □

In order to generalize the study of objective functions we use the following definition:

Definition 4.14. *An objective* $l : A \to X$ *is bounded below in* $(\mathbf{Base}, X)$ *if there exists some* $x : * \to X$ *such that for any* $a : * \to A$ *we have* $x \leq l \circ a$.

In both the standard and r-polynomial domains an objective is bounded below if its image has an infimum.

Definition 4.15. *The generalized gradient of the objective* $l : A \to X$ *in* $(\mathbf{Base}, X)$ *is:*

$$R[l]_1 : A \to A$$
$$R[l]_1 = R[l] \circ \langle id_A, 1_{AX} \rangle$$

The underscore-1 in the notation $R[l]_1$ is simply notational shorthand intended to remind the reader that the generalized gradient is determined by the multiplicative identity $1_X : * \to X$.

In the standard domain the generalized gradient of $l : \mathbb{R}^n \to \mathbb{R}$ is just the gradient:

$$R[l]_1(x) = R[l](x, 1) = \nabla l(x) \cdot 1 = \nabla l(x)$$

and in the r-polynomial domain the generalized gradient of $l_P : n \to 1$ is:

$$R[l_P]_1(x) = \left(\frac{\partial l_P}{\partial x[1]}(x), \cdots, \frac{\partial l_P}{\partial x[n]}(x) \right)$$

where $\frac{\partial l_P}{\partial x[i]}$ is the formal derivative of the polynomial l_P in the ith component $x[i]$ of the vector x.

Definition 4.16. *The generalized n-derivative of the morphism $f : X \to A$ in $(\mathbf{Base}, X)$ is $D_n[f] : X \to A$ where:*

$$D_1[f] = D[f] \circ \langle id_X, 1_{XX} \rangle$$
$$D_n[f] = D[D_{n-1}[f]] \circ \langle id_X, 1_{XX} \rangle$$

For any $f : \mathbb{R} \to \mathbb{R}$ in the standard domain the generalized n-derivative of f is:

$$D_n[f](x) = (D[D_{n-1}[f]] \circ \langle id_X, 1_{XX} \rangle)(x) = \frac{\partial f^{(n-1)}}{\partial x}(x) \cdot 1 = f^{(n)}(x)$$

where $f^{(n)}$ is the n-derivative of f. Similarly, in the r-polynomial domain the generalized n-derivative of $l_P : 1 \to 1$ is the formal n-derivative $\frac{\partial^n l_P}{\partial x^n}$.

The derivative over the reals has a natural interpretation as a rate of change. We can generalize this as follows:

Definition 4.17. *We say that a morphism $f : X \to X$ in* **Base** *is n-smooth in $(\mathbf{Base}, X)$ if for any choice of:*

$$t_1 : * \to X \qquad t_2 : * \to X$$

where $t_1 \leq t_2$ and for all $t_1 \leq t \leq t_2 : * \to X$ and $k \leq n$ we have:

$$D_k[f] \circ t : * \to X$$
$$D_k[f] \circ t \leq 0_X$$

then:

$$f \circ t_2 \leq f \circ t_1$$

Intuitively, f is n-smooth if it cannot increase on any interval over which its generalized derivatives of order n and below are non-positive. Some examples include:

- Any map $f : \mathbb{R} \to \mathbb{R}$ in **Euc** is 1-smooth in the standard domain by the mean value theorem.

- For any ordered commutative ring r, the polynomial:

$$l_P = \sum_{k=0}^{n} c_k t^k : 1 \to 1$$

of degree n is n-smooth in the r-polynomial domain since for any t_1 we can use the binomial theorem to write:

$$l_P(t) = l_P(t_1) + \sum_{k=1}^{n} c'_k (t - t_1)^k$$

where c'_k is a constant such that:

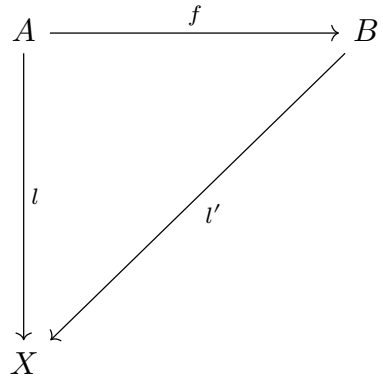
$$(c'_k)(k!) = D_k[l_P](t_1)$$

c'_k must exist by the definition of the formal derivative of l_P, and must be non-positive if $D_k[l_P](t_1)$ is non-positive.

4.1.3.2 Optimization Functors

In this section we generalize optimization schemes (Section 4.1.2.2) to arbitrary optimization domains. This will enable us to characterize the equivariance properties of our generalized optimization schemes in terms of the categories out of which they are functorial. Given an optimization domain $(\mathbf{Base}, X)$ we can define the following categories:

Definition 4.18. *The objects in the category* **Objective** *over the optimization domain* $(\mathbf{Base}, X)$ *are objectives* $l : A \to X$*. The morphisms from* $l : A \to X$ *to* $l' : B \to X$ *are morphisms* $f : A \to B$ *in* $\mathbf{Base}[A, B]$ *where* $l' \circ f = l$*.*



The composition of morphisms in **Objective** *is the same as in* **Base***.*

Objective is a subcategory of the slice category $\mathbf{Base}/X$.

In the standard domain the objects in **Objective** are objectives $l : \mathbb{R}^n \to \mathbb{R}$. In the r-polynomial domain, the objects in **Objective** are r-polynomials $l_P : n \to 1$.

Definition 4.19. *A generalized optimizer over the optimization domain* $(\mathbf{Base}, X)$ *with state space* $A \in \mathbf{Base}$ *and dimension* $k \in \mathbb{N}$ *is an endomorphism* $d : A^k \to A^k$ *in* **Base** *where:*

$$A^k = A \times A \times \cdots \times A$$

Definition 4.20. *The objects in the category* **Optimizer** *over* $(\mathbf{Base}, X)$ *are generalized optimizers, and the morphisms from the generalized optimizer* $d : A^k \to A^k$ *to the generalized optimizer* $d' : B^k \to B^k$ *are* **Base***-morphisms* $f : A \to B$ *such that:*

$$f^k \circ d : A^k \to B^k$$
$$f^k \circ d = d' \circ f^k$$

That is:

$$\begin{array}{ccc} A^k = A \times A \times \cdots \times A & \xrightarrow{f^k = f \times f \times \cdots \times f} & B^k = B \times B \times \cdots \times B \\ \Big\downarrow d & & \Big\downarrow d' \\ A^k = A \times A \times \cdots \times A & \xrightarrow{f^k = f \times f \times \cdots \times f} & B^k = B \times B \times \cdots \times B \end{array}$$

Morphisms in **Optimizer** *only exist between generalized optimizers with the same dimension. The composition of morphisms in* **Optimizer** *is the same as in* **Base**.

In the standard domain a generalized optimizer with dimension k is an optimizer $d : \mathbb{R}^{kn} \to \mathbb{R}^{kn}$ (Definition 4.1) with dimension k.

Definition 4.21. *Given a subcategory* $\mathbf{D}$ *of* **Objective**, *an optimization functor with dimension* k *over* $\mathbf{D}$ *is a functor* $\mathbf{D} \to$ **Optimizer** *that maps the objective* $l : A \to X$ *to a generalized optimizer* $d : A^k \to A^k$ *over* $(\mathbf{Base}, X)$ *with state space* A *and dimension* k. *Optimization functors act as the identity on morphisms.*

Optimization functors are generalizations of optimization schemes (Definition 4.6) that map objectives to generalized optimizers. Explicitly, an optimization scheme u that maps $l : \mathbb{R}^n \to \mathbb{R}$ to $u(l) : \mathbb{R}^{kn} \to \mathbb{R}^{kn}$ defines an optimization functor in the standard domain.

The equivariance properties of optimization functors are represented by the subcategory $\mathbf{D} \subseteq$ **Objective** out of which they are functorial. Concretely, consider the following categories over the optimization domain $(\mathbf{Base}, X)$:

Proposition 4.22. *We can form a subcategory* $\mathbf{Objective}_I$ *of* **Objective** *in which the morphisms from the objective:*

$$l : A \to X$$

to the objective:

$$l' : B \to X$$

are limited to invertible linear morphisms $f : A \to B$ *in* **Base** *where:*

$$(f^{-1})^\dagger = (f^\dagger)^{-1} = f^{-\dagger}$$

where $\dagger$ *is the involution over the linear maps in* **Base** *described in Section 3.1.1.2.*

We can also form a subcategory $\mathbf{Objective}_\perp$ *of* $\mathbf{Objective}_I$ *in which the inverse of the morphism* $f : A \to B$ *in* **Base** *is the morphism* $f^\dagger : B \to A$ *in* **Base**.

Proof. To start, note that $\mathbf{Objective}_I$ and $\mathbf{Objective}_\perp$ contain all identity morphisms since any identity morphism $id_A : A \to A$ in **Objective** is an invertible linear morphism where:

$$id_A = id_A^{-1} = id_A^\dagger$$

Next, for any three objects:

$$l : A \to X \qquad l' : B \to X \qquad l'' : C \to X$$

in $\mathbf{Objective}_I$ and morphisms:

$$f_1 : A \to B \in \mathbf{Objective}_I[l, l'] \qquad f_2 : B \to C \in \mathbf{Objective}_I[l', l'']$$

in $\mathbf{Objective}_I$ we need to show that:

$$f_2 \circ f_1 : A \to C$$

is a morphism in $\mathbf{Objective}_I$. First, since the linear maps in a reverse derivative category are closed under composition we have that $f_2 \circ f_1$ is linear. Next, note that $f_2 \circ f_1$ is invertible since:

$$(f_1^{-1} \circ f_2^{-1}) \circ (f_2 \circ f_1) = id_A$$
$$(f_2 \circ f_1) \circ (f_1^{-1} \circ f_2^{-1}) = id_C$$

Next, note that:

$$\begin{aligned}
& f_1^\dagger \circ f_2^\dagger \\
= \quad & \{\text{Rewrite } f_2^\dagger \text{ with projection } \pi \text{ and inclusion } \iota\} \\
& f_1^\dagger \circ (f_2^\dagger \circ \pi_1) \circ \langle f_1 \circ \pi_0, \pi_1 \rangle \circ \iota_1 \\
= \quad & \{\text{Rewrite } f_1^\dagger \text{ with projection } \pi \text{ and inclusion } \iota\} \\
& (f_1^\dagger \circ \pi_1) \circ (id_A \times (f_2^\dagger \circ \pi_1)) \circ \langle \pi_0, \langle f_1 \circ \pi_0, \pi_1 \rangle \rangle \circ \iota_1 \\
= \quad & \{\text{Apply } f^\dagger \circ \pi_1 = R[f] \text{ for linear } f\} \\
& R[f_1] \circ (id_A \times R[f_2]) \circ \langle \pi_0, \langle f_1 \circ \pi_0, \pi_1 \rangle \rangle \circ \iota_1 \\
= \quad & \{\text{Apply the chain rule}\} \\
& R[f_2 \circ f_1] \circ \iota_1 \\
= \quad & \{\text{Apply } R[f] \circ \iota_1 = f^\dagger\} \\
& (f_2 \circ f_1)^\dagger
\end{aligned}$$

Therefore we can show:

$$\begin{aligned}
& ((f_2 \circ f_1)^{-1})^\dagger \\
= \quad & \{\text{Definition of inverse}\} \\
& (f_1^{-1} \circ f_2^{-1})^\dagger \\
= \quad & \{\text{Apply } (f_2 \circ f_1)^\dagger = f_1^\dagger \circ f_2^\dagger\} \\
& (f_2^{-\dagger} \circ f_1^{-\dagger}) \\
= \quad & \{\text{Definition of inverse}\} \\
& (f_1^\dagger \circ f_2^\dagger)^{-1} = \\
= \quad & \{\text{Apply } f_1^\dagger \circ f_2^\dagger = (f_2 \circ f_1)^\dagger\} \\
& ((f_2 \circ f_1)^\dagger)^{-1}
\end{aligned}$$

Therefore $f_2 \circ f_1$ is a morphism in $\mathbf{Objective}_I$ and $\mathbf{Objective}_I$ is closed under composition.

Next, to show that $\mathbf{Objective}_\perp$ is closed under composition we need to show that when f_1, f_2 are morphisms in $\mathbf{Objective}_\perp$ then $f_2 \circ f_1$ is a morphism in $\mathbf{Objective}_\perp$ as well. We show this as follows:

$$
\begin{aligned}
& (f_2 \circ f_1)^{-1} \\
= \quad & \{\text{Definition of inverse}\} \\
& f_1^{-1} \circ f_2^{-1} \\
= \quad & \{\text{If } f \text{ is a morphism in } \mathbf{Objective}_\perp \text{ then } f^{-1} = f^\dagger\} \\
& f_1^\dagger \circ f_2^\dagger \\
= \quad & \{\text{Apply } f_1^\dagger \circ f_2^\dagger = (f_2 \circ f_1)^\dagger\} \\
& (f_2 \circ f_1)^\dagger
\end{aligned}
$$

□

In both the standard domain and r-polynomial domain, the morphisms in $\mathbf{Objective}_I$ are linear maps defined by an invertible matrix and the morphisms in $\mathbf{Objective}_\perp$ are linear maps defined by an orthogonal matrix (matrix inverse is equal to matrix transpose). We will now generalize Proposition 4.8 by defining generalized gradient descent and momentum functors.

Definition 4.23. *Generalized gradient descent sends the objective $l : A \to X$ to the generalized optimizer:*

$$-R[l]_1 : A \to A$$

with dimension 1*.*

Definition 4.24. *Generalized momentum sends the objective $l : A \to X$ to the generalized optimizer:*

$$\langle \pi_1, -\pi_1 - (R[l]_1 \circ \pi_0) \rangle : A^2 \to A^2$$

with dimension 2*.*

Generalized momentum and generalized gradient descent have a very similar structure, with the major difference between the two being that generalized momentum uses a placeholder variable and generalized gradient descent does not. In the standard domain we have that $-R[l]_1(x) = -\nabla l(x)$ and:

$$
\begin{aligned}
& \langle \pi_1, -\pi_1 - (R[l]_1 \circ \pi_0) \rangle : \mathbb{R}^2 \to \mathbb{R}^2 \\
& (\langle \pi_1, -\pi_1 - (R[l]_1 \circ \pi_0) \rangle)(x, y) = (y, -y - \nabla l(x))
\end{aligned}
$$

so generalized gradient descent and generalized momentum are equivalent to the gradient descent and momentum optimization schemes that we defined in Section 4.1.2.2. Similarly, in the r-polynomial domain generalized gradient descent maps $l_P : n \to 1$ to:

$$-R[l_P]_1(x) = -\left(\frac{\partial l_P}{\partial x[1]}(x), \cdots, \frac{\partial l_P}{\partial x[n]}(x) \right)$$

and generalized momentum maps l_P to:

$$\langle \pi_1, -\pi_1 - (R[l_P]_1 \circ \pi_0) \rangle : n^2 \to n^2$$
$$\langle \pi_1, -\pi_1 - (R[l_P]_1 \circ \pi_0) \rangle (x, y) = \left(y, -y - \left(\frac{\partial l_P}{\partial x[1]}(x), \cdots, \frac{\partial l_P}{\partial x[n]}(x) \right) \right)$$

We now generalize part of Proposition 4.8:

Theorem 4.25. *Both generalized gradient descent and generalized momentum are optimization functors (Definition 4.21) from* **Objective**$_\perp$ *to* **Optimizer**.

Proof. Since generalized gradient descent and generalized momentum act as the identity on morphisms, each of these maps trivially preserve composition and identity. Therefore, we simply need to show that each functor maps objects and morphisms in its source category to objects and morphisms in its target category.

First we show that generalized gradient descent:

$$GRAD(l) = -R[l]_1$$

is a functor out of **Objective**$_\perp$. Given an objective $l : A \to X$ in **Objective**$_\perp$ the map $GRAD(l) : A \to A$ is a morphism in **Base** since morphisms in **Base** are closed under the reverse derivative and $-$ operations. Therefore $GRAD(l) : A \to A$ is a generalized optimizer with state space A and dimension 1.

Next, given an objective $l' : B \to X$ in **Objective**$_\perp$ and a morphism in **Objective**$_\perp[l', l]$: that is, an invertible linear map $f : B \to A$ where $l \circ f = l'$ and:

$$f \circ f^\dagger = id_A$$
$$f^\dagger \circ f = id_B$$

we have:

$$\begin{aligned}
& f \circ GRAD(l') : B \to A \\
& f \circ GRAD(l') \\
= \quad & \{\text{Apply } l' = l \circ f\} \\
& f \circ GRAD(l \circ f) \\
= \quad & \{\text{Apply definition of } GRAD \text{ and linearity of } f\} \\
& -f \circ R[l \circ f]_1 \\
= \quad & \{\text{Apply definition of generalized gradient}\} \\
& -f \circ R[l \circ f] \circ \langle id_B, 1_{BX} \rangle \\
= \quad & \{\text{Apply chain rule}\} \\
& -f \circ R[f] \circ (id_B \times R[l]_1) \circ \langle id_B, f \rangle \\
= \quad & \{\text{Apply } f^\dagger \circ \pi_1 = R[f] \text{ for linear } f\} \\
& -f \circ f^\dagger \circ \pi_1 \circ (id_B \times R[l]_1) \circ \langle id_B, f \rangle \\
= \quad & \{\text{If } f \text{ is a morphism in } \mathbf{Objective}_\perp \text{ then } f^{-1} = f^\dagger\} \\
& -\pi_1 \circ (id_B \times R[l]_1) \circ \langle id_B, f \rangle \\
= \quad & \{\text{Apply projection}\} \\
& -R[l]_1 \circ f \\
= \quad & \{\text{Definition of } GRAD\} \\
& GRAD(l) \circ f
\end{aligned}$$

Next we show that generalized momentum:

$$MOM(l) = \langle \pi_1, -\pi_1 - (R[l]_1 \circ \pi_0)$$

is a functor out of $\mathbf{Objective}_\perp$. Given an objective $l : A \to X$ the map $MOM(l) : A^2 \to A^2$ is a morphism in **Base** since morphisms in **Base** are closed under the pairing, reverse derivative and $-$ operations. Therefore $MOM(l) : A^2 \to A^2$ is a generalized optimizer with state space A and dimension 2.

Next, given an objective $l' : B \to X$ in $\mathbf{Objective}_\perp$ and a morphism in $\mathbf{Objective}_\perp[l', l]$: that is, an invertible linear map $f : B \to A$ where $l \circ f = l'$ and:

$$f \circ f^\dagger = id_A$$
$$f^\dagger \circ f = id_B$$

we have:

$f^2 \circ MOM(l') : B^2 \to A^2$
$f^2 \circ MOM(l')$
$=$ {Apply $l' = l \circ f$ and rewrite}
$(f \times f) \circ MOM(l \circ f)$
$=$ {Definition of MOM}
$(f \times f) \circ \langle \pi_1, -\pi_1 - (R[l \circ f]_1 \circ \pi_0) \rangle$
$=$ {Apply composition}
$\langle f \circ \pi_1, f \circ (-\pi_1 - (R[l \circ f]_1 \circ \pi_0)) \rangle$
$=$ {Linearity of f}
$\langle f \circ \pi_1, -f \circ \pi_1 - (f \circ R[l \circ f]_1 \circ \pi_0) \rangle$
$=$ {Apply same steps as gradient descent above}
$\langle f \circ \pi_1, -f \circ \pi_1 - (R[l]_1 \circ f \circ \pi_0) \rangle$
$=$ {Simplify}
$\langle \pi_1, -\pi_1 - (R[l]_1 \circ \pi_0) \rangle \circ (f \times f)$
$=$ {Definition of MOM}
$MOM(l) \circ f^2$

□

Since Newton's method involves the computation of an inverse Hessian, it is not immediately obvious how we can express it in terms of Cartesian reverse derivatives. However, by the inverse function theorem we can rewrite the inverse Hessian as the Jacobian of the inverse gradient function, which makes this easier. That is:

$$\left[(\nabla^2 l)(x)\right]^{-1} = \left[J_{\nabla l}(x)\right]^{-1} = J_{(\nabla l)^{-1}}(\nabla l(x)) \tag{1}$$

where $J_{\nabla l}(x) = (\nabla^2 l)(x)$ is the Hessian of l at x, $J_{(\nabla l)^{-1}}(\nabla l(x))$ is the Jacobian of the inverse gradient function evaluated at $\nabla l(x)$, and the second equality holds by the inverse function theorem. We can therefore rewrite the update term in Newton's method as:

$$\left[-\nabla^2(l)(x)\right]^{-1} \nabla l(x) = J_{(\nabla l)^{-1}}(\nabla l(x)) \nabla l(x)$$

We can therefore generalize Newton's method as follows.

Definition 4.26. *The category* **Objective**$_{I^*}$ *is the subcategory of* **Objective**$_I$ *in which objects are limited to objectives* $l : A \to X$ *where* $R[l]_1$ *has an inverse* $R[l]_1^{-1}$ *in* **Base** *where:*

$$R[l]_1^{-1} \circ R[l]_1 : A \to A$$
$$R[l]_1^{-1} \circ R[l]_1 = R[l]_1 \circ R[l]_1^{-1} = id_A$$

Definition 4.27. *Consider some objective* $l : A \to X$ *in* **Base** *in* **Objective**$_{I^*}$*. Generalized Newton's method sends the objective* $l : A \to X$ *to the generalized optimizer:*

$$-R[R[l]_1^{-1}] \circ \langle R[l]_1, R[l]_1 \rangle : A \to A$$

with dimension 1*.*

The computation above suggests that generalized Newton's method in the standard domain is equivalent to Newton's optimization scheme that we defined in Section 4.1.2.2. In the r-polynomial domain generalized Newton's method maps the polynomial $l_P : n \to 1$ to:

$$- R[R[l_P]_1^{-1}] \circ \langle R[l_P]_1, R[l_P]_1 \rangle : n \to n$$
$$- R[R[l_P]_1^{-1}] \circ \langle R[l_P]_1, R[l_P]_1 \rangle(x) =$$
$$- \begin{pmatrix} \frac{\partial (R[l_P]_1^{-1}[1])}{\partial x[1]}(R[l_P]_1(x)) & \cdots & \frac{\partial (R[l_P]_1^{-1}[1])}{\partial x[n]}(R[l_P]_1(x)) \\ \vdots & \ddots & \\ \frac{\partial (R[l_P]_1^{-1}[n])}{\partial x[1]}(R[l_P]_1(x)) & & \frac{\partial (R[l_P]_1^{-1}[n])}{\partial x[n]}(R[l_P]_1(x)) \end{pmatrix}^T \left(\frac{\partial l_P}{\partial x[1]}(x), \cdots, \frac{\partial l_P}{\partial x[n]}(x) \right)$$

$R[l_P]_1^{-1}[i] : n \to 1$ is the ith projection of the inverse of the reverse derivative map and $\frac{\partial (R[l_P]_1^{-1}[i])}{\partial x[j]}$ is a formal derivative of this polynomial.

When $(\mathbf{Base}, X)$ is the standard domain many objectives $l : \mathbb{R}^n \to \mathbb{R}$ do not have invertible gradients over their entire domain and are therefore not objects in **Objective**$_{I^*}$. However, this is not actually required to apply Newton's method to l. For any point $a \in \mathbb{R}^n$ and neighborhood $\mathcal{N}(a)$ of a on which ∇l is invertible we can write the inverse of ∇l on this neighborhood as $(\nabla l)^{-1}{}_{\mathcal{N}(a)}$ and derive the update term:

$$J_{(\nabla l)^{-1}{}_{\mathcal{N}(a)}}(\nabla l(a)) \nabla l(a)$$

That is, taking a Newton's method step for the objective $l : \mathbb{R}^n \to \mathbb{R}$ at a point $a \in \mathbb{R}^n$ does not require that ∇l is invertible on all of $\mathbb{R}^n$. By the inverse function theorem we simply need ∇l to be continuously differentiable on a neighborhood of a and have a Jacobian with nonzero determinant at a.

Therefore, given a pair of an objective $l : \mathbb{R}^n \to \mathbb{R}$ in **Objective** and a point $a \in \mathbb{R}^n$ at which the determinant of $\nabla^2 l$ is nonzero we can construct an objective in **Objective**$_{I^*}$ from the restriction of l to $\mathcal{N}(a)$.

Theorem 4.28. *Generalized Newton's method is an optimization functor (Definition 4.21) from* **Objective**$_{I^*}$ *to* **Optimizer**.

Proof. Since Newton's method acts as the identity on morphisms, it trivially preserves composition and identity. Therefore, we simply need to show that this functor maps objects and morphisms in **Objective**$_{I^*}$ to objects and morphisms in **Optimizer**.

First we show that generalized Newton's method:

$$NEW(l) = -R[R[l]_1^{-1}] \circ \langle R[l]_1, R[l]_1 \rangle$$

is a functor out of $\mathbf{Objective}_{I^*}$. Given an objective $l : A \to X$ in $\mathbf{Objective}_{I^*}$ note that the map $NEW(l) : A \to A$ is a morphism in **Base** since $R[l]_1^{-1}$ is a morphism in **Base** by the definition of $\mathbf{Objective}_{I^*}$ and morphisms in **Base** are closed under the reverse derivative and $-$ operations. Therefore $NEW(l) : A \to A$ is a generalized optimizer with state space A and dimension 1.

Next, given an objective $l' : B \to X$ in $\mathbf{Objective}_{I^*}$ and an invertible linear map $f : B \to A$ in $\mathbf{Objective}_{I^*}[l', l]$ such that $l \circ f = l'$ we have:

$$\begin{aligned}
& f \circ NEW(l') : B \to A \\
& f \circ NEW(l') \\
= \quad & \{\text{Apply } l' = l \circ f \ \} \\
& f \circ NEW(l \circ f) \\
= \quad & \{\text{Definition of } NEW \text{ and linearity of } f\} \\
& - f \circ R[R[l \circ f]_1^{-1}] \circ \langle R[l \circ f]_1, R[l \circ f]_1 \rangle \\
= \quad & \{\text{Apply } \star \text{ below}\} \\
& - f \circ f^{-1} \circ R[R[l]_1^{-1}] \circ (f^{-\dagger} \times f^{-\dagger}) \circ \langle R[l \circ f]_1, R[l \circ f]_1 \rangle = \\
= \quad & \{\text{Apply } \star\star \text{ below}\} \\
& - f \circ f^{-1} \circ R[R[l]_1^{-1}] \circ (f^{-\dagger} \times f^{-\dagger}) \circ \langle f^\dagger \circ R[l]_1 \circ f, f^\dagger \circ R[l]_1 \circ f \rangle \\
= \quad & \{\text{Apply inverses}\} \\
& - R[R[l]_1^{-1}] \circ \langle f^{-\dagger} \circ f^\dagger \circ R[l]_1 \circ f, f^{-\dagger} \circ f^\dagger \circ R[l]_1 \circ f \rangle \\
= \quad & \{\text{Apply inverses}\} \\
& - R[R[l]_1^{-1}] \circ \langle R[l]_1, R[l]_1 \rangle \circ f \\
= \quad & \{\text{Definition of } NEW\} \\
& NEW(l) \circ f
\end{aligned}$$

where $\star$ holds by:

$$
\begin{aligned}
&R[R[l \circ f]_1^{-1}] : B \times B \to B \\
&R[R[l \circ f]_1^{-1}] \\
=\quad &\{\text{Apply } \star\star \text{ below}\} \\
&R[f^{-1} \circ R[l]_1^{-1} \circ f^{-\dagger}] \\
=\quad &\{\text{Apply the chain rule}\} \\
&R[f^{-\dagger}] \circ (id_B \times R[f^{-1} \circ R[l]_1^{-1}]) \circ \langle \pi_0, \langle f^{-\dagger} \circ \pi_0, \pi_1 \rangle \rangle \\
=\quad &\{\text{Apply } f^{\dagger} \circ \pi_1 = R[f] \text{ for linear } f\} \\
&f^{-\dagger\dagger} \circ \pi_1 \circ (id_B \times R[f^{-1} \circ R[l]_1^{-1}]) \circ \langle \pi_0, \langle f^{-\dagger} \circ \pi_0, \pi_1 \rangle \rangle \\
=\quad &\{\text{Apply the projections and } f^{\dagger\dagger} = f\} \\
&f^{-1} \circ R[f^{-1} \circ R[l]_1^{-1}] \circ \langle f^{-\dagger} \circ \pi_0, \pi_1 \rangle \\
=\quad &\{\text{Apply the chain rule}\} \\
&f^{-1} \circ R[R[l]_1^{-1}] \circ (id_A \times R[f^{-1}]) \circ \langle \pi_0, \langle R[l]_1^{-1} \circ \pi_0, \pi_1 \rangle \rangle \circ \langle f^{-\dagger} \circ \pi_0, \pi_1 \rangle \\
=\quad &\{\text{Apply } R[f] = f \circ \pi_1\} \\
&f^{-1} \circ R[R[l]_1^{-1}] \circ (id_A \times (f^{-\dagger} \circ \pi_1)) \circ \langle \pi_0, \langle R[l]_1^{-1} \circ \pi_0, \pi_1 \rangle \rangle \circ \langle f^{-\dagger} \circ \pi_0, \pi_1 \rangle \\
=\quad &\{\text{Apply projections}\} \\
&f^{-1} \circ R[R[l]_1^{-1}] \circ (id_A \times f^{-\dagger}) \circ \langle f^{-\dagger} \circ \pi_0, \pi_1 \rangle \\
=\quad &\{\text{Apply composition}\} \\
&f^{-1} \circ R[R[l]_1^{-1}] \circ (f^{-\dagger} \times f^{-\dagger})
\end{aligned}
$$

and where $\star\star$ holds by:

$$
\begin{aligned}
&R[l \circ f]_1^{-1} : B \to B \\
&R[l \circ f]_1^{-1} \\
=\quad &\{\text{Apply the definition of the generalized gradient and the chain rule}\} \\
&((R[f] \circ (id_B \times R[l]) \circ \langle \pi_0, \langle f \circ \pi_0, \pi_1 \rangle \rangle) \circ \langle id_B, 1_{BX} \rangle)^{-1} \\
=\quad &\{\text{Simplify}\} \\
&(R[f] \circ (id_B \times R[l]) \circ \langle id_B, \langle f, 1_{BX} \rangle \rangle)^{-1} \\
=\quad &\{\text{Apply the definition of the generalized gradient}\} \\
&(R[f] \circ \langle id_B, R[l]_1 \circ f \rangle)^{-1} \\
=\quad &\{\text{Apply } \star\star\star \text{ below}\} \\
&f^{-1} \circ R[l]_1^{-1} \circ R[f^{-1}] \circ (1_A \times id_B) \\
=\quad &\{\text{Apply } f^{\dagger} \circ \pi_1 = R[f] \text{ for linear } f\} \\
&f^{-1} \circ R[l]_1^{-1} \circ f^{-\dagger}
\end{aligned}
$$

and where $\star\star\star$ holds by:

$$
\begin{aligned}
& [R[f] \circ \langle id_B, R[l]_1 \circ f \rangle] \circ [f^{-1} \circ R[l]_1^{-1} \circ R[f^{-1}] \circ (1_A \times id_B)] \\
= \quad & \{\text{Apply composition}\} \\
& R[f] \circ \langle id_B, R[l]_1 \circ f \circ f^{-1} \circ R[l]_1^{-1} \circ R[f^{-1}] \circ (1_A \times id_B) \rangle \\
= \quad & \{\text{Apply the inverses}\} \\
& R[f] \circ \langle id_B, R[f^{-1}] \circ (1_A \times id_B) \rangle \\
= \quad & \{\text{Apply } f^\dagger \circ \pi_1 = R[f] \text{ for linear } f\} \\
& f^\dagger \circ \pi_1 \circ \langle id_B, R[f^{-1}] \circ (1_A \times id_B) \rangle \\
= \quad & \{\text{Apply projection}\} \\
& f^\dagger \circ R[f^{-1}] \circ (1_A \times id_B) \\
= \quad & \{\text{Apply } f^\dagger \circ \pi_1 = R[f] \text{ for linear } f\} \\
& f^\dagger \circ f^{-\dagger} \circ \pi_1 \circ (1_A \times id_B) \\
= \quad & \{\text{Apply the inverses}\} \\
& id_B
\end{aligned}
$$

□

Theorems 4.25 and 4.28 imply that the equivariance properties of our optimization functors mirror the equivariance properties of their optimization scheme counterparts.

Note that the proofs of Theorems 4.25 and 4.28 follow a similar flow to the proof of Proposition 4.8. However, since the generalized formulation of Newton's method generalizes the Jacobian of the inverse gradient function instead of the inverse Hessian, the proof of Theorem 4.28 requires several additional applications of the chain rule.

4.1.3.3 Generalized Optimization Flows

In Section 4.1.2 we demonstrated how we can derive continuous and discrete dynamical systems from an optimizer $d : \mathbb{R}^{kn} \to \mathbb{R}^{kn}$. In this section we extend this insight to generalized optimizers.

To do this, we define a morphism $s : X \to A^k$ whose generalized 1-derivative $D_1[s] : X \to A^k$ (Definition 4.16) factors through a generalized optimizer $d : A^k \to A^k$. Since $\mathbf{Base}[*, X]$ is totally ordered, we can interpret morphisms in this homset as either times $t : * \to X$ or objective values $x : * \to X$. Therefore we can interpret the morphism $s : X \to A^k$ as describing how the state of our dynamical system evolves in time. Formally we can put this together in the following structure:

Definition 4.29. *A generalized optimization flow over the optimization domain* $(\mathbf{Base}, X)$ *with state space* $A \in \mathbf{Base}$ *and dimension* $k \in \mathbb{N}$ *is a tuple* (l, d, s, τ) *where* $l : A \to X$ *is an objective,* $d : A^k \to A^k$ *is a generalized optimizer,* $s : X \to A^k$ *(which we call the state map) is a morphism in* $\mathbf{Base}$ *and* τ *is an interval in* $\mathbf{Base}[*, X]$ *such that for* $t \in \tau$ *we have:*

$$
\begin{aligned}
& d \circ s \circ t : * \to A^k \\
& d \circ s \circ t = D_1[s] \circ t
\end{aligned}
$$

Intuitively, l is an objective, d is a generalized optimizer, and the state map s maps times in τ to the system state such that $d \circ s : X \to A^k$ describes the generalized 1-derivative of the state map $D_1[s]$.

In the standard domain we can define a generalized optimization flow $(l, d, s, \mathbb{R})$ from an optimizer $d : \mathbb{R}^{kn} \to \mathbb{R}^{kn}$ and an initial state $s_0 \in \mathbb{R}^{kn}$ by defining a state map $s : \mathbb{R} \to \mathbb{R}^{kn}$ where:

$$s(\mathbf{t}) = s_0 + \int_0^{\mathbf{t}} d(s(t))dt$$

We can think of a state map in the standard domain as a simulation of Euler's method with infinitesimal α:

$$lim_{\alpha \to 0} s(t + \alpha) = lim_{\alpha \to 0} \left[s(t) + \alpha d(s(t)) \right]$$

Definition 4.30. *A generalized optimization flow (l, d, s, τ) over the optimization domain $(\mathbf{Base}, X)$ is an n-descending flow if for any $t \in \tau$ and $k \leq n$ we have:*

$$D_k[l \circ \pi_0 \circ s] \circ t : * \to X$$
$$D_k[l \circ \pi_0 \circ s] \circ t \leq 0_X$$

If (l, d, s, τ) is an n-descending flow and $l \circ \pi_0 \circ s : X \to X$ is n-smooth (Definition 4.17), then $l \circ \pi_0 \circ s$ must be monotonically decreasing in t on τ.

Definition 4.31. *The generalized optimization flow (l, d, s, τ) over the optimization domain $(\mathbf{Base}, X)$ converges if for any $\delta : * \to X$ where $\delta > 0_X$ there exists some $t \in \tau$ such that for any $t' \geq t \in \tau$ we have:*

$$-\delta \leq (l \circ \pi_0 \circ s \circ t') - (l \circ \pi_0 \circ s \circ t) \leq \delta$$

In the standard domain this reduces to a familiar definition of convergence that is similar to what Ang (2020) uses: a flow converges if there exists a time t after which the value of the objective l does not change by more than an arbitrarily small amount.

Now suppose (l, d, s, τ) is an n-descending flow, $l \circ \pi_0 \circ s : X \to X$ is n-smooth and l is bounded below (Definition 4.14). Since $l \circ \pi_0 \circ s$ must decrease monotonically in t, it must be that (l, d, s, τ) converges.

We will now explore some examples of optimization flows defined by the generalized gradient that satisfy these conditions.

Definition 4.32. *A generalized gradient flow is a generalized optimization flow of the form $(l, -R[l]_1, s, \tau)$.*

Given a smooth objective $l : \mathbb{R}^n \to \mathbb{R}$ an example generalized gradient flow in the standard domain is $(l, -\nabla l, s, \mathbb{R})$ where:

$$s(\mathbf{t}) = s_0 + \int_0^{\mathbf{t}} -\nabla l(s(t))\ dt$$

for some $s_0 \in \mathbb{R}^n$.

One of the most useful properties of a generalized gradient flow is that we can write its generalized 1-derivative with a structure that is similar to an inner product:

Proposition 4.33. *Given a choice of time $t \in \tau$ and a generalized gradient flow $(l, -R[l]_1, s, \tau)$ we can write the following:*

$$D_1[l \circ \pi_0 \circ s] \circ t : * \to X$$
$$D_1[l \circ \pi_0 \circ s] \circ t = -R[l]^{\dagger}_{s_t} \circ R[l]_{s_t} \circ 1_X$$

where:

$$R[l]_{s_t} : X \to A$$
$$R[l]_{s_t} = R[l] \circ \langle s \circ t \circ !_X, id_X \rangle$$

Proof. For:

$$D_1[l \circ s] \circ t : * \to X$$

we have that:

$$
\begin{aligned}
& D_1[l \circ s] \circ t \\
= \quad & \{\text{Definition of } D_1\} \\
& D[l \circ s] \circ \langle t, 1_X \rangle \\
= \quad & \{\text{Cartesian chain rule}\} \\
& D[l] \circ \langle s \circ \pi_0, D[s] \rangle \circ \langle t, 1_X \rangle \\
= \quad & \{\text{Pull } \langle t, 1_X \rangle \text{ in}\} \\
& D[l] \circ \langle s \circ \pi_0 \circ \langle t, 1_X \rangle, D[s] \circ \langle t, 1_X \rangle \rangle \\
= \quad & \{\text{Apply projection, pull } t \text{ out}\} \\
& D[l] \circ \langle s \circ t, D[s] \circ \langle id_X, 1_{XX} \rangle \circ t \rangle \\
= \quad & \{\text{Definition of } D_1\} \\
& D[l] \circ \langle s \circ t, D_1[s] \circ t \rangle \\
= \quad & \{\text{Definition of optimization flow, pull } t \text{ out}\} \\
& D[l] \circ \langle s, d \circ s \rangle \circ t \\
= \quad & \{\text{Definition of } d\} \\
& D[l] \circ \langle s, -R[l] \circ \langle id_A, 1_{AX} \rangle \circ s \rangle \circ t \\
= \quad & \{\text{Apply composition}\} \\
& - D[l] \circ \langle s, R[l] \circ \langle s, 1_{XX} \rangle \rangle \circ t \\
= \quad & \{\text{Rewrite } D \text{ in terms of } R\} \\
& - \pi_1 \circ R[R[l]] \circ (\langle id_A, 0_{AX} \rangle \times id_A) \circ \langle s, R[l] \circ \langle s, 1_{XX} \rangle \rangle \circ t \\
= \quad & \{\text{Apply composition}\} \\
& - \pi_1 \circ R[R[l]] \circ \langle \langle s, 0_{XX} \rangle, R[l] \circ \langle s, 1_{XX} \rangle \rangle \circ t \\
= \quad & \{\text{Move } t \text{ in}\} \\
& - \pi_1 \circ R[R[l]] \circ \langle \langle s \circ t, 0_X \rangle, R[l] \circ \langle s \circ t, 1_X \rangle \rangle \\
= \quad & \{\text{Decompose the rightmost term into a composition}\} \\
& - \pi_1 \circ R[R[l]] \circ (\langle s \circ t, 0_X \rangle \times id_A) \circ \langle id_*, R[l] \circ \langle s \circ t, 1_X \rangle \rangle \\
= \quad & \{\text{Definition of } R[l]_{s_t}\} \\
& - \pi_1 \circ R[R[l]] \circ (\langle s \circ t, 0_X \rangle \times id_A) \circ \langle id_*, R[l]_{s_t} \circ 1_X \rangle \\
= \quad & \{\text{Rewrite to pull out the } 0_X\} \\
& - \pi_1 \circ R[R[l]] \circ (\langle s \circ t \circ !_X, id_X \rangle \times id_A) \circ \langle 0_X, R[l]_{s_t} \circ 1_X \rangle \\
= \quad & \{\text{Reverse derivative of injection map}\} \\
& - R[R[l] \circ \langle s \circ t \circ !_X, id_X \rangle] \circ \langle 0_X, R[l]_{s_t} \circ 1_X \rangle \\
= \quad & \{\text{Apply } f^\dagger \circ \pi_1 = R[f] \text{ for linear } f\} \\
& - (R[l] \circ \langle s \circ t \circ !_X, id_X \rangle)^\dagger \circ \pi_1 \circ \langle 0_X, R[l]_{s_t} \circ 1_X \rangle \\
= \quad & \{\text{Apply projection}\} \\
& - (R[l] \circ \langle s \circ t \circ !_X, id_X \rangle)^\dagger \circ R[l]_{s_t} \circ 1_X \\
= \quad & \{\text{Definition of } R[l]_{s_t}\} \\
& - R[l]^\dagger_{s_t} \circ R[l]_{s_t} \circ 1_X
\end{aligned}
$$

□

Intuitively, $s \circ t : * \to A$ is the state at time t and $R[l]_{s_t} \circ 1_X : * \to A$ is the value of the generalized gradient of l at time t. To understand the importance of this result consider the following definition:

Definition 4.34. $(\mathbf{Base}, X)$ *supports generalized gradient based optimization when any generalized gradient flow over* $(\mathbf{Base}, X)$ *is a* 1*-descending flow.*

Intuitively, an optimization domain supports generalized gradient based optimization if loss decreases in the direction of the gradient. Proposition 4.33 is important because it helps us identify the optimization domains for which this holds.

Theorem 4.35. *Both the standard domain and any r-polynomial domain support generalized gradient based optimization.*

Proof. In the standard domain we have that:

$$
\begin{aligned}
& -R[l]^{\dagger}_{s_t} \circ R[l]_{s_t} \circ 1_{\mathbb{R}} \\
= \quad & \{\text{Definition of } R[l]_{s_t}\} \\
& -\nabla l(s(t))^T \nabla l(s(t)) \\
= \quad & \{\text{Vector dot product}\} \\
& -\|\nabla l(s(t))\|^2
\end{aligned}
$$

which must be non-positive by the definition of a norm. Recall that $x[i]$ is the ith component of the vector x. Then in the r-polynomial domain we have that:

$$
\begin{aligned}
& -R[l_P]^{\dagger}_{s_t} \circ R[l_P]_{s_t} \circ 1_1 \\
= \quad & \{\text{Definition of } R[l_P]_{s_t}\} \\
& -R[l_P]^{\dagger}_{s_t} \circ R[l_P]_1(s(t)) \\
= \quad & \{\text{Evaluate the composition}\} \\
& -\sum_{i=1}^{n} R[l_P]_1(s(t))[i]\ R[l_P]_1(s(t))[i] \\
= \quad & \{\text{Rewrite as formal derivative}\} \\
& -\sum_{i=1}^{n} \frac{\partial l_P}{\partial x[i]}(s(t)) \frac{\partial l_P}{\partial x[i]}(s(t))
\end{aligned}
$$

which must be non-positive since in an ordered ring no negative element is a square.

Therefore the statement holds by Proposition 4.33. □

4.1.4 Example and Experiment

We start this section with a demonstration of the structure and behavior of an example optimization flow. We then build on this example to define an algorithm for finding integer minima of multivariate polynomials. We demonstrate that this algorithm consistently outperforms random search.

4.1.4.1 Illustrative Example: Integer Polynomial State Map

Suppose $l_P : 1 \to 1$ is an objective and u is an optimization functor with dimension 1 in the integer polynomial domain. Given a choice of integer $x_0 \in \mathbb{Z}$, we can follow the pattern laid out in Section 4.1.2 and form a discrete dynamical system:

$$x_{t+1} = x_t + u(l_P)(x_t)$$

Now suppose that for some $\tau = 1, 2, \cdots, m$ we would like to construct an optimization flow $(l_P, u(l_P), s, \tau)$ that traces out the values of this dynamical system. The state map s must be an integer polynomial that satisfies two properties:

1. The integer polynomial s intersects the discrete dynamical system at each $t \in \tau$ and therefore:

$$s(t+1) = s(t) + u(l_P)(s(t))$$

2. By the definition of an optimization flow it must be that $u(l_P)$ defines the derivative of s. That is, for $t \in \tau$:

$$u(l_P)(s(t)) = \frac{\partial s}{\partial t}(t)$$

By Proposition 4.33 we expect that s will move towards a minima of l_P at each step.

There may be no, some, or an infinite number of integer polynomials:

$$s(t) = p_0 + p_1 t + p_2 t^2 + \cdots + p_n t^n$$

that satisfy these conditions. For example, consider the simple case in which:

$$l_P(x) = ax^2 + b$$
$$u(l_P) = -R[l_P] = -\frac{\partial l_P}{\partial x}$$

In this case the condition:

$$u(l_P)(s(t)) - \frac{\partial s}{\partial t}(t) = 0$$

becomes:

$$\begin{aligned}
& u(l_P)(s(t)) - \frac{\partial s}{\partial t}(t) \\
= \quad & \{\text{Definition of gradient descent optimizer}\} \\
& -\frac{\partial l_P}{\partial x}(s(t)) - \frac{\partial s}{\partial t}(t) \\
= \quad & \{\text{Compute derivative of } l_P\} \\
& -2as(t) - \frac{\partial s}{\partial t}(t) \\
= \quad & \{\text{Expand polynomials}\} \\
& -(2ap_0 + 2ap_1 t + 2ap_2 t^2 + \cdots + 2ap_n t^n) - (p_1 + 2p_2 t + 3p_3 t^2 + \cdots + np_n t^{n-1}) \\
= \quad & \{\text{Rearrange terms}\} \\
& (2ap_0 + p_1) + (2ap_1 + 2p_2)t + (2ap_2 + 3p_3)t^2 + \cdots + (2ap_{n-1} + np_n)t^{n-1} + 2ap_n t^n = 0
\end{aligned}$$

and the condition:

$$s(t+1) - s(t) - u(l_P)(s(t)) = 0$$

becomes:

$$\begin{aligned}
& s(t+1) - s(t) - u(l_P)(s(t)) \\
= & \quad \{\text{Definition of } u(l_P)\} \\
& s(t+1) - s(t) + \frac{\partial l_P}{\partial x}(s(t)) \\
= & \quad \{\text{Compute derivative of } l_P\} \\
& s(t+1) - s(t) + 2as(t) \\
= & \quad \{\text{Collapse the two } s(t) \text{ terms together}\} \\
& s(t+1) + (2a-1)s(t) \\
= & \quad \{\text{Expand out } s(t) \text{ and } s(t+1)\} \\
& (p_0 + p_1(t+1) + \cdots + p_n(t+1)^n) + ((2a-1)p_0 + (2a-1)p_1 t + \cdots + (2a-1)p_n t^n) \\
= & \quad \{\text{Collapse into a sum}\} \\
& \sum_{i=0}^{n} p_i((t+1)^i + (2a-1)t^i) = 0
\end{aligned}$$

Evaluated at each

$$t = 1, 2, \cdots, m-1, m$$

in τ this forms a linear Diophantine system with $2m+1$ unique equations. There are therefore infinitely many degree $2m+2$ polynomials s that satisfy these equations. We show two examples in Figure 10. As we would expect from Proposition 4.33, at each t the dynamical system takes a step in the direction of the value of $s(t)$ that minimizes $l_P(s(t))$.

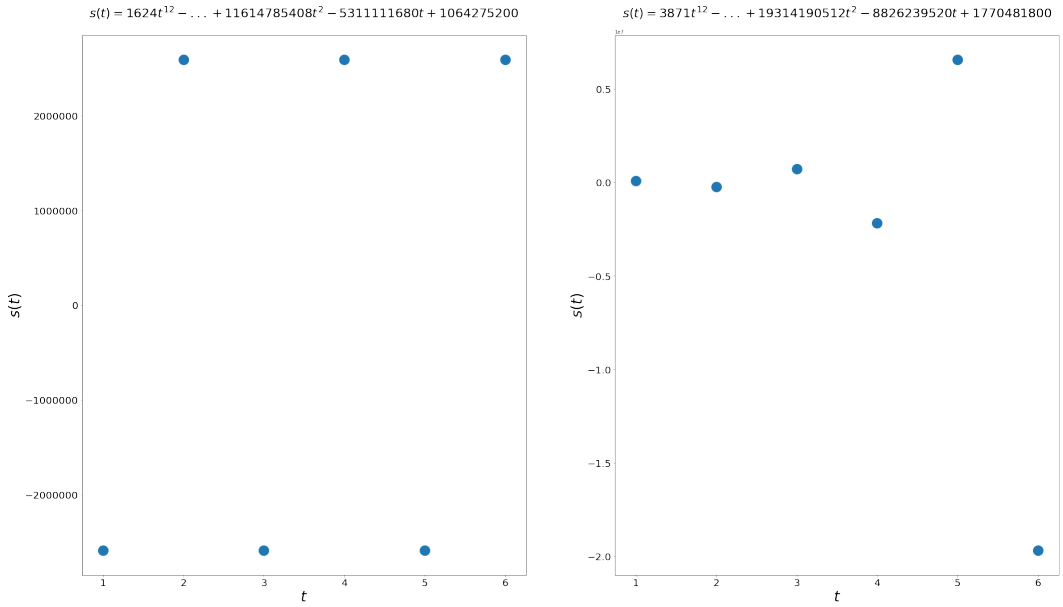


Figure 10: *Example integer polynomial state maps* $s(t) = p_1 t + p_2 t^2 + \cdots + p_n t^n$ *for* $\tau = 1, 2, 3, 4, 5, 6$. *If* $l_P = x^2$ *(left) and* $l_P = 2x^2 - 1$ *(right) then* $(l_P, -R[l_P]_1, s, \tau)$ *forms a gradient flow. The code to run this example is on GitHub (Shiebler, 2021a).*

4.1.4.2 Experiment: Integer Gradient Descent

Although Figure 10 shows that each step the dynamical system takes is in the right direction, these steps are too large to minimize the function, which can cause s to oscillate or diverge. In the standard domain we would mitigate this problem by choosing a smaller step size (learning rate) α for the dynamical system:

$$s(t+1) = s(t) - \alpha \frac{\partial l_P}{\partial x}(s(t))$$

but this is not possible in the integer polynomial domain since there are no α between 0 and 1. Instead, we can simply modify the dynamical system to instead take steps of size 1 in the direction of the negative gradient.

We can assess how well this method performs at finding integer minima of arbitrary multivariate polynomials by testing it on randomly generated polynomials. In Table 2 we show that this method consistently outperforms random search at finding minima of polynomials that can be written as a sum of squared terms (which are guaranteed to have both a global minimum and a global integer minimum).

Number of Steps (N)	Frequency that Integer Gradient Descent is Better than Random Search
5	0.808 ($\pm$0.023)
10	0.868 ($\pm$0.016)
50	0.891 ($\pm$0.012)
100	0.897 ($\pm$0.011)

Table 2: *In this experiment we randomly generate polynomials l_P as sums of squared terms and use both integer gradient descent and random search to minimize l_P. In integer gradient descent we sample a point, compute the gradient at that point, take a step in the direction of the gradient, and repeat this process N times. In random search we sample N points (log-uniformly) and choose the best one. We then compute the frequency with which gradient descent finds a better x than random search. The number of variables, terms, and coefficient values are sampled from $[1, 10]$. Means and two standard error confidence bounds from 100 experiments of 10 polynomials each are shown. The code to run these experiments is on GitHub (Shiebler, 2021a).*

4.1.5 Closing Thoughts on Generalized Optimization

Although there has been a tremendous amount of recent progress on categorical generalizations of machine learning, there has been relatively little research on the properties of these generalized algorithms. That is, although categorical machine learning has started to gain traction, categorical learning theory is still far behind. We aim to reduce this gap by exploring the properties of optimizers generalized over other categories such as $\mathbf{Poly}_r$.

We believe that this line of research will accelerate future machine learning research by helping researchers better understand the foundational components of the algorithms that they use. This generalized perspective may also help us better understand the domains over which different algorithms will be successful.

4.2 Optimizers as Dynamical Systems

In Section 4.1 we described how the application of algorithms like gradient descent and momentum to optimize a parametric function describes a dynamical system whose state is the function parameters. In this section we dig deeper into this perspective.

First, we introduce a category theoretic framework by Jaz Myers (2021) in which dynamical systems are a special case of lenses. We use this framework to express the dynamical systems defined by gradient descent (Definition 2.12), momentum (Definition 2.14), stochastic gradient descent (Definition 2.17), and stochastic momentum (Definition 2.18) as lenses. Through this framework we explore how we can leverage the composition of dynamical systems to construct complex optimization algorithms from simpler components. In particular, we demonstrate that momentum and stochastic momentum can be constructed from the composition of gradient descent with some basic lens operations.

4.2.1 Dynamical Systems as Lenses

Jaz Myers (2021) introduces the following characterization of dynamical systems as lenses:

Definition 4.36. *A discrete system is a* **Set***-lens (Definition 2.7):*

$$\binom{S}{S} \xrightarrow{(f_g, f_p)} \binom{I}{O}$$

or equivalently, a set of states S, a set of inputs I, a set of outputs O, and two functions:

- *A get (read) function $f_g : S \to O$ that generates an output from a state.*
- *A put (update) function $f_p : S \times I \to S$ that takes a pair of a state and an input and returns an updated state.*

Intuitively, a discrete system represents the stepwise application of the update function f_p, potentially in response to a sequence of inputs. That is, the discrete system:

$$\binom{S}{S} \xrightarrow{(f_g, f_p)} \binom{I}{O}$$

describes a dynamical system whose state $x_{S_t} \in S$ at time $t+1$ is described by the equation:

$$x_{S_{t+1}} = x_{S_t} + f_p(x_{S_t}, x_{I_t})$$

where $x_{I_t} \in I$ is the system input at time t.

For example, we can represent gradient descent (Definition 2.12) over a function $l : \mathbb{R}^p \to \mathbb{R}$ with a discrete system that accepts no inputs:

Definition 4.37. *Given a function $l : \mathbb{R}^p \to \mathbb{R}$ the closed gradient descent dynamical system (g_g, g_p) has the following structure:*

$$\binom{\mathbb{R}^p}{\mathbb{R}^p} \xrightarrow{(g_g, g_p)} \binom{*}{\mathbb{R}^p}$$
$$g_g(x_p) = x_p$$
$$g_p(x_p, *) = -\nabla l(x_p)$$

We can also represent momentum (Definition 2.14) over l with a discrete system that accepts no inputs:

Definition 4.38. *Given a function $l : \mathbb{R}^p \to \mathbb{R}$ the closed momentum dynamical system m has the following structure:*

$$\binom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{(m_g, m_p)} \binom{*}{\mathbb{R}^p}$$
$$m_g((x_p, x'_p)) = x_p$$
$$m_p((x_p, x'_p), *) = (x'_p, -x'_p - \nabla l(x_p))$$

4.2.2 Constructing Momentum from Gradient Descent

One benefit of this characterization of dynamical systems is that we can construct more complex dynamical systems from simpler component parts. We can use the parallel product of lenses (Definition 2.8) to model dynamical systems executing in parallel, and we can use the composition of lenses (Definition 2.9) to connect dynamical systems together.

For example, we can construct the momentum dynamical system in terms of the tensor and composition of the gradient descent dynamical system with a series of basic lenses. To start, consider the following discrete systems:

Definition 4.39. *The discrete system add has the following structure:*

$$\begin{pmatrix}\mathbb{R}^p\\ \mathbb{R}^p\end{pmatrix} \xrightarrow{(add_g, add_p)} \begin{pmatrix}\mathbb{R}^p\times\mathbb{R}^p\\ \mathbb{R}^p\end{pmatrix}$$
$$add_g(x_p) = x_p$$
$$add_p(x_p, (x'_p, x''_p)) = x'_p + x''_p$$

The *add* system reads the state directly and uses the sum of the input values as the state update.

Definition 4.40. *The discrete system cp has the following structure:*

$$\begin{pmatrix}\mathbb{R}^p\\ \mathbb{R}^p\end{pmatrix} \xrightarrow{(cp_g, cp_p)} \begin{pmatrix}*\\ \mathbb{R}^p\end{pmatrix}$$
$$cp_g(x_p) = x_p$$
$$cp_p(x_p, *) = x_p$$

The *cp* system reads the state directly and uses the current state as the state update.

Definition 4.41. *The discrete system sw has the following structure:*

$$\begin{pmatrix}\mathbb{R}^p\times\mathbb{R}^p\\ \mathbb{R}^p\times\mathbb{R}^p\end{pmatrix} \xrightarrow{(sw_g, sw_p)} \begin{pmatrix}\mathbb{R}^p\times\mathbb{R}^p\\ \mathbb{R}^p\times\mathbb{R}^p\end{pmatrix}$$
$$sw_g(x_p, x'_p) = (x_p, x'_p)$$
$$sw_p((x_p, x'_p), (x''_p, x'''_p)) = (x'''_p, x''_p)$$

The *sw* dynamical system reads the state directly and swaps the positions of the input values to generate the state update.

Consider also the following **Set**-lenses:

Definition 4.42. *The lens ng has the following structure*

$$\begin{pmatrix}\mathbb{R}^p\\ *\end{pmatrix} \xrightarrow{(ng_g, ng_p)} \begin{pmatrix}\mathbb{R}^p\\ *\end{pmatrix}$$
$$ng_g(*) = *$$
$$ng_p(*, x_p) = -x_p$$

ng uses the negated input value as the state update.

Definition 4.43. *The lens rp has the following structure:*

$$\begin{pmatrix}\mathbb{R}^p\\ \mathbb{R}^p\times\mathbb{R}^p\end{pmatrix} \xrightarrow{(rp_g, rp_p)} \begin{pmatrix}*\\ \mathbb{R}^p\end{pmatrix}$$
$$rp_g(x_p, x'_p) = x_p$$
$$rp_p((x_p, x'_p), *) = x'_p$$

rp reads the left component of the system state and uses the right component to generate the state update.

We can now construct momentum from these components. To start, draw the composition:

$$\begin{pmatrix}\mathbb{R}^p\\ \mathbb{R}^p\end{pmatrix} \xrightarrow{(((g\times ng)\circ add)_g, ((g\times ng)\circ add)_p)} \begin{pmatrix}\mathbb{R}^p\\ \mathbb{R}^p\end{pmatrix}$$

where:

$$
\begin{aligned}
&((g \times ng) \circ add)_g : \mathbb{R}^p \to \mathbb{R}^p \\
&((g \times ng) \circ add)_g(x_p) \\
=\quad &\{\text{Composition of get maps}\} \\
&(g \times ng)_g(add_g(x_p)) \\
=\quad &\{\text{Compute } add_g\} \\
&(g \times ng)_g(x_p) \\
=\quad &\{\text{Compute } g_g \text{ and } ng_g\} \\
&x_p
\end{aligned}
$$

and:

$$
\begin{aligned}
&((g \times ng) \circ add)_p : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^p \\
&((g \times ng) \circ add)_p(x_p, c_p) \\
=\quad &\{\text{Composition of put maps}\} \\
&add_p(x_p, (g \times ng)_p(add_g(x_p), c_p)) \\
=\quad &\{\text{Apply } (g \times ng)_p\} \\
&add_p(x_p, g_p(add_g(x_p)), ng_p(c_p)) \\
=\quad &\{\text{Compute } add_g \text{ and } ng_p\} \\
&add_p(x_p, g_p(x_p), -c_p) \\
=\quad &\{\text{Compute } g_p\} \\
&add_p(x_p, -\nabla l(x_p), -c_p) \\
=\quad &\{\text{Compute } add_p\} \\
&-c_p - \nabla l(x_p)
\end{aligned}
$$

We can also draw the map $((g \times ng) \circ add)_p$ as follows:

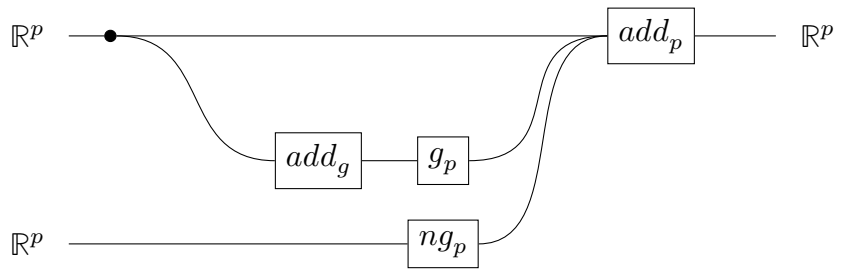


Figure 11: $(g \times ng) \circ add)_p : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^p$

We can build on this to form:

$$
\begin{aligned}
&\tbinom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{((((g \times ng) \circ add) \times cp)_g, (((g \times ng) \circ add) \times cp)_p)} \tbinom{\mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \\
&(((g \times ng) \circ add) \times cp)_g(x_p, x'_p) = (x_p, x'_p) \\
&(((g \times ng) \circ add) \times cp)_p((x_p, x'_p), c_p) = (-c_p - \nabla l(x_p), x'_p)
\end{aligned}
$$

and:

$$\binom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{(((((g \times ng) \circ add) \times cp) \circ sw)_g, ((((g \times ng) \circ add) \times cp) \circ sw)_p)} \binom{\mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p}$$

$$((((g \times ng) \circ add) \times cp) \circ sw)_g(x_p, x'_p) = (x_p, x'_p)$$

$$((((g \times ng) \circ add) \times cp) \circ sw)_p((x_p, x'_p), c_p) = (x'_p, -c_p - \nabla l(x_p))$$

Putting it all together we have the following.

Proposition 4.44. *We can express the closed momentum dynamical system*

$$\binom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{(m_g, m_p)} \binom{*}{\mathbb{R}^p}$$

as the lens composition:

$$m = rp \circ ((((g \times ng) \circ add) \times cp) \circ sw$$

Proof. We have:

$$\begin{aligned}
&(rp \circ ((((g \times ng) \circ add) \times cp) \circ sw))_g : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^p \\
&(rp \circ ((((g \times ng) \circ add) \times cp) \circ sw))_g(x_p, x'_p) \\
= &\quad \{\text{Apply get composition}\} \\
&rp_g(((((g \times ng) \circ add) \times cp) \circ sw)_g(x_p, x'_p)) \\
= &\quad \{\text{Apply steps above}\} \\
&rp_g(x_p, x'_p) \\
= &\quad \{\text{Compute } rp_g\} \\
&x_p \\
= &\quad \{\text{Definition of } m_g\} \\
&m_g(x_p, x'_p)
\end{aligned}$$

and:

$$\begin{aligned}
&(rp \circ ((((g \times ng) \circ add) \times cp) \circ sw))_p : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^p \times \mathbb{R}^p \\
&(rp \circ ((((g \times ng) \circ add) \times cp) \circ sw))_p((x_p, x'_p), *) \\
= &\quad \{\text{Apply steps above}\} \\
&(x'_p, -x'_p - \nabla l(x_p)) \\
= &\quad \{\text{Definition of } m_p\} \\
&m_p((x_p, x'_p), *)
\end{aligned}$$

□

We can also draw this as follows. Note that $((((g \times ng) \circ add) \times cp) \circ sw)_g = id$:

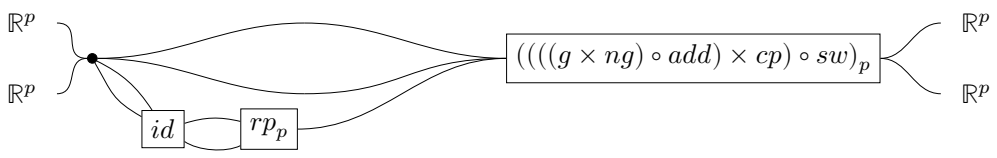


Figure 12: $m_p : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^p \times \mathbb{R}^p$

4.2.3 Constructing Open Momentum from Open Gradient Descent

In many applications we will not have a fixed dataset to optimize over, and will instead receive data as a stream. For example, consider a system for forecasting a time series in $\mathbb{R}$. At each time t this system will use the state of the world (represented as a vector in $\mathbb{R}^a$) to predict what the value of the time series (a real number) will be at time $t+1$. At time $t+1$ the system will receive information about the correct value of the time series at time $t+1$, represented as a pair $(x_a, y) \in \mathbb{R}^a \times \mathbb{R}$, and will need to predict the value of the time series at time $t+2$.

One way to build such a system would be to choose a loss function $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and use gradient descent (Definition 2.12) or momentum (Definition 2.14) to train a model $f : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}$ on all of the data before some time t. We could then use that trained model to generate predictions at $t+n$. If the time series is not stationary then this may produce poor results for $t+n$ where n is large. Another option would be to continuously train the model f as each new sample $(x_a, y) \in \mathbb{R}^a \times \mathbb{R}$ is observed. This is known as online learning (Boyd and Vandenberghe, 2004), and we can use algorithms like stochastic gradient descent (Definition 2.17) or stochastic momentum (Definition 2.18) to apply this strategy. We can represent this with the following dynamical systems:

Definition 4.45. *Given a pair of a loss function $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and inference function $f : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}$ the open gradient descent dynamical system sg has the following structure:*

$$\binom{\mathbb{R}^p}{\mathbb{R}^p} \xrightarrow{(sg_g, sg_p)} \binom{\mathbb{R}^a \times \mathbb{R}}{\mathbb{R}^p}$$
$$sg_g(x_p) = x_p$$
$$sg_p(x_p, (x_a, y)) = -\nabla l(f(x_p, x_a), y)$$

Unlike closed gradient descent, the open gradient descent system accepts an input in $\mathbb{R}^a \times \mathbb{R}$. We can think of the open gradient descent system as iteratively updating the parameters x_p each time a new sample $(x_a, y) \in \mathbb{R}^a \times \mathbb{R}$ is observed.

We can also define a dynamical system to represent stochastic momentum:

Definition 4.46. *Given a pair of a loss function $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and inference function $f : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}$ the open momentum dynamical system sm has the following structure:*

$$\binom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{(sm_g, sm_p)} \binom{\mathbb{R}^a \times \mathbb{R}}{\mathbb{R}^p}$$
$$sm_g((x_p, x'_p)) = x_p$$
$$sm_p((x_p, x'_p), (x_a, y)) = (x'_p, -x'_p - \nabla l(f(x_p, x_a), y))$$

Unlike closed momentum, the open momentum system accepts an input in $\mathbb{R}^a \times \mathbb{R}$.

Like in Section 4.2.2, we can construct open momentum from the composition and tensor of open gradient descent with some basic lenses. To start, consider the following lens:

Definition 4.47. *The lens srp has the following structure:*

$$\binom{\mathbb{R}^a \times \mathbb{R} \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{(srp_g, srp_p)} \binom{\mathbb{R}^a \times \mathbb{R}}{\mathbb{R}^p}$$
$$srp_g(x_p, x'_p) = x_p$$
$$srp_p((x_p, x'_p), (x_a, y)) = (x_a, y, x'_p)$$

Intuitively, *srp* behaves like *rp* (Definition 4.43), except it persists the input in the update step. We can now construct the open momentum dynamical system from these components. To start, draw the composition:

$$\binom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p} \xrightarrow{((sg\times ng)_g,(sg\times ng)_p)} \binom{\mathbb{R}^a \times \mathbb{R} \times \mathbb{R}^p}{\mathbb{R}^p}$$

and then:

$$\binom{\mathbb{R}^p}{\mathbb{R}^p} \xrightarrow{(((sg\times ng)\circ add)_g,((sg\times ng)\circ add)_p)} \binom{\mathbb{R}^a \times \mathbb{R} \times \mathbb{R}^p}{\mathbb{R}^p}$$

where:

$$\begin{aligned}
&((sg \times ng) \circ add)_g : \mathbb{R}^p \to \mathbb{R}^p \\
&((sg \times ng) \circ add)_g(x_p) \\
=\quad &\{\text{Get composition}\} \\
&(sg \times ng)_g(add_g(x_p)) \\
=\quad &\{\text{Compute } add_g\} \\
&(sg \times ng)_g(x_p) \\
=\quad &\{\text{Compute } sg_g\} \\
&x_p
\end{aligned}$$

and:

$$\begin{aligned}
&((sg \times ng) \circ add)_p : \mathbb{R}^p \times \mathbb{R}^a \times \mathbb{R} \times \mathbb{R}^p \to \mathbb{R}^p \\
&((sg \times ng) \circ add)_p(x_p, ((x_a, y), c_p)) \\
=\quad &\{\text{Definition of put composition}\} \\
&add_p(x_p, (sg \times ng)_p(add_g(x_p), ((x_a, y), c_p))) \\
=\quad &\{\text{Apply } (sg_p \times ng_p)\} \\
&add_p(x_p, sg_p(add_g(x_p), (x_a, y)), ng_p(c_p)) \\
=\quad &\{\text{Compute } add_g \text{ and } ng_p\} \\
&add_p(x_p, sg_p(x_p, (x_a, y)), -c_p) \\
=\quad &\{\text{Compute } sg_p\} \\
&add_p(x_p, -\nabla l(f(x_p, x_a), y), -c_p) \\
=\quad &\{\text{Compute } add_p\} \\
&-c_p - \nabla l(f(x_p, x_a), y)
\end{aligned}$$

We can also draw this as follows:

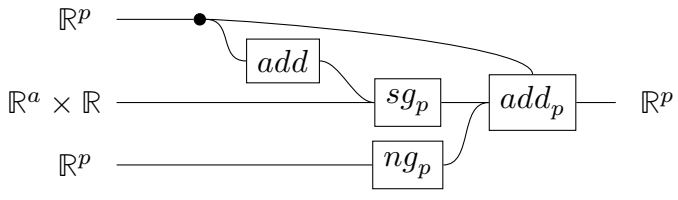


Figure 13: $((sg \times ng) \circ add)_p : \mathbb{R}^p \times \mathbb{R}^a \times \mathbb{R} \times \mathbb{R}^p \to \mathbb{R}^p$

We can build on this to form:

$$\binom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{((((sg \times ng) \circ add) \times cp)_g, (((sg \times ng) \circ add) \times cp)_p)} \binom{\mathbb{R}^a \times \mathbb{R} \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p}$$

$$(((sg \times ng) \circ add) \times cp)_g(x_p, x'_p) = (x_p, x'_p)$$

$$(((sg \times ng) \circ add) \times cp)_p((x_p, x'_p), (x_a, y, c_p))) = (-c_p - \nabla l(f(x_p, x_a), y), x'_p)$$

and:

$$\binom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{(((((sg \times ng) \circ add) \times cp) \circ sw)_g, ((((sg \times ng) \circ add) \times cp) \circ sw)_p)} \binom{\mathbb{R}^a \times \mathbb{R} \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p}$$

$$((((sg \times ng) \circ add) \times cp) \circ sw)_g(x_p, x'_p) = (x_p, x'_p)$$

$$((((sg \times ng) \circ add) \times cp) \circ sw)_p((x_p, x'_p), (x_a, y, c_p))) = (x'_p, -c_p - \nabla l(f(x_p, x_a), y))$$

Putting it all together we have the following.

Proposition 4.48. *We can express the open momentum dynamical system*

$$\binom{\mathbb{R}^p \times \mathbb{R}^p}{\mathbb{R}^p \times \mathbb{R}^p} \xrightarrow{(sm_g, sm_p)} \binom{\mathbb{R}^a \times \mathbb{R}}{\mathbb{R}^p}$$

as the lens composition:

$$sm = (srp \circ ((((sg \times ng) \circ add) \times cp) \circ sw))$$

Proof. We have:

$$(srp \circ ((((sg \times ng) \circ add) \times cp) \circ sw))_g : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^p$$

$$(srp \circ ((((sg \times ng) \circ add) \times cp) \circ sw))_g(x_p, x'_p)$$

$$= \quad \{\text{Apply steps above and compute } srp_g\}$$

$$x_p$$

$$= \quad \{\text{Definition of } sm_g\}$$

$$sm_g((x_p, x'_p))$$

and:

$$(srp \circ ((((sg \times ng) \circ add) \times cp) \circ sw))_p : \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^a \times \mathbb{R} \to \mathbb{R}^p \times \mathbb{R}^p$$

$$(srp \circ ((((sg \times ng) \circ add) \times cp) \circ sw))_p((x_p, x'_p), (x_a, y))$$

$$= \quad \{\text{Apply steps above}\}$$

$$(x'_p, -x'_p - \nabla l(f(x_p, x_a), y))$$

$$= \quad \{\text{Definition of } sm_p\}$$

$$sm_p((x_p, x'_p), (x_a, y))$$

□

We can also draw this as follows. Note that $((((sg \times ng) \circ add) \times cp) \circ sw)_g = id$:

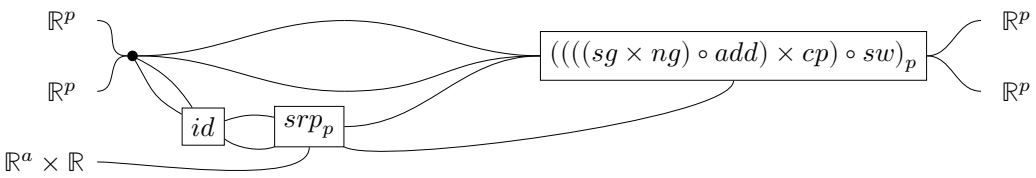


Figure 14: $sm_p : \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^a \times \mathbb{R} \to \mathbb{R}^p \times \mathbb{R}^p$

4.2.4 Generalizing to Differential Systems

Another major benefit of this construction is that the same theory and operations work equally well for differential systems as for discrete dynamical systems. Jaz Myers (2021) defines a differential system as follows:

Definition 4.49. *A differential system is a* **Euc***-lens (Definition 2.7):*

$$\binom{S}{S} \xrightarrow{(f_g, f_p)} \binom{I}{O}$$

or equivalently, a state space $\mathbb{R}^S$, an input space $\mathbb{R}^I$, an output space $\mathbb{R}^O$, and two smooth functions:

- *A get (read) function $f_g : \mathbb{R}^S \to \mathbb{R}^O$ that generates an output from a state.*
- *A put (update) function $f_p : \mathbb{R}^S \times \mathbb{R}^I \to \mathbb{R}^S$ that describes the derivatives of the system state.*

Intuitively, a differential system represents the continuous evolution of a system over time, potentially in response to an input process. That is, the differential system:

$$\binom{S}{S} \xrightarrow{(f_g, f_p)} \binom{I}{O}$$

describes a dynamical system whose state $x_{S_t} \in \mathbb{R}^S$ at time t has the derivative:

$$\frac{\partial x_{S_t}}{\partial t} = f_p(x_{S_t}, x_{I_t})$$

where $x_{I_t} \in \mathbb{R}^I$ is the value of the input process at time t.

That is, we can equally interpret the lens corresponding to closed gradient descent:

$$g_g(x_p) = x_p$$
$$g_p(x_p, *) = -\nabla l(x_p)$$

as the discrete system in which $t \in \mathbb{N}$ and:

$$x_{p_{t+1}} = x_{p_t} - \nabla l(x_{p_t})$$

or the differential system in which $t \in \mathbb{R}_{\geq 0}$ and:

$$\frac{\partial x_p}{\partial t} = -\nabla l(x_p)$$

4.2.5 Closing Thoughts on Optimizers as Dynamical Systems

In this section we explored how we can leverage the composition of dynamical systems to construct complex optimization algorithms from simpler components. In particular, we demonstrated that the momentum and stochastic momentum optimization algorithms can be constructed from nothing more than gradient descent and some basic lens operations. This enables us to better understand the flow of information in these optimization algorithms. We could likely build on this perspective to represent the optimization hyperparameters or the configuration of the data generation process as the dynamical system input.

5 Probability

In Chapter 3 we explored how some authors have focused on constructing category theoretic frameworks for optimizing neural networks with automatic differentiation, whereas other authors have explored category theoretic frameworks for constructing probabilistic models from data. We aim to bridge these streams of research by using a probabilistic construction to define an optimization objective. This will enable us to capture the shared structure between optimization task and optimization process.

5.1 Introduction

Many machine learning algorithms contain an irreducible aspect of randomness. Using category theory to reason about how this randomness breaks down into compositional and functorial structure helps us build a high-level picture of probabilistic learning and its connections to other fields. There are two kinds of uncertainty that most probabilistic reasoning aims to capture.

The first is epistemic uncertainty, or uncertainty due to limited data or knowledge. If we have a very small amount of data then we need to cope with high epistemic uncertainty. Cho and Jacobs (2019) and Culbertson and Sturtz (2014, 2013) explore how new data points affect their models' epistemic uncertainty. For example, a simple model of a complex nonlinear system is likely to have high epistemic uncertainty.

The second is aleatoric uncertainty, or the inherent uncertainty in a system that can cause results to differ each time we run the same experiment. If we aim to predict the output of a system that includes a non-deterministic stage (such as a coin toss), we will need to cope with aleatoric uncertainty. Aleatoric uncertainty is common in physical systems. For example, many biological processes will produce slightly different results based on randomness in turbulent fluid flows. For this reason, models that approximate physical systems often implicitly or explicitly produce a probability distribution over the possible outputs conditioned on some input (Walker et al., 2020).

Even models that produce point estimates can be viewed as predicting the expected value of some unknown probability distribution. For example, suppose we have some system $X \to y$ that contains a degree of aleatoric uncertainty such that $P(y \mid X)$ is Gaussian. Now suppose we train a point estimate model that predicts y from X such that the mean square error between the model's predictions and the observations from the execution of this system is minimized. This is approximately equivalent to minimizing the Kullback-Leibler (KL) divergence (which measures how one probability distribution is different from a second, reference probability distribution) between a distribution with expected value given by the model's output and $P(y \mid X)$. In this way the structure of the model's aleatoric uncertainty is captured in its loss function (mean square error in this case).

Now consider a physical system which has several components, each of which has some degree of aleatoric uncertainty. Suppose we want to build a compositional model for this system. If we use the composition of Fong et al. (2019), then we can only represent the full model's uncertainty with the loss function that parameterizes the backpropagation functor. As a result, we cannot characterize the interactions between the uncertainty in the different parts of the system.

To address this, Eberhardt et al. (2016) build a convolutional neural network model to assess how the visual cortex performs a rapid stimulus categorization task. Their model

includes multiple layers which represent the hierarchy within the central nervous system from photoreceptors in the eye, to edge-detecting neurons in the primary visual cortex, to higher-order feature detectors in the later stages of visual cortex. Although there is aleatoric uncertainty at each layer of this biological system, Eberhardt et al. (2016) use a standard composition of neural network layers and therefore can only represent this uncertainty with a cross-entropy loss over the model's final output.

We describe an alternative strategy for constructing and composing parametric models such that we can explicitly characterize how different subsystems' uncertainties interact. We use this strategy to build a generalized framework for training neural networks that have stochastic processes as layers. To do this, we replace the domain **Para** of the backpropagation functor by Fong et al. (2019) with a probabilistically motivated category over which we can define the error function $er : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ through the maximum likelihood procedure. Our specific contributions are to:

- Develop a strategy for composing stochastic processes that is compatible with both subordination (Lalley, 2007) and parametric function composition (Fong et al., 2019).
- Introduce two categories with this compositional structure, one based on **Para** (Fong et al., 2019) and one based on the co-Kleisli category of the product comonad, and explore their relationships with each other and with the category **EucStoch** of Markov kernels.
- Extend the category of stochastic processes to a category of parametric statistical models.
- Define a family of subcategories of parametric statistical models over which we can use the maximum likelihood procedure to define a backpropagation functor into the category **Learn** of learning algorithms (Fong et al., 2019).

5.2 Random Variables and Independence

In any categorical presentation of probability, a natural question is how to reason about the notion of independence of random variables (Gerhold et al., 2016; Franz, 2002; Fritz, 2020).

5.2.1 Random Variables and Independence in EucStoch

We can define an embedding functor from **EucMeas** into **EucStoch** that acts as the identity on objects and sends the measurable function $f : (A, \Sigma_A) \to (B, \Sigma_B)$ to the Dirac Markov kernel $\delta_f : A \times \Sigma_B \to [0, 1]$ where for $x_a \in A, \sigma_b \in \Sigma_B$:

$$\delta_f(x_a, \sigma_b) = \begin{cases} 1 & f(x_a) \in \sigma_b \\ 0 & f(x_a) \notin \sigma_b \end{cases}$$

This formalizes the intuition that Markov kernels are a generalization of both measurable functions and probability measures, and provides an avenue to directly study random variables and their independence in **EucStoch**.

Suppose we have a probability space (Ω, Σ, μ) such that $(\Omega, \Sigma) \in Ob(\mathbf{EucMeas})$, and two real valued random variables defined on this space f, f'. We can think of these

random variables as morphisms in **EucMeas** from (Ω, Σ) to $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$. We can represent this probability space as a morphism in **EucStoch** from the monoidal unit $*$ to (Ω, Σ): that is, a Markov kernel $\mu : * \times \Sigma \to [0, 1]$. Going forward we will write the type signature $* \times \Sigma \to [0, 1]$ as $\Sigma \to [0, 1]$ for convenience.

We can then represent f and f' with their embeddings into **EucStoch**: the Dirac Markov kernels $\delta_f, \delta_{f'}$. If we compose δ_f and μ in **EucStoch**, we form a new probability measure $(\delta_f \circ \mu) : \mathcal{B}(\mathbb{R}) \to [0, 1]$, which is the pushforward measure $f_*\mu$ of μ along f.

We now have a hint of how we can reason about the independence or dependence of random variables in **EucStoch**. First, consider the probability measure:

$$(\delta_f \circ \mu) \otimes (\delta_{f'} \circ \mu) : \mathcal{B}(\mathbb{R} \times \mathbb{R}) \to [0, 1]$$

where for $\sigma \times \sigma' \in \mathcal{B}(\mathbb{R} \times \mathbb{R})$:

$$\begin{aligned}
&\left[(\delta_f \circ \mu) \otimes (\delta_{f'} \circ \mu)\right](\sigma \times \sigma') \\
=\ &\{\text{Tensor of Markov kernels}\} \\
&\left[\int_{\omega \in \Omega} \delta_f(\omega, \sigma) d\mu\right] \left[\int_{\omega \in \Omega} \delta_{f'}(\omega, \sigma') d\mu\right] \\
=\ &\{\text{Definition of } \delta\} \\
&f_*\mu(\sigma) f'_*\mu(\sigma')
\end{aligned}$$

This is simply the product measure over $(\mathbb{R}\times\mathbb{R}, \mathcal{B}(\mathbb{R}\times\mathbb{R}))$ of the probability measures $(\delta_f \circ \mu)$ and $(\delta_{f'} \circ \mu)$ over $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$. It is completely determined by the marginal distributions of f and f' over the probability space (Ω, Σ, μ), and it is agnostic to the independence or dependence structure of f and f'. The reason for this is that the measure μ is essentially "duplicated", and the random variables f and f' are not actually compared over the same probability space.

In contrast, consider instead the probability measure:

$$(\delta_f \otimes \delta_{f'}) \circ \mathsf{cp} \circ \mu : \mathcal{B}(\mathbb{R} \times \mathbb{R}) \to [0, 1]$$

where $\mathsf{cp} : \Omega \to \Omega \otimes \Omega$ is the comonoidal copy map at Ω in **EucStoch** (Fritz, 2020). We can see that for $\sigma \times \sigma' \in \mathcal{B}(\mathbb{R} \times \mathbb{R})$:

$$\left[(\delta_f \otimes \delta_{f'}) \circ \mathsf{cp} \circ \mu\right](\sigma \times \sigma') = \left[\int_{\omega \in \Omega} \delta_f(\omega, \sigma) \delta_{f'}(\omega, \sigma') d\mu\right].$$

This is the probability measure over $(\mathbb{R}\times\mathbb{R}, \mathcal{B}(\mathbb{R}\times\mathbb{R}))$ associated with the joint distribution of the random variables f and f' over (Ω, Σ, μ).

Therefore, the random variables f and f' are independent over the probability space (Ω, Σ, μ) if and only if the probability measures $(\delta_f \circ \mu) \otimes (\delta_{f'} \circ \mu)$ and $(\delta_f \otimes \delta'_f) \circ \mathsf{cp} \circ \mu$ are equal.

5.3 Probabilistic Maps

Fong et al. (2019); Gavranovic (2019); Cruttwell et al. (2021) build their characterization of machine learning optimization problems on top of the category of Euclidean spaces and parameterized infinitely differentiable maps between them. Rather than represent the loss function itself categorically, the authors treat it as an externally-provided hyperparameter.

However, in practice the loss function is usually implied by the problem. A common problem statement is as follows: given some parameterized random variable, derive the parameters that maximize the likelihood of some observed data being drawn from the distribution of this random variable. A natural question is therefore whether it is possible to replace the parameterized infinitely differentiable maps in these constructions with parameterized random variables.

A quick note on the category of Euclidean spaces and parameterized infinitely differentiable maps between them: Fong et al. (2019) calls this category **Para** whereas Cruttwell et al. (2021) call it **Para**(**Euc**) (Definition 3.6). We will use the notation by Cruttwell et al. (2021), but we will work with the category **EucMeas** (Proposition 2.33) instead of **Euc** to make it easier to talk about probabilistic constructions.

Before moving to **Para**(**EucMeas**), we will start with the category **EucMeas** (Proposition 2.33) of Euclidean spaces and infinitely differentiable maps between them. Our first step will be to replace the morphisms in **EucMeas** with stochastic processes (Definition 2.30).

5.3.1 The co-Kleisli Construction

It is often convenient to use a single source of randomness to bootstrap a stochastic system. For example, most computer simulations use random number generators with a single random seed. This can make it easier to reason about the system behavior, repeat a simulation, or encode complex dependence or independence relationships between different parts of a system. Our first construction will use this "single source of randomness" approach.

To start, note that $(O \times \_) : \mathbf{C} \to \mathbf{C}$ is an endofunctor that maps the object $A \in Ob(\mathbf{C})$ to $O \times A$ and maps the morphism $f : A \to B$ to the morphism $id_O \times f : O \times A \to O \times B$. We can now introduce the following definition:

Definition 5.1. *Given a Cartesian monoidal category* $\mathbf{C}$ *and object* O *in* $\mathbf{C}$, $\mathbf{CoKl}_O(\mathbf{C})$ *is the co-Kleisli category of* $\mathbf{C}$ *under the product comonad* $(O \times \_)$.

The category $\mathbf{CoKl}_O(\mathbf{C})$ has the same objects as $\mathbf{C}$ and the morphisms in $\mathbf{CoKl}_O(\mathbf{C})[A, B]$ are morphisms in $\mathbf{C}[O \times A, B]$. The identity map at the object A is:

$$(\mathsf{del}_O \times id_A) : O \times A \to A$$

where $\mathsf{del}_O : O \to *$ is the unique map from O to the terminal object $*$ in $\mathbf{C}$ and $(\mathsf{del}_O \times id_A)$ is the counit of the product comonad.

The $\mathbf{CoKl}_O(\mathbf{C})$ composition of the morphisms $f : O \times A \to B$ and $g : O \times B \to C$ is:

$$(g \circ_{\mathbf{CoKl}_O(\mathbf{C})} f) : O \times A \to C$$
$$g \circ_{\mathbf{CoKl}_O(\mathbf{C})} f = g \circ_{\mathbf{C}} (id_O \times f) \circ_{\mathbf{C}} (\mathsf{cp}_O \times id_A)$$

where $\mathsf{cp}_O : O \to O \times O$ is the copy map (diagonal) in $\mathbf{C}$ and $(\mathsf{cp}_O \times id_A)$ is the comultiplication of the product comonad.

For example, if Ω is $\mathbb{R}^n$ for some $n \in \mathbb{N}$, the category $\mathbf{CoKl}_{(\Omega, \mathcal{B}(\Omega))}(\mathbf{EucMeas})$ (which we will hereafter abbreviate **CEM**, see Table 3) has the same objects as **EucMeas**, and the morphisms from $\mathbb{R}^a$ to $\mathbb{R}^b$ are continuously differentiable (and therefore Borel measurable) functions of the form $f : \Omega \times \mathbb{R}^a \to \mathbb{R}^b$.

In **CEM**, the identity arrow at $\mathbb{R}^a$ is the function $f(\omega, x_a) = x_a$ and the composition of $f : \Omega \times \mathbb{R}^a \to \mathbb{R}^b$ and $f' : \Omega \times \mathbb{R}^b \to \mathbb{R}^c$ is $(f' \circ f) : \Omega \times \mathbb{R}^a \to \mathbb{R}^c$ where for $\omega \in \Omega, x_a \in \mathbb{R}^a$:

$$(f' \circ f)(\omega, x_a) = f'(\omega, f(\omega, x_a)).$$

One important thing to note is that ω is reused when we compose f and f'. This allows us to make the following claim:

Proposition 5.2. *For any $\omega \in \Omega$, the identity-on-objects map that sends the function $f : \Omega \times \mathbb{R}^a \to \mathbb{R}^b$ in* **CEM** *to the function $f(\omega, \_) : \mathbb{R}^a \to \mathbb{R}^b$ in* **EucMeas** *is a functor $R_\omega : \mathbf{CEM} \to \mathbf{EucMeas}$, which we call the realization functor.*

Proof. First, if f is the identity map in **CEM** then $f(\omega, \_)$ is by definition the identity function. Next, consider $f : \Omega \times \mathbb{R}^a \to \mathbb{R}^b, f' : \Omega \times \mathbb{R}^b \to \mathbb{R}^c$ in **CEM** and any $x_a \in \mathbb{R}^a$. Then:

$$(R_\omega f' \circ R_\omega f) : \mathbb{R}^a \to \mathbb{R}^c$$
$$(R_\omega f' \circ R_\omega f)(x_a) = (f'(\omega, \_) \circ f(\omega, \_))(x_a) = f'(\omega, f(\omega, x_a)) = R_\omega(f' \circ f)(x_a)$$

so composition is preserved. □

Given a probability measure $\mu : \mathcal{B}(\Omega) \to [0, 1]$, we can think of **CEM** as a category of differentiable stochastic processes defined on the probability space $(\Omega, \mathcal{B}(\Omega), \mu)$.

5.3.1.1 Independence and Dependence in CEM

Since all of the stochastic processes in **CEM** are defined over the same probability space $(\Omega, \mathcal{B}(\Omega), \mu)$, there is a major difference between how **CEM** and **EucStoch** represent independence and dependence. Given the arrows $f : \Omega \times \mathbb{R}^a \to \mathbb{R}^b$ and $f' : \Omega \times \mathbb{R}^c \to \mathbb{R}^d$ in **CEM** and the vectors $x_a \in \mathbb{R}^a, x_c \in \mathbb{R}^c$, the random variables $f(\_, x_a)$ and $f'(\_, x_c)$ may be either dependent or independent.

In order to see how this differs from the situation in **EucStoch**, recall that the pushforward of μ along the stochastic process $f : \Omega \times \mathbb{R}^a \to \mathbb{R}^b$ is a mapping from **CEM** to **EucStoch** such that for $x_a \in \mathbb{R}^a, \sigma_b \in \mathcal{B}(\mathbb{R}^b)$:

$$f_*\mu : \mathbb{R}^a \times \mathcal{B}(\mathbb{R}^b) \to [0, 1]$$
$$f_*\mu(x_a, \sigma_b) = f(\_, x_a)_*\mu(\sigma_b) = \mu(f(\_, x_a)^{-1}(\sigma_b)) = \int_{\omega \in \Omega} \delta(f(\omega, x_a), \sigma_b) d\mu.$$

However, this mapping does not form a functor. We see that for $f : \Omega \times \mathbb{R}^a \to \mathbb{R}^b$,

$f' : \Omega \times \mathbb{R}^b \to \mathbb{R}^c$, $x_a \in \mathbb{R}^a, \sigma_c \in \mathcal{B}(\mathbb{R}^c)$ the pushforward of the **CEM** composition is:

$$
\begin{aligned}
& (f' \circ f)_* \mu : \mathbb{R}^a \times \mathcal{B}(\mathbb{R}^c) \to [0,1] \\
& (f' \circ f)_* \mu (x_a, \sigma_c) \\
= \quad & \{\text{Definition of pushforward}\} \\
& \mu((f' \circ f)(\_, x_a)^{-1}(\sigma_c)) \\
= \quad & \{\text{Composition in } \mathbf{CEM}\} \\
& \int_{\omega \in \Omega} \delta((f'(\omega, f(\omega, x_a)), \sigma_c) d\mu \\
= \quad & \{\text{Rewrite in terms of } \delta\} \\
& \int_{\omega \in \Omega} \left(\int_{x_b \in \mathbb{R}^b} \delta(f'(\omega, x_b), \sigma_c) d\delta(f(\omega, x_a), \_) \right) d\mu \\
= \quad & \{\text{Rearrange integrals}\} \\
& \int_{x_b \in \mathbb{R}^b} \int_{\omega \in \Omega} \delta(f'(\omega, x_b), \sigma_c) \; d\delta(f(\omega, x_a), \_) \; d\mu
\end{aligned}
$$

whereas when we take the **Stoch** composition of the pushforwards we have:

$$
\begin{aligned}
& [f'_* \mu \circ f_* \mu] : \mathbb{R}^a \times \mathcal{B}(\mathbb{R}^c) \to [0,1] \\
& [f'_* \mu \circ f_* \mu] (x_a, \sigma_c) \\
= \quad & \{\text{Composition in } \mathbf{Stoch}\} \\
& \int_{x_b \in \mathbb{R}^b} \mu(f'(\_, x_b)^{-1}(\sigma_c)) \; d\mu(f(\_, x_a)^{-1}(\_)) \\
= \quad & \{\text{Rewrite in terms of } \delta\} \\
& \int_{x_b \in \mathbb{R}^b} \left(\int_{\omega \in \Omega} \delta(f'(\omega, x_b), \sigma_c) d\mu \right) \left(\int_{\omega \in \Omega} d\delta(f(\omega, x_a), \_) d\mu \right).
\end{aligned}
$$

These are not necessarily equivalent if the random variables $f'(\_, x_b), x_b \in \mathbb{R}^b$ are not independent of the random variable $f(\_, x_a)$.

The reason for this mismatch comes down to the fact that composition in **EucStoch** is based on the Markov property, whereas composition in **CEM** is not. In the next section we will define a new category of stochastic processes that exhibits this independence behavior.

Shorthand Name	Full Name
CEM	$\mathbf{CoKl}_{(\Omega, \mathcal{B}(\Omega))}(\mathbf{EucMeas})$
PEM	$\mathbf{Para}_{(\Omega, \mathcal{B}(\Omega))^*}(\mathbf{EucMeas})$

Table 3: *We introduce several compositional constructions for building new categories in this section. These can produce unwieldy names, so for readability we have abbreviated some of them here.*

5.3.2 The Parameterization Construction

In order to reason about the behavior of a system of stochastic processes, it is useful to study them in a simpler setting. There are two simple ways to do this: take pushforwards and study stochastic processes as Markov kernels, or take expectations and study stochastic processes as functions. In order to make these lines of study rigorous, we first need to establish the functoriality of these transformations. To this end, over the next few sections we build a new category of stochastic processes in which the map $f \to f_*\mu$ is functorial. In Section 5.4.3 we will explore the functoriality of the expectation.

In order to elevate the pushforward to a functor, we need to modify the definition of how stochastic processes compose. Unlike in **CEM**, where we treat all stochastic processes as if they were defined over the same probability space, the category in this section will consist of stochastic processes defined over different, non-interacting probability spaces. The composition of two stochastic processes in this new category will produce a stochastic process over the product of those processes' associated probability spaces. This will allow us to treat all of the stochastic processes in this category as if they were mutually independent.

We note that this strategy of expanding the probability space each time we introduce a new source of randomness is commonly used by probability theorists (Tao, 2010; Billingsley, 1986; Ash and Gardner, 1975).

5.3.2.1 A Subcategory of Para(C)

Proposition 5.3. *For the small strict symmetric monoidal categories* $\mathbf{C}$ *and* $\mathbf{D}$ *equipped with an identity-on-morphisms (faithful) identity-on-objects strict monoidal functor* $\iota : \mathbf{D} \hookrightarrow \mathbf{C}$ *we can form a subcategory* $\mathbf{Para_D(C)}$ *of* $\mathbf{Para(C)}$ *(Definition 3.6) in which the morphisms in* $\mathbf{Para_D(C)}[A, B]$ *are pairs* (P, f) *where* P *is an object in the image of* ι *and* $f : P \otimes A \to B$ *is a morphism in* $\mathbf{C}$.

Proof. In order to show that $\mathbf{Para_D(C)}$ is a subcategory of $\mathbf{Para(C)}$ we simply need to show that $\mathbf{Para_D(C)}$ is closed under composition and contains all identities.

To start, note that $\mathbf{Para_D(C)}$ contains all identities. Since $\mathbf{D}$ is a strict monoidal category, we can write the signature of the identity arrow id_A in $\mathbf{D}$ as:

$$id_A : * \otimes A \to A$$

where $*$ is the monoidal unit. Therefore id_A is an arrow in $\mathbf{Para_D(C)}$. This arrow is trivially the $\mathbf{Para_D(C)}$-identity at A.

Next, note that $\mathbf{Para_D(C)}$ is closed under composition. Consider the morphisms:

$$f_1 : P_1 \otimes A \to B \qquad f_2 : P_2 \otimes B \to C$$

where P_1, P_2 are objects in the image of ι. Since ι is identity-on-objects and strict monoidal, it must be that $P_2 \otimes P_1$ is an object in $\mathbf{D}$. Therefore:

$$f_2 \circ_{\mathbf{Para(C)}} f_1 : P_2 \otimes P_1 \otimes A \to C$$

is a morphism in $\mathbf{Para_D(C)}$. □

We briefly note that although we could likely relax the identity-on-morphisms requirement for ι, we keep this requirement to make some of the proofs and definitions in the following sections more straightforward.

5.3.2.2 A Category of Parametric Measurable Maps

In this section, we will use the $\mathbf{Para}_{\mathbf{D}}$ construction (Proposition 5.3) to build a new category of stochastic processes over which the mapping $f \to f_*\mu$ is functorial. In this category composition will have the same independence structure that it has in **EucStoch**. We begin with the following definition:

Definition 5.4. *Suppose* $\mathbf{C}$ *is a strict Cartesian monoidal category,* O^* *is a Lawvere theory with generating object* O*, and* ι *is an identity-on-morphisms (faithful) identity-on-objects strict monoidal functor* $\iota : O^* \hookrightarrow \mathbf{C}$*. Then* $\mathbf{Para}_{O^*}(\mathbf{C})$ *is a Lawvere parameterization of* $\mathbf{C}$*.*

The objects in O^* are of the form $O \times O \times \cdots \times O$. When the tensor is repeated n times we will write this as O^n. We also write O^0 for the monoidal unit $*$. For any strict Cartesian monoidal category $\mathbf{C}$ with a Lawvere parameterization we can define a mapping:

$$Copy : \mathbf{Para}_{O^*}(\mathbf{C}) \to \mathbf{CoKl}_O(\mathbf{C})$$

This mapping acts as identity-on-objects and sends the arrow $f : O^n \times A \to B$ in $\mathbf{Para}_{O^*}(\mathbf{C})$ to the following arrow in $\mathbf{CoKl}_O(\mathbf{C})$:

$$f \circ_{\mathbf{C}} (\mathsf{cp}_O(n) \otimes id_A^{\mathbf{C}}) : O \times A \to B.$$

where $id_A^{\mathbf{C}}$ is the identity arrow on A in $\mathbf{C}$ and $\mathsf{cp}_O(n)$ is the $n-1$ repeated application of the copy map $O \to O \times O$ in $\mathbf{C}$. That is:

- $\mathsf{cp}_O(3) : O \to O \times O \times O$ is the double application of the copy map $(id_O \otimes \mathsf{cp}_O) \circ \mathsf{cp}_O$.
- $\mathsf{cp}_O(2) : O \to O \times O$ is just the copy map cp_O.
- $\mathsf{cp}_O(1) : O \to O$ is the identity map id_O in $\mathbf{C}$.
- $\mathsf{cp}_O(0) : O \to *$ is del_O, the unique map from O to the terminal object $*$.

,

Proposition 5.5. *Suppose* $\mathbf{C}$ *is a strict Cartesian monoidal category and* O *is an object in* $\mathbf{C}$*. Then* $Copy : \mathbf{Para}_{O^*}(\mathbf{C}) \to \mathbf{CoKl}_O(\mathbf{C})$ *is a full identity-on-objects functor.*

Proof. First, we note that *Copy* is identity-on-objects by definition.

Next, consider any objects A, B in $\mathbf{C}$ and any arrow $f : O \times A \to B$ in $\mathbf{CoKl}_O(\mathbf{C})$ (Definition 5.1). Then f is also an arrow in $\mathbf{Para}_{O^*}(\mathbf{C})$ and *Copy* maps f to:

$$f \circ_{\mathbf{C}} (\mathsf{cp}_O(1) \otimes id_A^{\mathbf{C}}) = f.$$

Therefore *Copy* is full.

Next, since $\mathbf{C}$ is strict monoidal we have

$$A = * \times A = O^0 \times A$$

which implies that *Copy* maps the $id_A : A \to A$ arrow in $\mathbf{Para}_{O^*}(\mathbf{C})$ to the arrow:

$$id_A \circ_{\mathbf{C}} (\mathsf{cp}_O(0) \otimes id_A^{\mathbf{C}}) : O \times A \to A$$

which is the identity arrow in $\mathbf{CoKl}_O(\mathbf{C})$. Therefore, *Copy* preserves identity morphisms.

Next, we will show *Copy* preserves composition. Suppose $f : O^m \times A \to B$ and $f' : O^n \times B \to C$ are arrows in $\mathbf{Para}_{O^*}(\mathbf{C})$:

$$
\begin{aligned}
&(Copyf' \circ Copyf) : O \otimes A \to C \\
&(Copyf' \circ_{\mathbf{CoKl}_O(\mathbf{C})} Copyf) \\
=\quad &\{\text{Apply } Copy\} \\
&(f' \circ_{\mathbf{C}} (\mathsf{cp}_O(n) \otimes id_B^{\mathbf{C}})) \circ_{\mathbf{CoKl}_O(\mathbf{C})} (f \circ_{\mathbf{C}} (\mathsf{cp}_O(m) \otimes id_A^{\mathbf{C}})) \\
=\quad &\{\text{Compute composition in } \mathbf{CoKl}_O(\mathbf{C})\} \\
&(f' \circ_{\mathbf{C}} (\mathsf{cp}_O(n) \otimes id_B^{\mathbf{C}})) \circ_{\mathbf{C}} (id_O^{\mathbf{C}} \otimes (f \circ_{\mathbf{C}} (\mathsf{cp}_O(m) \otimes id_A^{\mathbf{C}}))) \circ_{\mathbf{C}} (\mathsf{cp}_O \otimes id_A^{\mathbf{C}}) \\
=\quad &\{\text{Apply identities}\} \\
&f' \circ_{\mathbf{C}} (\mathsf{cp}_O(n) \otimes (f \circ_{\mathbf{C}} (\mathsf{cp}_O(m) \otimes id_A^{\mathbf{C}}))) \circ_{\mathbf{C}} (\mathsf{cp}_O \otimes id_A^{\mathbf{C}}) \\
=\quad &\{\text{Rewrite application of } \mathsf{cp}_O(n)\} \\
&f' \circ_{\mathbf{C}} (id_{O^n} \otimes (f \circ_{\mathbf{C}} (\mathsf{cp}_O(m) \otimes id_A^{\mathbf{C}}))) \circ_{\mathbf{C}} (\mathsf{cp}_O(n+1) \otimes id_A^{\mathbf{C}}) \\
=\quad &\{\text{Rewrite application of } \mathsf{cp}_O(m)\} \\
&f' \circ_{\mathbf{C}} (id_{O^n} \otimes f) \circ_{\mathbf{C}} (\mathsf{cp}_O(n+m) \otimes id_A^{\mathbf{C}}) \\
=\quad &\{\text{Definition of composition in } \mathbf{Para}_{O^*}(\mathbf{C})\} \\
&(f' \circ_{\mathbf{Para}_{O^*}(\mathbf{C})} f) \circ_{\mathbf{C}} (\mathsf{cp}_O(n+m) \otimes id_A^{\mathbf{C}}) \\
=\quad &\{\text{Definition of } Copy\} \\
&Copy(f' \circ_{\mathbf{Para}_{O^*}(\mathbf{C})} f)
\end{aligned}
$$

□

Proposition 5.5 links the parameterization and co-Kleisli constructions. We will demonstrate the significance of this linkage below.

Now suppose we have a probability space $(\Omega, \mathcal{B}(\Omega), \mu)$ where Ω is $\mathbb{R}^k, k \in \mathbb{N}$. We can form the Lawvere theory $(\Omega, \mathcal{B}(\Omega))^*$ with generating object $(\Omega, \mathcal{B}(\Omega))$ and tuples:

$$(\Omega, \mathcal{B}(\Omega))^n = (\Omega^n, \mathcal{B}(\Omega^n))$$

as objects. We can also form the identity-on-morphisms (faithful) identity-on-objects strict monoidal functor as the inclusion:

$$\iota : (\Omega, \mathcal{B}(\Omega))^* \hookrightarrow \mathbf{EucMeas}$$

Then for any:

$$(\Omega^n, \mathcal{B}(\Omega^n)) \in (\Omega, \mathcal{B}(\Omega))^*$$

we can create the probability space $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$ where μ^n is the product measure:

$$
\begin{aligned}
&\mu^n : \mathcal{B}(\Omega^n) \to [0,1] \\
&\mu^n(\sigma_1 \times \sigma_2 \times \cdots \times \sigma_n) = \mu(\sigma_1)\mu(\sigma_2)\cdots\mu(\sigma_n).
\end{aligned}
$$

Definition 5.6. *We can apply* $\mathbf{Para}_{(\Omega,\mathcal{B}(\Omega))^*}$ *to* **EucMeas** *to form the Lawvere parameterization* $\mathbf{Para}_{(\Omega,\mathcal{B}(\Omega))^*}(\mathbf{EucMeas})$*, which we will hereafter abbreviate* **PEM**.

Intuitively, **PEM** allows us to reason about probabilistic relationships in terms of measurable functions rather than probability measures.

Next, by Proposition 5.5, we have an identity-on-objects functor, *Copy*, from **PEM** to **CEM**. Let's drill deeper into this relationship. We can view an arrow of the form $f : \Omega^n \times \mathbb{R}^a \to \mathbb{R}^b$ in **PEM** as a stochastic process over $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$. However, unlike in **CEM**, if we compose f with another arrow in **PEM**, we do not get another stochastic process over $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$. Instead, we get a stochastic process over some other probability space. Intuitively, we can think of the stochastic processes in **PEM** as being defined over different, non-interacting probability spaces.

Furthermore *Copy* enables us to mirror the structures of the co-Kleisli construction within the parameterization construction. For example, we can apply $R_\omega \circ Copy$ to **PEM** to propagate a single choice of ω throughout the entire category.

Now given some arrow $f : \Omega^n \times \mathbb{R}^a \to \mathbb{R}$ in **PEM** and $x_a \in \mathbb{R}^a$, the measurable function $f(\_, x_a)$ is a real valued random variable over the probability space $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$. The pushforward of μ^n along this random variable $f(\_, x_a)_* \mu^n(\_) : \mathcal{B}(\mathbb{R}) \to [0, 1]$ is then a probability measure over the space $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$.

In general, we can extend this pushforward procedure to define a mapping between parametric families of measurable maps and Markov kernels. Given some $f : \Omega^n \times \mathbb{R}^a \to \mathbb{R}^b$ we can define:

$$Push_\mu f : \mathbb{R}^a \times \mathcal{B}(\mathbb{R}^b) \to [0, 1]$$

where for $x_a \in \mathbb{R}^a, \sigma_b \in \mathcal{B}(\mathbb{R}^b)$:

$$Push_\mu f(x_a, \sigma_b) = f(\_, x_a)_* \mu^n(\sigma_b) = \int_{\omega_n \in \Omega^n} \delta(f(\omega_n, x_a), \sigma_b) d\mu^n.$$

Theorem 5.7. *The mapping $Push_\mu$ that takes a parametric family $f : \Omega^n \times \mathbb{R}^a \to \mathbb{R}^b$ of measurable maps to the Markov kernel $f_* \mu^n$ is an identity-on-objects functor from* **PEM** *to* **EucStoch**.

Proof. We first note that since $Push_\mu$ acts as the identity on objects and:

$$Ob(\mathbf{PEM}) = Ob(\mathbf{EucMeas}) = Ob(\mathbf{EucStoch})$$

it must be that $Push_\mu$ maps objects in **PEM** to objects in **EucStoch**.

Next, note that for any $\mathbb{R}^a$, $Push_\mu$ maps the identity at $\mathbb{R}^a$ in **PEM** to the identity at $\mathbb{R}^a$ in **EucStoch** since:

$$\begin{aligned}
& Push_\mu id_{\mathbb{R}^a}(x_a, \sigma_a) \\
= \quad & \{\text{Definition of } Push_\mu\} \\
& \int_{\omega_n \in \Omega^n} \delta(id_{\mathbb{R}^a}(\omega_n, x_a), \sigma_a) d\mu^n \\
= \quad & \{\text{Apply identity}\} \\
& \int_{\omega_n \in \Omega^n} \delta(x_a, \sigma_a) d\mu^n \\
= \quad & \{\text{Compute integral}\} \\
& \delta(x_a, \sigma_a)
\end{aligned}$$

Next, we will demonstrate that $Push_\mu$ preserves composition. Suppose we have some:

$$f : \Omega^n \times \mathbb{R}^a \to \mathbb{R}^b \qquad f' : \Omega^m \times \mathbb{R}^b \to \mathbb{R}^c$$
$$x_a \in \mathbb{R}^a \qquad \sigma_c \in \mathcal{B}(\mathbb{R}^c)$$

Then we can write:

$$\begin{aligned}
& Push_\mu\,(f' \circ f) : \mathbb{R}^a \times \mathcal{B}(\mathbb{R}^c) \to [0,1] \\
& Push_\mu\,(f' \circ f)\,(x_a, \sigma_c) \\
= \quad & \{\text{Definition of } Push_\mu\} \\
& \int_{(\omega_m,\omega_n)\in\Omega^m\times\Omega^n} \delta((f' \circ f)((\omega_m,\omega_n),x_a),\sigma_c) d\mu^{n+m} \\
= \quad & \{\text{Apply } \textbf{PEM} \text{ composition}\} \\
& \int_{\omega_m\in\Omega^m} \int_{\omega_n\in\Omega^n} \delta(f'(\omega_m, f(\omega_n, x_a)),\sigma_c) d\mu^n d\mu^m \\
= \quad & \{\text{Rewrite with } \delta\} \\
& \int_{\omega_m\in\Omega^m} \int_{\omega_n\in\Omega^n} \int_{x_b\in\mathbb{R}^b} \delta(f'(\omega_m, x_b),\sigma_c) d\delta(f(\omega_n, x_a), _) d\mu^n d\mu^m \\
= \quad & \{\text{Rearrange integrals}\} \\
& \int_{x_b\in\mathbb{R}^b} \left[\int_{\omega_m\in\Omega^m} \delta(f'(\omega_m, x_b),\sigma_c) d\mu^m\right] d \left[\int_{\omega_n\in\Omega^n} \delta(f(\omega_n, x_a), _) d\mu^n\right] \\
= \quad & \{\text{Definition of } Push_\mu\} \\
& \int_{x_b\in\mathbb{R}^b} [Push_\mu f'](x_b,\sigma_c)\; d[Push_\mu f](x_a, _) \\
= \quad & \{\text{Definition of composition in } \textbf{Stoch}\} \\
& (Push_\mu f' \circ Push_\mu f)(x_a, \sigma_c)
\end{aligned}$$

□

5.3.3 Composition Experiments

We can express the difference between composition in **PEM** and **CEM** with a simple experiment using the numpy (Harris et al., 2020) and scipy (Virtanen et al., 2020) libraries.

Consider the probability space $(\Omega, \mathcal{B}(\Omega), \mu)$ where $\Omega = [0,1]$ and μ is the uniform measure. We can represent samples from this with the following:

```
import numpy as np
omega_samples = np.random.random(10000)
```

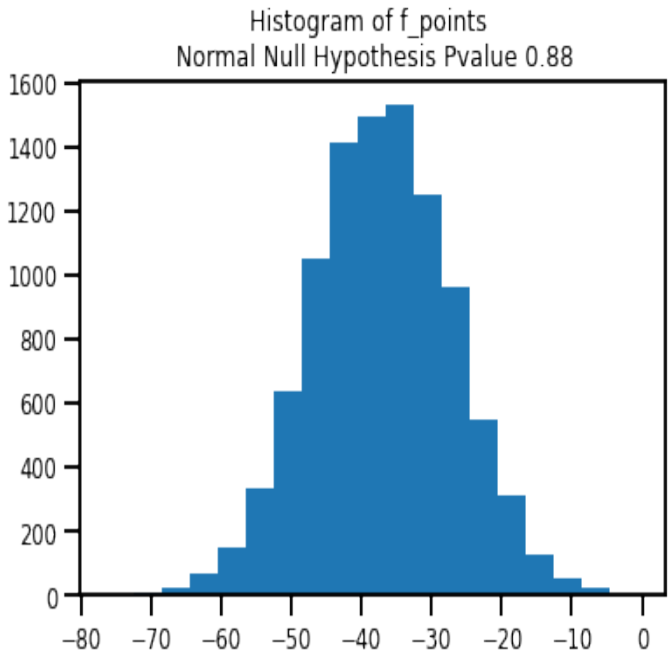
Now consider the following stochastic process $f : \Omega \times \mathbb{R} \to \mathbb{R}$ over $(\Omega, \mathcal{B}(\Omega), \mu)$:

```
from scipy.stats import norm
def f(omega, x):
```

```
    normal_points = norm(loc=5 - x, scale=10).ppf(omega)
    return normal_points
```

For any $x \in \mathbb{R}$, the random variable $f(\_, x)$ has a normal distribution:

```
input_x = 42
f_points = f(omega_samples, input_x)
```

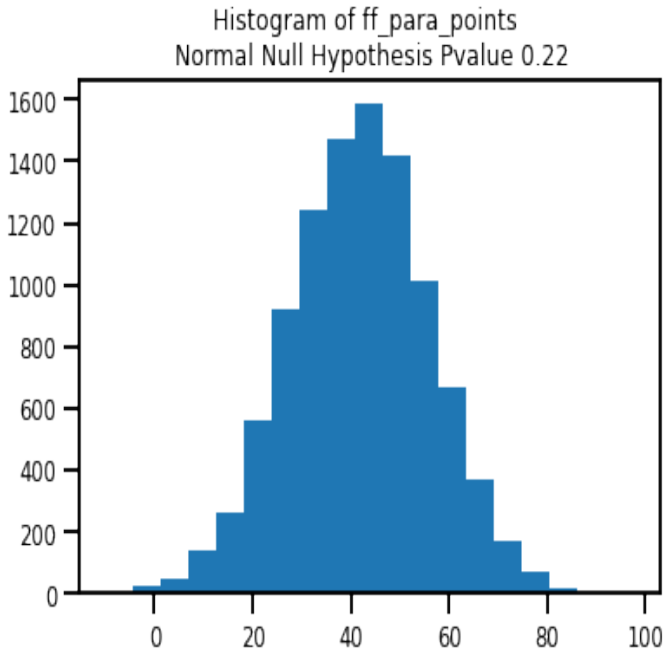


f is an endomorphism on $\mathbb{R}$ in **PEM**, so we can take the **PEM** composition of f with itself to form the arrow $(f \circ f) : \Omega^2 \times \mathbb{R} \to \mathbb{R}$. We write this arrow as:

```
def ff_para(omega1, omega2, x):
    return f(omega2, f(omega1, x))
```

Note that $(f \circ f)$ is a stochastic process over the product probability space $(\Omega^2, \mathcal{B}(\Omega^2), \mu^2)$, and that for any $x \in \mathbb{R}$, the random variable $(f \circ f)(\_, \_, x)$ over $(\Omega^2, \mathcal{B}(\Omega^2), \mu^2)$ has a normal distribution as well:

```
input_x = 42
omega2 = np.random.random((10000, 2))
ff_para_points = ff_para(omega2[:, 0], omega2[:, 1], input_x)
```



Now recall the functor $Copy : \mathbf{PEM} \to \mathbf{CEM}$ from Proposition 5.5. This functor acts as identity-on-objects and sends the arrow $f : \Omega^n \times \mathbb{R} \to \mathbb{R}$ in **PEM** to the following arrow in **CEM**:

$$f \circ_{\mathbf{EucMeas}} (\mathsf{cp}_\Omega(n) \otimes id_{\mathbb{R}}) : \Omega \times \mathbb{R} \to \mathbb{R}.$$

We can implement this functor as follows:

```
import inspect
from functools import partial
def CopyFunctor(f):
    def g(omega, x, f=f):
        while len(inspect.getargspec(f).args) > 1:
            f = partial(f, omega)
        return f(x)
    return g
```

Note that:

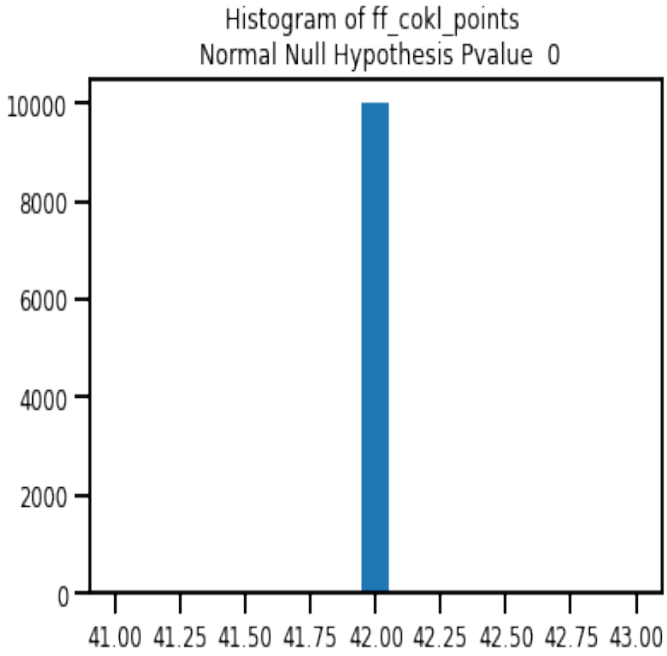
$$Copy(f \circ f) : \Omega \times \mathbb{R} \to \mathbb{R}$$

is a stochastic process over the probability space $(\Omega, \mathcal{B}(\Omega), \mu)$. However, unlike $(f \circ f)(\_, \_, x)$, the random variable:

$$Copy(f \circ f)(\_, x) : \Omega \to \mathbb{R}$$

is not normal for any fixed $x \in \mathbb{R}$. Instead, it is constant:

```
input_x = 42
omega = np.random.random(10000)
ff_cokl_points = CopyFunctor(ff_para)(omega, input_x)
```



5.4 Parameterized Statistical Models

We have been discussing the arrows in **PEM** as parameterized random variables, or stochastic processes, but we can also think of them as **EucMeas** arrows with an element of randomness that is dictated by the probability measure μ. One of the primary goals of this work is to replace the domain of the backpropagation functor by Fong et al. (2019) with a probabilistically motivated category over which we can define the error function $er : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ through maximum likelihood. Therefore, a natural next step is to extend **PEM** to a category in which we can instead think of the arrows as **Para**(**EucMeas**) arrows with an element of randomness added.

In order to do this, we will replace the stochastic processes in **PEM** with parameterized stochastic processes, which we will also refer to as parametric statistical models. That is, the arrows in this category will consist of families of random variables that have two layers of parameterization: one layer acts as the model input (e.g. the independent variable in a linear regression model) and one layer acts as the model parameters (e.g. the slope, intercept and variance terms).

5.4.1 The Category DF

Given a probability space $(\Omega, \mathcal{B}(\Omega), \mu)$ where $\Omega = \mathbb{R}^k, k \in \mathbb{N}$, any stochastic process $f : \Omega^n \times \mathbb{R}^a \to \mathbb{R}^b$ in **PEM** defines a stochastic relationship between values in $\mathbb{R}^a$ and $\mathbb{R}^b$. A parametric statistical model is a parameterized family of such relationships. For example, consider a univariate linear regression model $l : \Omega^n \times \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ where for

$\omega_n \in \Omega^n, [a, b, s] \in \mathbb{R}^3, x \in \mathbb{R}$:

$$l(\omega_n, [a, b, s], x) = ax + b + f_{\mathcal{N}(0,s^2)}(\omega_n)$$

and $f_{\mathcal{N}(0,s^2)}$ is a normally distributed random variable with mean 0 and variance s^2. Any value $[a, b, s] \in \mathbb{R}^3$ defines a stochastic process, or **PEM** arrow:

$$l(\_, [a, b, s], \_) : \Omega^n \times \mathbb{R} \to \mathbb{R}.$$

For any model input value $x \in \mathbb{R}$, the function $l(\_, [a, b, s], x)$ is then a random variable defined on the probability space $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$. Like with any ordinary univariate linear regression model, this random variable is normally distributed on the real line.

We can define a category of such models.

Proposition 5.8. *Suppose we have a probability space $(\Omega, \mathcal{B}(\Omega), \mu)$ where Ω is $\mathbb{R}^k, k \in \mathbb{N}$. We can define a category* **DF** *that has the same objects as* **EucMeas** *(Definition 2.33) such that the morphisms from $\mathbb{R}^a$ to $\mathbb{R}^b$ are* **EucMeas***-morphisms of the form:*

$$f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$

The composition of the morphisms:

$$f_1 : \Omega^{n_1} \times \mathbb{R}^{p_1} \times \mathbb{R}^a \to \mathbb{R}^b \qquad f_2 : \Omega^{n_2} \times \mathbb{R}^{p_2} \times \mathbb{R}^b \to \mathbb{R}^c$$

is the morphism:

$$f_2 \circ f_1 : \Omega^{n_2+n_1} \times \mathbb{R}^{p_2+p_1} \times \mathbb{R}^a \to \mathbb{R}^c$$
$$(f_2 \circ f_1)(\omega_{n_2}, \omega_{n_1}, x_{p_2}, x_{p_1}, x_a) = f_2(\omega_{n_2}, x_{p_2}, f_1(\omega_{n_1}, x_{p_1}, x_a))$$

Proof. We need to show that **DF** contains all identities, is closed under composition, and that composition is associative.

To start, note that the identity arrow at the object $\mathbb{R}^a$ in **DF** is the function:

$$id : \Omega^0 \times \mathbb{R}^0 \times \mathbb{R}^a \to \mathbb{R}^a$$
$$id(x_a) = x_a$$

and therefore **DF** contains all identities.

Next, note that the composition of the morphisms:

$$f_1 : \Omega^{n_1} \times \mathbb{R}^{p_1} \times \mathbb{R}^a \to \mathbb{R}^b \qquad f_2 : \Omega^{n_2} \times \mathbb{R}^{p_2} \times \mathbb{R}^b \to \mathbb{R}^c$$
$$f_2 \circ f_1 : \Omega^{n_2+n_1} \times \mathbb{R}^{p_2+p_1} \times \mathbb{R}^a \to \mathbb{R}^c$$

is in $\mathbf{DF}[\mathbb{R}^a, \mathbb{R}^c]$ and therefore **DF** is closed under composition.

Next, consider the morphisms

$$f_1 : \Omega^{n_1} \times \mathbb{R}^{p_1} \times \mathbb{R}^a \to \mathbb{R}^b \qquad f_2 : \Omega^{n_2} \times \mathbb{R}^{p_2} \times \mathbb{R}^b \to \mathbb{R}^c \qquad f_3 : \Omega^{n_3} \times \mathbb{R}^{p_3} \times \mathbb{R}^c \to \mathbb{R}^d$$

We have that:

$$f_3 \circ (f_2 \circ f_1) : \Omega^{n_3+(n_2+n_1)} \times \mathbb{R}^{p_3+(p_2+p_1)} \times \mathbb{R}^a \to \mathbb{R}^d$$

$$
\begin{aligned}
&(f_3 \circ (f_2 \circ f_1))((\omega_{n_3}, (\omega_{n_2}, \omega_{n_1})), (x_{p_3}, (x_{p_2}, x_{p_1})), x_a) = \\
&f_3(\omega_{n_3}, x_{p_3}, (f_2 \circ f_1)((\omega_{n_2}, \omega_{n_1}), (x_{p_2}, x_{p_1}), x_a)) = \\
&f_3(\omega_{n_3}, x_{p_3}, f_2(\omega_{n_2}, x_{p_2}, f_1(\omega_{n_1}, x_{p_1}, x_a)))
\end{aligned}
$$

which is equal to:

$$
(f_3 \circ f_2) \circ f_1 : \Omega^{(n_3+n_2)+n_1} \times \mathbb{R}^{(p_3+p_2)+p_1} \times \mathbb{R}^a \to \mathbb{R}^d
$$

$$
\begin{aligned}
&((f_3 \circ f_2) \circ f_1)(((\omega_{n_3}, \omega_{n_2}), \omega_{n_1}), ((x_{p_3}, x_{p_2}), x_{p_1}), x_a) = \\
&(f_3 \circ f_2)((\omega_{n_3}, \omega_{n_2}), (x_{p_3}, x_{p_2}), f_1(\omega_{n_1}, x_{p_1}, x_a)) = \\
&f_3(\omega_{n_3}, x_{p_3}, f_2(\omega_{n_2}, x_{p_2}, f_1(\omega_{n_1}, x_{p_1}, x_a)))
\end{aligned}
$$

and therefore composition in **DF** is associative. □

We can intuitively think of the category **DF** as the application of the **Para** construction to **PEM**. In fact, an earlier version of Shiebler (2021c) used exactly this strategy to construct **DF**. However, since **PEM** is not a monoidal category the category **Para**(**PEM**) is not unambiguously defined.

The name **DF** derives from the fact that the arrows in this category are **D**iscriminative and **F**requentist statistical models (see Table 3 for a list of all such abbreviations). That is, each arrow operates as if both the parameters and input values are fixed and only the output value is probabilistic. For example, the homset $\mathbf{DF}[\mathbb{R}, \mathbb{R}]$ includes the linear regression model above. In contrast, generative models and Bayesian models assume a probability distribution over the input and parameter values respectively.

5.4.2 Gaussian Preserving Transformations

Definition 5.9. *A Gaussian preserving transformation $T : \mathbb{R}^a \to \mathbb{R}^b$ is a Borel measurable function such that for any multivariate normal random variable $f : \Omega^n \to \mathbb{R}^a$ defined on the probability space $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$, the random variable $(T \circ f) : \Omega^n \to \mathbb{R}^b$ is multivariate normal and we have:*

$$
\int_{\omega_n \in \Omega^n} T(f(\omega_n)) d\mu = T\left(\int_{\omega_n \in \Omega^n} f(\omega_n) d\mu\right)
$$

For example, any linear function is Gaussian preserving. We will particularly focus on Gaussian preserving transformations for three reasons:

1. Gaussian distributions are common in practice, not least due to the central limit theorem.

2. As we will see below, Gaussian preserving transformations are closed under composition. This will enable us to form a category from Gaussian preserving transformations.

3. Theorem 5.21 will rely on the possibility of our category of study being characterizable by a single marginal error function. Using Gaussian preserving transformations as our primitive makes it easier to see how we could extend this result to other classes of distributions in the future.

5.4.2.1 A Subcategory of Gaussian Preserving Transformations

For some probability space $(\Omega, \mathcal{B}(\Omega), \mu)$ where $\Omega = \mathbb{R}^k, k \in \mathbb{N}$, we can construct a set of **DF**-arrows $\mathcal{N}_\mu$ such that for any $f \in \mathcal{N}_\mu$ with the signature:

$$f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$

there exists some map $T : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ and multivariate normal random variable $G : \Omega^n \to \mathbb{R}^b$ defined on the probability space $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$ such that for all $\omega_n \in \Omega^n, x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$ the map $T(x_p, \_) : \mathbb{R}^a \to \mathbb{R}^b$ is a Gaussian preserving transformation and:

$$f(\omega_n, x_p, x_a) = T(x_p, x_a) + G(\omega_n)$$

This includes the univariate linear regression model l, as well as the identity arrow, since constant distributions are multivariate normal with variance 0.

Since $\mathcal{N}_\mu$ contains the identity arrows, we can construct a useful subcategory of **DF**.

Definition 5.10. $\mathbf{DF}_{\mathcal{N}_\mu}$ *is the category with the same objects as* **DF** *and arrows generated by the* **DF** *composition of arrows in* $\mathcal{N}_\mu$.

Proposition 5.11. *Given any arrow* $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ *in* $\mathbf{DF}_{\mathcal{N}_\mu}$ *and* $x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$, $f(\_, x_p, x_a) : \Omega^n \to \mathbb{R}^b$ *is a multivariate normal random variable defined on the probability space* $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$.

Proof. We will show that this property holds for the arrows in $\mathcal{N}_\mu$ and that it is preserved by composition.

To begin, note that for any n, m, the pushforward of μ^m along $f : \Omega^m \to \mathbb{R}^a$ is equivalent to the pushforward of μ^{m+n} along the following random variable:

$$f^l : \Omega^{m+n} \to \mathbb{R}^a$$
$$f^l(\omega_m, \omega_n) = f(\omega_m)$$

We can see this as follows. For any $\sigma_a \in \mathcal{B}(\mathbb{R}^a)$:

$$\begin{aligned}
& f^l_* \mu^{m+n} : \mathcal{B}(\mathbb{R}^a) \to [0,1] \\
& f^l_* \mu^{m+n}(\sigma_a) \\
= \quad & \{\text{Rewrite with } \delta\} \\
& \int_{(\omega_m, \omega_n) \in \Omega^{m+n}} \delta(f^l(\omega_m, \omega_n), \sigma_a) d\mu^{m+n} \\
= \quad & \{\text{Product measure}\} \\
& \int_{\omega_m \in \Omega^m} \int_{\omega_n \in \Omega^n} \delta(f^l(\omega_m, \omega_n), \sigma_a) d\mu^m d\mu^n \\
= \quad & \{\text{Definition of } f^l\} \\
& \int_{\omega_m \in \Omega^m} \delta(f(\omega_m), \sigma_a) d\mu^m \int_{\omega_n \in \Omega^n} d\mu^n \\
= \quad & \{\text{Rewrite as pushforward}\} \\
& f_* \mu^m(\sigma_a)
\end{aligned}$$

By a similar argument we have that the pushforward of μ^m along $f : \Omega^m \to \mathbb{R}^a$ is equivalent to the pushforward of μ^{n+m} along the random variable $f^r(\omega_n, \omega_m) = f(\omega_m)$.

Next, we note that for any $x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$ and arrow $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b \in \mathcal{N}_\mu$, the random variable $f(\_, x_p, x_a) : \Omega^n \to \mathbb{R}^b$ is multivariate normal and defined on the probability space $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$. This follows from the fact that for $\omega_n \in \Omega^n$:

$$f(\omega_n, x_p, x_a) = T(x_p, x_a) + G(\omega_n)$$

where $T(x_p, x_a)$ is a constant and $G : \Omega^n \to \mathbb{R}^b$ is multivariate normal. Next, we show that for any $x_p \in \mathbb{R}^p, x_q \in \mathbb{R}^q, x_a \in \mathbb{R}^a$, arrow $f' : \Omega^m \times \mathbb{R}^q \times \mathbb{R}^b \to \mathbb{R}^c$ in $\mathcal{N}_\mu$ and arrow $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ in **DF** such that the random variable $f(\_, x_p, x_a) : \Omega^n \to \mathbb{R}^b$ is multivariate normal, the random variable:

$$(f' \circ f)(\_, (x_q, x_p), x_a) : \Omega^{m+n} \to \mathbb{R}^b$$

is multivariate normal over $(\Omega^{m+n}, \mathcal{B}(\Omega^{m+n}), \mu^{m+n})$ since:

$$\begin{aligned}
&(f' \circ f)((\omega_m, \omega_n), (x_q, x_p), x_a) \\
=\quad &\{\text{Composition in } \mathbf{DF}\} \\
&f'(\omega_m, x_q, f(\omega_n, x_p, x_a)) \\
=\quad &\{\text{Definition of } f'\} \\
&T'(x_q, f(\omega_n, x_p, x_a)) + G'(\omega_m).
\end{aligned}$$

Since the random variable $f(\_, x_p, x_a) : \Omega^n \to \mathbb{R}^b$ is multivariate normal over $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$, by the note above we have that the random variable:

$$\begin{aligned}
&f^r(_, x_p, x_a) : \Omega^{m+n} \to \mathbb{R}^b \\
&f^r((\omega_m, \omega_n), x_p, x_a) = f(\omega_n, x_p, x_a)
\end{aligned}$$

defined over $(\Omega^{m+n}, \mathcal{B}(\Omega^{m+n}), \mu^{m+n})$ is multivariate normal. Since x_q is constant, this implies that the following random variable is also multivariate normal:

$$T'(x_q, f^r(\_, x_p, x_a)) : \Omega^{m+n} \to \mathbb{R}^c.$$

Similarly, the random variable:

$$\begin{aligned}
&G'^l : \Omega^{m+n} \to \mathbb{R}^b \\
&G'^l(\omega_m, \omega_n) = G'(\omega_m)
\end{aligned}$$

is also multivariate normal and independent of $T(x_q, f^r(\_, x_p, x_a))$. Therefore, we can write:

$$\begin{aligned}
&(f' \circ f)((\omega_m, \omega_n), (x_q, x_p), x_a) \\
=\quad &\{\text{Definition of } f'\} \\
&T'(x_q, f(\omega_n, x_p, x_a)) + G'(\omega_m) \\
=\quad &\{\text{Definition of } f^r, G'^r\} \\
&T'(x_q, f^r((\omega_m, \omega_n), x_p, x_a)) + G'^l(\omega_m, \omega_n)
\end{aligned}$$

Since this is a sum of independent normally distributed random variables, the following random variable is also multivariate normal:

$$(f' \circ f)(\_, (x_q, x_p), x_a) : \Omega^{m+n} \to \mathbb{R}^c.$$

□

As an aside, note that $\mathcal{N}_\mu$ itself is not closed under composition. Suppose

$$f' : \Omega^m \times \mathbb{R}^q \times \mathbb{R}^b \to \mathbb{R}^c$$
$$f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$

are in $\mathcal{N}_\mu$ and that:

$$f'(\omega_m, x_q, x_b) = T'(x_q, x_b) + G'(\omega_m)$$

where $T'(x_q, x_b) = \|x_q\|_1 x_b$. T' is Gaussian preserving since the product of a constant and a Gaussian is Gaussian. Now if we write:

$$f(\omega_n, x_p, x_a) = T(x_p, x_a) + G(\omega_n)$$

we see that:

$$(f' \circ f) : \Omega^{m+n} \times \mathbb{R}^{q+p} \times \mathbb{R}^a \to \mathbb{R}^c$$
$$(f' \circ f)((\omega_m, \omega_n), (x_q, x_p), x_a) = \|x_q\|_1 T(x_p, x_a) + \|x_q\|_1 G(\omega_n) + G'(\omega_m),$$

which we cannot express in general as a sum of a Gaussian preserving transformation over $\mathbb{R}^{q+p} \times \mathbb{R}^a \to \mathbb{R}^b$ and a multivariate normal random variable defined on $(\Omega^{n+m}, \mathcal{B}(\Omega^{n+m}), \mu^{n+m})$. $(f' \circ f)$ is therefore not in $\mathcal{N}_\mu$. However, for any choice of $x_q \in \mathbb{R}^q, x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$ the random variable:

$$(f' \circ f)(\_, (x_q, x_p), x_a) : \Omega^{n+m} \to \mathbb{R}^c$$

is a linear function of multivariate normal random variables and is therefore itself multivariate normal.

5.4.2.2 Relationship to Gauss

$\mathbf{DF}_{\mathcal{N}_\mu}$ is similar to the category **Gauss** from Section 6 of Fritz (2020), with a few key differences.

Definition 5.12. *In the category* **Gauss** *(Fritz, 2020) objects are natural numbers and morphisms $a \to b$ are tuples (M, C, s) where M is a matrix in $\mathbb{R}^{b \times a}$, C is a positive semidefinite matrix in $\mathbb{R}^{b \times b}$ and s is a vector in $\mathbb{R}^b$.*

Intuitively, the morphisms in **Gauss** represent transformations of random variables. That is, (M, C, s) implicitly represents the following transformation of random variables:

$$g(f) = Mf + \xi_{s,C}.$$

where $\xi_{s,C}$ is a multivariate normal random variable with mean s and covariance matrix C that is independent of f. If the random variable f is normally distributed, then $g(f)$ is as well.

For example, the morphism $(1, 1, 0) : 1 \to 1$ represents the transformation:

$$g(f) = f + \xi_{0,1}$$

where $\xi_{0,1}$ is a standard normal random variable.

A primary difference between **Gauss** and $\mathbf{DF}_{\mathcal{N}_\mu}$ is that the morphisms in $\mathbf{DF}_{\mathcal{N}_\mu}$ explicitly include the functional form of $\xi_{s,C}$ in the morphism itself. For any arrow $(M, C, s) : a \to b$ in **Gauss** and a choice of such an $\xi_{s,C}$ over $(\Omega, \mathcal{B}(\Omega), \mu)$, we can form the $\mathbf{DF}_{\mathcal{N}_\mu}$ arrow:

$$f' : \Omega \times \mathbb{R}^0 \times \mathbb{R}^a \to \mathbb{R}^b$$

where for $\omega \in \Omega, x_a \in \mathbb{R}^a$:

$$f'(\omega, x_a) = Mx_a + \xi_{s,C}(\omega).$$

However, this arrow is dependent on the choice of $\xi_{s,C}$.

5.4.3 Expectation Composition

Definition 5.13. *A subcategory* $\mathbf{C}$ *of* $\mathbf{DF}$ *is an Expectation Composition category if for any* $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ *and* $f' : \Omega^m \times \mathbb{R}^q \times \mathbb{R}^b \to \mathbb{R}^c$ *in* $\mathbf{C}$ *and* $x_q \in \mathbb{R}^q, x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$*:*

$$\int_{(\omega_m, \omega_n) \in \Omega^{m+n}} f'(\omega_m, x_q, f(\omega_n, x_p, x_a)) d\mu^{m+n} =$$
$$\int_{\omega_m \in \Omega^m} f' \left(\omega_m, x_q, \int_{\omega_n \in \Omega^n} f(\omega_n, x_p, x_a) d\mu^n \right) d\mu^m.$$

Proposition 5.14. $\mathbf{DF}_{\mathcal{N}_\mu}$ *is an Expectation Composition category.*

Proof. Consider some $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ and $f' : \Omega^m \times \mathbb{R}^q \times \mathbb{R}^b \to \mathbb{R}^c$ in $\mathbf{DF}_{\mathcal{N}_\mu}$ and $x_q \in \mathbb{R}^q, x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$. We will prove by induction that Definition 5.13 holds.

By the definition of $\mathbf{DF}_{\mathcal{N}_\mu}$, there exists some $k \in \mathbb{N}$ such that we can express f' as a composition of k arrows in $\mathcal{N}_\mu$. First note that if $k = 1$, then f' is in $\mathcal{N}_\mu$, and the statement must hold since for $x_q \in \mathbb{R}^q, x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$:

$$\int_{(\omega_m, \omega_n) \in \Omega^{m+n}} f'(\omega_m, x_q, f(\omega_n, x_p, x_a)) d\mu^{m+n}$$
$$= \quad \{f' \text{ is an arrow in } \mathcal{N}_\mu\}$$
$$\int_{(\omega_m, \omega_n) \in \Omega^{m+n}} T'(x_q, f(\omega_n, x_p, x_a)) + G'(\omega_m) d\mu^{m+n}$$
$$= \quad \{\text{Product measure}\}$$
$$\int_{\omega_m \in \Omega^m} \int_{\omega_n \in \Omega^n} T'(x_q, f(\omega_n, x_p, x_a)) d\mu^n + G'(\omega_m) d\mu^m$$
$$= \quad \{\text{Gaussian preserving transformation}\}$$
$$\int_{\omega_m \in \Omega^m} T' \left(x_q, \int_{\omega_n \in \Omega^n} f(\omega_n, x_p, x_a) d\mu^n \right) + G'(\omega_m) d\mu^m$$
$$= \quad \{\text{Definition of } f'\}$$
$$\int_{\omega_m \in \Omega^m} f' \left(\omega_m, x_q, \int_{\omega_n \in \Omega^n} f(\omega_n, x_p, x_a) d\mu^n \right) d\mu^m$$

Next, if $k > 1$ then we can express $f' = h \circ f'_{k-1}$, where h is in $\mathcal{N}_\mu$ and f'_{k-1} is the composition of $k-1$ arrows in $\mathcal{N}_\mu$. Without loss of generality we will assume f'_{k-1} and h have the following signatures:

$$f'_{k-1} : \Omega^{m'} \times \mathbb{R}^{q'} \times \mathbb{R}^b \to \mathbb{R}^d \qquad h : \Omega^{m''} \times \mathbb{R}^{q''} \times \mathbb{R}^d \to \mathbb{R}^c.$$

Note that $q' + q'' = q$ and $m' + m'' = m$. Now for any

$$x_{q''} \in \mathbb{R}^{q''}, x_{q'} \in \mathbb{R}^{q'}, x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$$

we can show the following:

$$\begin{aligned}
&\int_{(\omega_{m''},\omega_{m'},\omega_n)\in\Omega^{m''+m'+n}} f'((\omega_{m''},\omega_{m'}),(x_{q''},x_{q'}),f(\omega_n,x_p,x_a))d\mu^{m''+m'+n} \\
=\quad &\{\text{Rewrite } f' \text{ in terms of } h \text{ and } f'_{k-1}\} \\
&\int_{(\omega_{m''},\omega_{m'},\omega_n)\in\Omega^{m''+m'+n}} h(\omega_{m''},x_{q''},f'_{k-1}(\omega_{m'},x_{q'},f(\omega_n,x_p,x_a)))d\mu^{m''+m'+n} \\
=\quad &\{h \text{ is an arrow in } \mathcal{N}_\mu\} \\
&\int_{(\omega_{m''},\omega_{m'},\omega_n)\in\Omega^{m''+m'+n}} T_h(x_{q''},f'_{k-1}(\omega_{m'},x_{q'},f(\omega_n,x_p,x_a))) + G_h(\omega_{m''})d\mu^{m''+m'+n} \\
=\quad &\{\text{Gaussian preserving transformation}\} \\
&\int_{\omega_{m''}\in\Omega^{m''}} T_h\left(x_{q''},\int_{(\omega_{m'},\omega_n)\in\Omega^{m'+n}} f'_{k-1}(\omega_{m'},x_{q'},f(\omega_n,x_p,x_a))d\mu^{m'+n}\right) + G_h(\omega_{m''})d\mu^{m''} \\
=\quad &\{\text{Apply inductive step}\} \\
&\int_{\omega_{m''}\in\Omega^{m''}} T_h\left(x_{q''},\int_{\omega_{m'}\in\Omega^{m'}} f'_{k-1}\left(\omega_{m'},x_{q'},\int_{\omega_n\in\Omega^n} f(\omega_n,x_p,x_a)d\mu^n\right)d\mu^{m'}\right) + G_h(\omega_{m''})d\mu^{m''} \\
=\quad &\{\text{Gaussian preserving transformation}\} \\
&\int_{(\omega_{m''},\omega_{m'})\in\Omega^{m''+m'}} T_h\left(x_{q''},f'_{k-1}\left(\omega_{m'},x_{q'},\int_{\omega_n\in\Omega^n} f(\omega_n,x_p,x_a)d\mu^n\right)\right) + G_h(\omega_{m''})d\mu^{m''+m'} \\
=\quad &\{\text{Definition of } h\} \\
&\int_{(\omega_{m''},\omega_{m'})\in\Omega^{m''+m'}} h\left(\omega_{m''},x_{q''},f'_{k-1}\left(\omega_{m'},x_{q'},\int_{\omega_n\in\Omega^n} f(\omega_n,x_p,x_a)d\mu^n\right)\right)d\mu^{m''+m'} \\
=\quad &\{\text{Definition of } f'\} \\
&\int_{(\omega_{m''},\omega_{m'})\in\Omega^{m''+m'}} f'\left((\omega_{m''},\omega_{m'}),(x_{q''},x_{q'}),\int_{\omega_n\in\Omega^n} f(\omega_n,x_p,x_a)d\mu^n\right)d\mu^{m''+m'}
\end{aligned}$$

By induction we have that the original statement holds for all $f', f \in \mathbf{DF}_{\mathcal{N}_\mu}$. □

We can now define the following functor:

Proposition 5.15. *Suppose* $\mathbf{C} \subseteq \mathbf{DF}$ *is an Expectation Composition category. We can define a map* $Exp : \mathbf{C} \to \mathbf{Para}(\mathbf{EucMeas})$ *that acts as the identity on objects and sends*

the arrow $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ *in* $\mathbf{C}$ *to the following function:*

$$f_E : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$
$$f_E(x_p, x_a) = E_{\mu^n}[f(\_, x_p, x_a)] = \int_{\omega_n \in \Omega^n} f(\omega_n, x_p, x_a) d\mu^n$$

$Exp : \mathbf{C} \to \mathbf{Para}(\mathbf{EucMeas})$ *is a functor.*

Proof. To start, note that Exp trivially sends objects in $\mathbf{C}$ to objects in $\mathbf{Para}(\mathbf{EucMeas})$. Next, note that for any morphism in $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ in $\mathbf{C}$ since f is infinitely differentiable the following function is differentiable and therefore also Borel measurable:

$$f_E : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$
$$f_E(x_p, x_a) = E_{\mu^n}[f(\_, x_p, x_a)] = \int_{\omega_n \in \Omega^n} f(\omega_n, x_p, x_a) d\mu^n$$

Therefore Exp sends morphisms in $\mathbf{C}$ to morphisms in $\mathbf{Para}(\mathbf{EucMeas})$.

Next, Exp preserves identities since $Exp(id)$ is the identity function in $\mathbf{Para}(\mathbf{EucMeas})$

$$Exp(id)(x_a) = E_{\mu^n}[id(\_, x_a)] = E_{\mu^n}[x_a] = x_a$$

Finally, consider the morphisms $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ and $f' : \Omega^m \times \mathbb{R}^q \times \mathbb{R}^b \to \mathbb{R}^c$ in $\mathbf{C}$. We have that for $x_q \in \mathbb{R}^q, x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$:

$$\begin{aligned}
& Exp(f' \circ f) : \mathbb{R}^{q+p} \times \mathbb{R}^a \to \mathbb{R}^c \\
& Exp(f' \circ f)(x_q, x_p, x_a) \\
= \quad & \{\text{Definition of } Exp\} \\
& \int_{(\omega_m, \omega_n) \in \Omega^{m+n}} (f' \circ f)((\omega_m, \omega_n), (x_q, x_p), x_a) d\mu^{m+n} \\
= \quad & \{\text{Apply } \mathbf{DF} \text{ composition}\} \\
& \int_{(\omega_m, \omega_n) \in \Omega^{m+n}} f'(\omega_m, x_q, f(\omega_n, x_p, x_a)) d\mu^{m+n} \\
= \quad & \{\mathbf{C} \text{ is an Expectation Composition category}\} \\
& \int_{\omega_m \in \Omega^m} f' \left(\omega_m, x_q, \int_{\omega_n \in \Omega^n} f(\omega_n, x_p, x_a) d\mu^n \right) d\mu^m \\
= \quad & \{\text{Definition of } Exp\} \\
& Exp(f')(x_q, Exp(f)(x_p, x_a)) \\
= \quad & \{\text{Composition in } \mathbf{Para}(\mathbf{EucMeas})\} \\
& (Exp(f') \circ Exp(f))(x_q, x_p, x_a))
\end{aligned}$$

This implies that Exp preserves composition. □

5.5 Likelihood and Learning

In this section we will apply the maximum likelihood procedure to the arrows in **DF** to derive the error function $er : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. We will then use this error function to define

a modification of the backpropagation functor by Fong et al. (2019). However, since different arrows in **DF** have likelihood functions of different forms, we will not define a single backpropagation functor out of **DF**. Instead, we will define multiple functors from subcategories of **DF** into **Learn**.

To do this, we will first define a substructure of **DF** with well-defined likelihood functions. Then, we will describe a class of subcategories of **DF** derived from this substructure. Finally, we will define a backpropagation functor for any subcategory in this class.

5.5.1 Conditional Likelihood

The conditional likelihood is a general measure of the goodness of fit of a set of parameters and observed data for a given parametric statistical model. We can define the conditional likelihood of a parametric statistical model $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ over the probability space $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$ at the points $x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a, x_b \in \mathbb{R}^b$ in terms of the pushforward measure of μ^n along the random variable $f(\_, x_p, x_a)$. To do this, we evaluate the Radon-Nikodym derivative of the probability measure:

$$f(\_, x_p, x_a)_* \mu^n : \mathcal{B}(\mathbb{R}^b) \to [0, 1]$$
$$f(\_, x_p, x_a)_* \mu^n = \mu^n(f(\_, x_p, x_a)^{-1})$$

with respect to a reference measure at the point x_b. We select the Lebesgue measure λ^b over $\mathbb{R}^b$ as the reference measure.

Note that the Radon-Nikodym derivative with respect to the Lebesgue measure is not defined for all measures. For example, no discrete measure has a Radon-Nikodym derivative with respect to the Lebesgue measure, since for any finite collection of points A in $\mathbb{R}^b$ we have $\lambda^b(A) = 0$.

Proposition 5.16. *Given a probability space $(\Omega, \mathcal{B}(\Omega), \mu)$ define $\mathbf{DF}_{\mathcal{R}_\mu}$ to be the substructure of* **DF** *with the same objects, but with morphisms from $\mathbb{R}^a$ to $\mathbb{R}^b$ limited to $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ such that the following Borel measurable and Lebesgue integrable function exists:*

$$L_f : \mathbb{R}^p \times \mathbb{R}^a \times \mathbb{R}^b \to \mathbb{R}$$
$$L_f(x_p, x_a, x_b) = \frac{d\ f(\_, x_p, x_a)_* \mu^n}{d\lambda^b}(x_b)$$

That is:

$$f(\_, x_p, x_a)_* \mu^n(\sigma_b) = \int_{x_b \in \sigma_b} L_f(x_p, x_a, x_b) d\lambda^b$$

$\mathbf{DF}_{\mathcal{R}_\mu}$ *is a semicategory.*

Proof. Since composition in $\mathbf{DF}_{\mathcal{R}_\mu}$ is the same as in **DF**, we simply need to show that $\mathbf{DF}_{\mathcal{R}_\mu}$ is closed under **DF** composition.

Suppose $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ and $f' : \Omega^m \times \mathbb{R}^q \times \mathbb{R}^b \to \mathbb{R}^c$ are arrows in $\mathbf{DF}_{\mathcal{R}_\mu}$. We can show that for all $x_q \in \mathbb{R}^q, x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a$ there exists some Borel measurable

and Lebesgue integrable $g : \mathbb{R}^c \to \mathbb{R}$ such that for $\sigma_c \in \mathcal{B}(\mathbb{R}^c)$:

$$(f' \circ f)(\_, (x_q, x_p), x_a)_* \mu^{m+n} : \mathcal{B}(\mathbb{R}^c) \to [0, 1]$$
$$(f' \circ f)(\_, (x_q, x_p), x_a)_* \mu^{m+n}(\sigma_c) = \int_{x_c \in \sigma_c} g(x_c) d\lambda^c$$

where λ^c is the Lebesgue measure over $\mathbb{R}^c$:

$$\begin{aligned}
&(f' \circ f)(_, (x_q, x_p), x_a)_* \mu^{m+n}(\sigma_c) \\
= &\quad \{\text{Definition of pushforward}\} \\
&\int_{(\omega_m, \omega_n) \in \Omega^m \times \Omega^n} \delta((f' \circ f)((\omega_m, \omega_n), (x_q, x_p), x_a), \sigma_c) d\mu^{n+m} \\
= &\quad \{\text{Composition in } \mathbf{DF}\} \\
&\int_{\omega_m \in \Omega^m} \int_{\omega_n \in \Omega^n} \delta(f'(\omega_m, x_q, f(\omega_n, x_p, x_a)), \sigma_c) d\mu^n d\mu^m \\
= &\quad \{\text{Rewrite with } \delta\} \\
&\int_{\omega_m \in \Omega^m} \int_{\omega_n \in \Omega^n} \int_{x_b \in \mathbb{R}^b} \delta(f'(\omega_m, x_q, x_b), \sigma_c) d\delta(f(\omega_n, x_p, x_a), _) d\mu^n d\mu^m \\
= &\quad \{\text{Rearrange integrals}\} \\
&\int_{x_b \in \mathbb{R}^b} \left[\int_{\omega_m \in \Omega^m} \delta(f'(\omega_m, x_q, x_b), \sigma_c) d\mu^m \right] d \left[\int_{\omega_n \in \Omega^n} \delta(f(\omega_n, x_p, x_a), _) d\mu^n \right] \\
= &\quad \{\text{Rewrite as pushforward}\} \\
&\int_{x_b \in \mathbb{R}^b} f'(_, x_q, x_b)_* \mu^m(\sigma_c) \; df(_, x_p, x_a)_* \mu^n \\
= &\quad \{f' \text{ and } f' \text{ are arrows in } \mathbf{DF}_{\mathcal{R}_\mu}\} \\
&\int_{x_b \in \mathbb{R}^b} \left[\int_{x_c \in \sigma_c} \frac{df'(_, x_q, x_b)_* \mu^m}{d\lambda^c}(x_c) d\lambda^c \right] \left[\frac{df(_, x_p, x_a)_* \mu^n}{d\lambda^b}(x_b) d\lambda^b \right] \\
= &\quad \{\text{Rearrange integrals}\} \\
&\int_{x_c \in \sigma_c} \left[\left(\int_{x_b \in \mathbb{R}^b} \frac{df'(_, x_q, x_b)_* \mu^m}{d\lambda^c}(x_c) \right) \left(\frac{df(_, x_p, x_a)_* \mu^n}{d\lambda^b}(x_b) d\lambda^b \right) \right] d\lambda^c
\end{aligned}$$

Therefore we have that:

$$L_{f' \circ f}((x_q, x_p), x_a, x_c) = \frac{d(f' \circ f)(\_, (x_q, x_p), x_a)_* \mu^{m+n}}{d\lambda^c}(x_c)$$

exists and is equal to:

$$\int_{x_b \in \mathbb{R}^b} \left(\frac{df'(\_, x_q, x_b)_* \mu^m}{d\lambda^c}(x_c) \right) \left(\frac{df(\_, x_p, x_a)_* \mu^n}{d\lambda^b}(x_b) \right) d\lambda^b =$$
$$\int_{x_b \in \mathbb{R}^b} L_{f'}(x_q, x_b, x_c) L_f(x_p, x_a, x_b) d\lambda^b$$

$L_{f' \circ f}$ is therefore Lebesgue integrable and Borel measurable since $L_{f'}$ and L_f are Lebesgue integrable and Borel measurable.

□

5.5.2 Maximum Likelihood

Suppose we have a probability space $(\mathbb{R}^a \times \mathbb{R}^b, \mathcal{B}(\mathbb{R}^a \times \mathbb{R}^b), \tau)$. The maximum expected log-likelihood estimator for $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ in $\mathbf{DF}_{\mathcal{R}_\mu}$ with respect to τ is the vector $x_p \in \mathbb{R}^p$ that maximizes the following function. Note that *log* is a monotonic transformation and we just use it to make the mathematics easier. The optimal value of x_p is the same with or without it.

$$L_\tau : \mathbb{R}^p \to \mathbb{R}$$
$$L_\tau(x_p) = \int_{(x_a, x_b) \in \mathbb{R}^a \times \mathbb{R}^b} log \frac{df(\_, x_p, x_a)_* \mu^n}{d\lambda^b}(x_b) d\tau.$$

That is, the maximum expected log-likelihood estimator for f with respect to τ is the vector x_p that maximizes the expected value of:

$$log \frac{df(\_, x_p, x_a)_* \mu^n}{d\lambda^b}(x_b)$$

over τ.

Now suppose that instead of observing a probability space $(\mathbb{R}^a \times \mathbb{R}^b, \mathcal{B}(\mathbb{R}^a \times \mathbb{R}^b), \tau)$ directly we have a dataset of samples:

$$S_n = \{(x_{a_1}, x_{b_1}), (x_{a_2}, x_{b_2}), \cdots, (x_{a_n}, x_{b_n})\}$$

in $\mathbb{R}^a \times \mathbb{R}^b$.

Definition 5.17. *The maximum log likelihood estimator for the arrow*

$$f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$

in $\mathbf{DF}_{\mathcal{R}_\mu}$ *with respect to the samples:*

$$(x_{a_1}, x_{b_1}), (x_{a_2}, x_{b_2}), \cdots, (x_{a_n}, x_{b_n}) \in \mathbb{R}^a \times \mathbb{R}^b$$

is the vector $x_p \in \mathbb{R}^p$ *that maximizes the function:*

$$L_{S_n}(x_p) : \mathbb{R}^p \to \mathbb{R}$$
$$L_{S_n}(x_p) = \sum_{i=1}^{n} log \frac{df(\_, x_p, x_{a_i})_* \mu^n}{d\lambda^b}(x_{b_i}).$$

If we assume the samples in S_n are drawn from $(\mathbb{R}^a \times \mathbb{R}^b, \mathcal{B}(\mathbb{R}^a \times \mathbb{R}^b), \tau)$, then by the weak law of large numbers $\frac{1}{n} L_{S_n}$ converges to L_τ in probability as $n \to \infty$.

However, it will be challenging to derive an objective function for the backpropagation functor of Fong et al. (2019) from L_{S_n} directly, since their construction assumes that the error function has the signature $er : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and has an invertible derivative. We will slightly modify L_{S_n} to make this easier.

For any $j \leq b$, the jth component of $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ in $\mathbf{DF}_{\mathcal{R}_\mu}$ is the function:

$$f[j] : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}$$

and the marginal likelihood at $x_p \in \mathbb{R}^p$ of this component for some sample $(x_{a_i}, x_{b_i}) \in S_n$ is:

$$l_{ij} : \mathbb{R}^p \to \mathbb{R}$$
$$l_{ij}(x_p) = \frac{df(\_, x_p, x_{a_i})[j]_* \mu^n}{d\lambda}(x_{b_i}[j])$$

where we write $x_{b_i}[j] \in \mathbb{R}$ for the jth component of the vector $x_{b_i} \in \mathbb{R}^b$. We can now define the following:

Definition 5.18. *The maximum log-marginal likelihood estimator for the arrow*

$$f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$

in $\mathbf{DF}_{\mathcal{R}_\mu}$ *with respect to the samples:*

$$(x_{a_1}, x_{b_1}), (x_{a_2}, x_{b_2}), \cdots, (x_{a_n}, x_{b_n}) \in \mathbb{R}^a \times \mathbb{R}^b$$

is the vector $x_p \in \mathbb{R}^p$ *that maximizes the function:*

$$M_{S_n} : \mathbb{R}^p \to \mathbb{R}$$
$$M_{S_n}(x_p) = \sum_{i=1}^{n} \sum_{j=1}^{b} log\ l_{ij}(x_p).$$

where $l_{ij}(x_p)$ *is the marginal likelihood at* $x_p \in \mathbb{R}^p$ *of the* j*th component of* f *for* $(x_{a_i}, x_{b_i}) \in S_n$.

We have $M_{S_n}(x_p) = L_{S_n}(x_p)$ when the real valued random variables:

$$f(\_, x_p, x_{a_i})[j] : \Omega^n \to \mathbb{R}$$

are mutually independent for all x_{a_i} and $j \leq b$.

This suggests a criterion for an error function $er : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ over which we can define the backpropagation functor: we want the following two real valued functions of $\mathbb{R}^p$ to move in tandem for any fixed $(x_a, y) \in \mathbb{R}^a \times \mathbb{R}$ and $j \leq b$:

$$l(x_p) = er\left(E_{\mu^n}[f(\_, x_p, x_a)[j]], y\right)$$
$$l'(x_p) = -\frac{df(\_, x_p, x_a)[j]_* \mu^n}{d\lambda}(y).$$

We will now make this formal.

5.5.3 Learning from Likelihoods

Suppose we have a real valued random variable f over the probability space $(\Omega^n, \mathcal{B}(\Omega^n), \mu^n)$. Write $E_{\mu^n}[f] \in \mathbb{R}$ for the expectation of f over μ^n:

$$E_{\mu^n}[f] = \int_{\omega_n \in \Omega^n} f(\omega_n)\ d\mu^n.$$

And define f^0 to be:

$$f^0(\omega_n) = f(\omega_n) - E_{\mu^n}[f].$$

Proposition 5.19. *Given an Expectation Composition category* $\mathbf{C}$*, there exists a semicategory* $\mathbf{C}_{\mathcal{R}_\mu}$ *with the same objects as* $\mathbf{C}$*, for any objects* $\mathbb{R}^a, \mathbb{R}^b \in \mathbf{C}$*:*

$$\mathbf{C}_{\mathcal{R}_\mu}[\mathbb{R}^a, \mathbb{R}^b] = \mathbf{C}[\mathbb{R}^a, \mathbb{R}^b] \cap \mathbf{DF}_{\mathcal{R}_\mu}[\mathbb{R}^a, \mathbb{R}^b].$$

and composition in $\mathbf{C}_{\mathcal{R}_\mu}$ *is* $\mathbf{DF}$ *composition. We call* $\mathbf{C}_{\mathcal{R}_\mu}$ *the* $\mathcal{R}_\mu$*-semicategory of* $\mathbf{C}$*.*

Proof. Since $\mathbf{C}$ and $\mathbf{DF}_{\mathcal{R}_\mu}$ are small, the intersection:

$$\mathbf{C}_{\mathcal{R}_\mu}[\mathbb{R}^a, \mathbb{R}^b] = \mathbf{C}[\mathbb{R}^a, \mathbb{R}^b] \cap \mathbf{DF}_{\mathcal{R}_\mu}[\mathbb{R}^a, \mathbb{R}^b].$$

is well-defined. Note also that if we have:

$$f_1 \in \mathbf{C}[\mathbb{R}^a, \mathbb{R}^b], f_1 \in \mathbf{DF}_{\mathcal{R}_\mu}[\mathbb{R}^a, \mathbb{R}^b]$$
$$f_2 \in \mathbf{C}[\mathbb{R}^b, \mathbb{R}^c], f_2 \in \mathbf{DF}_{\mathcal{R}_\mu}[\mathbb{R}^b, \mathbb{R}^c]$$

Then since $\mathbf{C}$ and $\mathbf{DF}_{\mathcal{R}_\mu}$ are closed under $\mathbf{DF}$ composition it must be that:

$$f_2 \circ f_1 \in \mathbf{C}[\mathbb{R}^a, \mathbb{R}^c] \cap \mathbf{DF}_{\mathcal{R}_\mu}[\mathbb{R}^a, \mathbb{R}^c]$$

Therefore $\mathbf{C}_{\mathcal{R}_\mu}$ is a semicategory. □

Definition 5.20. *Suppose* $\mathbf{C}$ *is an Expectation Composition category. Then* $\mathbf{C}$ *is a Marginal Likelihood Factorization Category over the measure* $\mu : \mathcal{B}(\Omega) \to [0, 1]$ *if there exists:*

- *A differentiable function with invertible derivative* $er : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- *For each* $n \in \mathbb{N}$*, a function* $\alpha_n : (\Omega^n \to \mathbb{R}) \to \mathbb{R}$
- *For each* $n \in \mathbb{N}$*, a nonnegative function* $\beta_n : (\Omega^n \to \mathbb{R}) \to \mathbb{R}$

such that for any:

$$x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a, j \leq b$$

and arrow $f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$ *in the* $\mathcal{R}_\mu$*-semicategory* $\mathbf{C}_{\mathcal{R}_\mu}$ *of* $\mathbf{C}$ *we can write:*

$$log\ \frac{df(\_, x_p, x_a)[j]_* \mu^n}{d\lambda} : \mathbb{R} \to \mathbb{R}$$

$$log \frac{df(\_, x_p, x_a)[j]_* \mu^n}{d\lambda}(y) =$$
$$\alpha_n(f^0(\_, x_p, x_a)[j]) - \beta_n(f^0(\_, x_p, x_a)[j]) er\left(E_{\mu^n}[f(\_, x_p, x_a)[j]], y\right)$$

We will refer to er *as a marginal error function of* $\mathbf{C}$*.*

Theorem 5.21. $\mathbf{DF}_{\mathcal{N}_\mu}$ *is a Marginal Likelihood Factorization Category with a marginal error function* $er(a, b) = (a - b)^2$*.*

Proof. Suppose $\mathbf{C}_{\mathcal{R}_\mu}$ is the $\mathcal{R}_\mu$-semicategory of $\mathbf{DF}_{\mathcal{N}_\mu}$. Now consider some

$$f : \Omega^n \times \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$$

in $\mathbf{C}_{\mathcal{R}_\mu}$, and note that for any:

$$x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a, j \leq b$$

the random variable:

$$f(\_, x_p, x_a)[j] : \Omega^n \to \mathbb{R}$$

is univariate normal (Proposition 5.11). For each $n \in \mathbb{N}$ we also define the standard deviation function:

$$s_n : (\Omega^n \to \mathbb{R}) \to \mathbb{R}$$
$$s_n(g) = \sqrt{E_{\mu_n}[(g - E_{\mu_n}[g])^2]}$$

Since f is in $\mathbf{C}_{\mathcal{R}_\mu}$, it must be that $s_n(f) > 0$.

Now for any $x_p \in \mathbb{R}^p, x_a \in \mathbb{R}^a, y \in \mathbb{R}, j \leq b$ we can write:

$$log\ \frac{df(\_, x_p, x_a)[j]_*\mu^n}{d\lambda} : \mathbb{R} \to \mathbb{R}$$
$$log\left(\frac{df(\_, x_p, x_a)[j]_*\mu^n}{d\lambda}(y)\right)$$
$$= \quad \{\text{The random variable } f(\_, x_p, x_a)[j] \text{ is univariate normal with nonzero variance}\}$$
$$log\left(\left(\frac{1}{s_n(f(\_, x_p, x_a)[j])\sqrt{2\pi}}\right)\exp\left(-\frac{\left(y - E_{\mu^n}[f(\_, x_p, x_a)[j]]\right)^2}{2s_n(f(\_, x_p, x_a)[j])^2}\right)\right)$$
$$= \quad \{\text{Apply logarithm}\}$$
$$-\frac{log(2\pi s_n(f(\_, x_p, x_a)[j])^2)}{2} - \frac{1}{2s_n(f(\_, x_p, x_a)[j])^2}\left(y - E_{\mu^n}[f(\_, x_p, x_a)[j]]\right)^2$$

Therefore:

$$\alpha_n(g) = -\frac{log(2\pi s_n(g)^2)}{2}$$
$$\beta_n(g) = \frac{1}{2s_n(g)^2}$$
$$er(a, b) = (a - b)^2$$

□

5.5.4 Backpropagation

The arrows in a Marginal Likelihood Factorization Category $\mathbf{C}$ are equipped with the structure that we need to derive both an optimization objective and a learning procedure. Therefore, for any Marginal Likelihood Factorization Category $\mathbf{C}$ and choice of learning rate ϵ we can define a backpropagation functor into the category **Learn** (Fong et al., 2019).

Definition 5.22. *Write F_{er} for the backpropagation functor of Fong et al. (2019) with learning rate ϵ under the marginal error function er of* $\mathbf{C}$*. We define the functor E_{er} to map a parametric statistical model in* $\mathbf{C}$ *to a learning algorithm:*

$$E_{er} : \mathbf{C} \to \mathbf{Learn}$$
$$E_{er} = F_{er} \circ Exp$$

Where Exp is defined in Proposition 5.15. For example, E_{er} sends parametric statistical models in $\mathbf{DF}_{\mathcal{N}_\mu}$ to learning algorithms that minimize the square error function with gradient descent. We can think of E_{er} as a point estimation functor: it sends an arrow f in $\mathbf{C}$ to a learner whose inference function is formed from f's expectation. The higher order moments of the pushforward distributions of the arrows in $\mathbf{C}$ are then used to define the loss function er.

5.6 Closing Thoughts on Categorical Stochastic Processes and Likelihood

Consider once again a physical system that is composed of several components, each of which has some degree of aleatoric uncertainty. If we construct a neural network model for this system like we describe in Section 4.1.1, we cannot characterize the interactions between the uncertainty in the different parts of the system. However, if we model the components of the system as stochastic processes and apply **DF** composition, we can capture how the uncertainty of the component parts combine. For example, given estimates of the kinds of uncertainties inherent to the photoreceptors in the eye, edge-detecting neurons in primary visual cortex, and higher-order feature detectors in the later stages of visual cortex, we may be able to build a more realistic model of how these sources of uncertainty interact than the one that Eberhardt et al. (2016) use to assess how the visual cortex performs a rapid stimulus categorization task.

Once we build such a model, we can use E_{er} to derive a Learner with a structure that incorporates this combined uncertainty. This functor will convert the model to a point estimator and bundle the combined uncertainty into a loss function.

One of the largest differences between this construction and those of Cho and Jacobs (2019) and Culbertson and Sturtz (2013) is the treatment of model updates in the face of new data. While these authors also describe categorical frameworks in which we can model how a new observation updates the parameters of a statistical model, they primarily study Bayesian algorithms in which the model parameters are represented with a probability distribution.

In contrast, our construction is inherently frequentist. While the backpropagation functor above aims to find an optimal parameter value given the data we have seen, it makes no assumptions about what that value may be. Although uncertainty motivates the objective that our parameter estimation procedure aims to optimize, the optimization algorithm does not use it directly.

Furthermore, our current definition of Marginal Likelihood Factorization Categories may be overly restrictive. For example, our definition specifies that each category is characterized by a single marginal error function er. This is a major limitation, and is why $\mathbf{DF}_{\mathcal{N}_\mu}$ is a Marginal Likelihood Factorization Category but **DF** is not. In order to expand our construction to work for a more general category like **DF** we may need to modify our definition of Marginal Likelihood Factorization Categories to support attaching different error functions er to each morphism.

6 Unsupervised Learning

Unsupervised learning algorithms extract structure from unlabelled data. Since these algorithms operate without labels, the kinds of structures a particular unsupervised algorithm will extract is very dependent on that algorithm's properties.

An important property of unsupervised learning algorithms is the collection of dataset transformations to which they are invariant or equivariant. Studying this property allows us to better understand how an algorithm separates signal from noise.

By characterizing an algorithm as a functor or natural transformation we can encode these invariances and equivariances in the morphisms of the source and target category. This enables us to draw conclusions about the behavior of the algorithm from the behaviors of these categories and functors between them.

6.1 Functorial Overlapping Clustering via Simplicial Complexes

In this section we explore the underlying relationships between simplicial complexes and clustering algorithms. We start by introducing a pair of adjoint functors between the category of finite nonnested flag covers **Cov** (Definition 2.65) and the category of finite simplicial complexes **SCpx** (Definition 2.52). Next, we demonstrate that the hierarchical versions of the single linkage clustering functor (Definition 3.11) and maximal linkage clustering functor (Definition 3.14) factor through the fuzzy singular set functor (McInnes et al., 2018). We use this insight to build on the results by Culbertson et al. (2016) and demonstrate that the single and maximal linkage clustering functors lie at two ends of a spectrum of hierarchical clustering functors.

6.1.1 Flat Clustering with and without Overlaps

6.1.1.1 Flat Clustering Functors

Nonoverlapping clustering algorithms transform ubermetric spaces $(X, d_X) \in \mathbf{UMet}$ (Definition 2.37) into partitions of X.

Definition 6.1. *Given a subcategory* $\mathbf{D}$ *of* **UMet***, a nonoverlapping flat* $\mathbf{D}$*-clustering functor is a functor* $F : \mathbf{D} \to \mathbf{Part}$ *that is the identity on morphisms and underlying sets.*

That is, a nonoverlapping flat (flat as in non-hierarchical, not related to other category theoretic definitions of the term) $\mathbf{D}$-clustering functor commutes with the forgetful functors from $\mathbf{D}$ and **Part** into **Set**. In the case that $\mathbf{D}$ is unspecified we simply call it a nonoverlapping flat clustering functor.

Overlapping clustering algorithms transform ubermetric spaces $(X, d_X) \in \mathbf{UMet}$ into nonnested flag covers of X (Definition 2.62). This lets us avoid some of the challenges of functorial nonoverlapping clustering. For example, given a chain of points between two dense clusters, a nonoverlapping clustering functor must either ignore the chain or collapse the dense clusters into one. An overlapping clustering functor can add the points in the chain to each of the dense clusters without joining them. In this way overlapping clustering algorithms can preserve more information about the original space.

Definition 6.2. *Given a subcategory* $\mathbf{D}$ *of* **UMet***, an overlapping flat* $\mathbf{D}$*-clustering functor is a functor* $F : \mathbf{D} \to \mathbf{Cov}$ *that is the identity on morphisms and underlying sets.*

Any nonoverlapping flat **D**-clustering functor is also an overlapping flat **D**-clustering functor, but the opposite is not necessarily true. Now define Λ_ϵ^1 to be the two-element metric space where the distance between the two elements is $\epsilon \in \mathbb{R}_{\geq 0}$. Most useful overlapping flat clustering functors are in the following class:

Definition 6.3. *Given a subcategory* **D** *of* **UMet**, *a non-trivial overlapping flat* **D***-clustering functor* F *is an overlapping flat clustering functor such that there exists a clustering parameter* $\delta_F \in \mathbb{R}_{\geq 0}$ *where* $F(\Lambda_\epsilon^1)$ *is two singleton clusters for any* $\epsilon > \delta_F$ *and a single two point cluster for any* $\epsilon \leq \delta_F$.

Intuitively, the clustering parameter δ_F is the maximum distance at which the clustering functor will identify two points as being part of the same cluster.

6.1.2 Hierarchical Clustering with and without Overlaps

One of the largest challenges when building clustering algorithms is the difficulty in capturing structure that exists at different scales. The Kleinberg Impossibility Theorem states that any scale invariant clustering algorithm must sacrifice either surjectivity or consistency (Kleinberg, 2003). As we discussed in Section 3.3.1.2, one way to get around this is to generate a series of clusterings, each at a different scale. In order to do this we replace the codomain of our clustering functors with categories that can represent groupings of points at multiple scales.

To start, recall the category **FPart** of fuzzy partitions (Definition 2.72):

Definition 6.4. *Given a subcategory* **D** *of* **UMet**, *a nonoverlapping hierarchical* **D***-clustering functor is a functor* $H : \mathbf{D} \to \mathbf{FPart}$ *such that for* $a \in (0, 1]^{op}$, $H(\_)(a) : \mathbf{D} \to \mathbf{Part}$ *is a nonoverlapping flat* **D***-clustering functor.*

Intuitively, a nonoverlapping hierarchical **D**-clustering functor maps a pair of a dataset (X, d_X) and a strength $a \in (0, 1]^{\text{op}}$ to a partition of the set X in a way such that increasing the distances between points in X or increasing the strength a may cause clusters to separate.

We can similarly define a hierarchical extension of overlapping clustering functors as functors into the category **FCov** of fuzzy nonnested flag covers (Definition 2.71):

Definition 6.5. *Given a subcategory* **D** *of* **UMet**, *an overlapping hierarchical* **D***-clustering functor is a functor* $H : \mathbf{D} \to \mathbf{FCov}$ *such that for* $a \in (0, 1]^{op}$, $H(\_)(a) : \mathbf{D} \to \mathbf{Cov}$ *is an overlapping flat* **D***-clustering functor.*

Most useful overlapping hierarchical clustering functors are in the following class:

Definition 6.6. *A non-trivial overlapping hierarchical* **D***-clustering functor* H *is an overlapping hierarchical* **D***-clustering functor such that for all* $a \in (0, 1]^{op}$, $H(\_)(a) : \mathbf{D} \to \mathbf{Cov}$ *is a non-trivial overlapping flat* **D***-clustering functor with clustering parameter* $\delta_{H,a}$ *(Definition 6.3).*

6.1.3 Simplicial Complexes and Clustering

We can now explore how the relationship between nonnested flag covers (Definition 2.62) and simplicial complexes (Definition 2.46) enables us to reason about clustering algorithms.

6.1.3.1 Flagification Functor

In the category **Cov** each nonnested flag cover $\mathcal{C}_X$ is uniquely determined by the 2-element subsets of the sets in $\mathcal{C}_X$. Consider two objects $(X, \mathcal{C}_X)$ and $(Y, \mathcal{C}_Y)$ in **Cov** and a function $f : X \to Y$. The condition that for any set S in $\mathcal{C}_X$ there exists some set S' in $\mathcal{C}_Y$ such that $f(S) \subseteq S'$ is equivalent to the condition that for any 2-element subset $\{x_1, x_2\}$ of any set in $\mathcal{C}_X$ there exists a set in $\mathcal{C}_Y$ that contains $\{f(x_1), f(x_2)\}$.

Given a nonnested flag cover $(X, \mathcal{C}_X)$ in **Cov**, we can generate a flag complex by treating the sets in $\mathcal{C}_X$ as the maximal simplices of the complex and adding in all of the subsets of these sets as the lower-dimensional simplices.

Definition 6.7. *The functor $S_{fl} : \mathbf{Cov} \to \mathbf{SCpx}$ maps the tuple $(X, \mathcal{C}_X)$ in **Cov** to the simplicial complex S_X whose faces are the subsets of sets in $\mathcal{C}_X$. The set of vertices (0-simplices) in S_X is X. S_{fl} acts as the identity on morphisms.*

Next, recall that we can convert any cover U of a finite set X into a nonnested flag cover by flagification (Definition 2.63). For example, consider some simplicial complex S_X with underlying set X. The simplices of S_X form a cover of X, which we can convert to a nonnested flag cover. We can use this procedure to develop the following functor:

Definition 6.8. *The functor $Flag : \mathbf{SCpx} \to \mathbf{Cov}$ maps the simplicial complex $S_X \in \mathbf{SCpx}$ to the tuple $(X, \mathcal{C}_X)$ where $\mathcal{C}_X$ is the flagification of the cover formed from the simplices of S_X. $Flag$ acts as the identity on morphisms.*

Proposition 6.9. *There exists an adjunction $S_{fl} \dashv Flag$.*

Proof. We will show that there exists an isomorphism:

$$\mathbf{SCpx}[S_{fl}\ c, s] \simeq \mathbf{Cov}[c, Flag\ s]$$

that is natural in $s \in Ob(\mathbf{SCpx})$ and $c \in Ob(\mathbf{Cov})$.

Consider first the function $f_{c,s} : \mathbf{SCpx}[S_{fl}\ c, s] \to \mathbf{Cov}[c, Flag\ s]$ that sends a simplicial map $h : S_{fl}\ c \to s$ to the function defined by its action on vertices. We will show that this function is a morphism in $\mathbf{Cov}[c, Flag\ s]$. Consider the set $S \in c$. For any $x_1, x_2 \in S$ the simplex $\{x_1, x_2\}$ must be in $S_{fl}\ c$. Since h is a simplicial map, this implies that the simplex $\{h(x_1), h(x_2)\}$ is in s, which implies that there must exist some set in $Flag\ s$ that contains $\{h(x_1), h(x_2)\}$ and therefore $h(S)$.

Next, consider the function $g_{c,s} : \mathbf{Cov}[c, Flag\ s] \to \mathbf{SCpx}[S_{fl}\ c, s]$ that sends a function $h : c \to Flag\ s$ to the map that applies h to the vertices of $S_{fl}\ c$. We will show that this is a simplicial map (Definition 2.51) in $\mathbf{SCpx}[S_{fl}\ c, s]$. First, note that since c is a nonnested flag cover $S_{fl}\ c$ is a flag complex. Since $S_{fl}\ c$ is a flag complex, we only need to show that this simplicial map preserves 1-simplices. Consider the 1-simplex $\{x_1, x_2\} \in S_{fl}\ c$. There must exist some set S in c such that $\{x_1, x_2\} \subseteq S$, which implies that there exists some set S' in $Flag\ s$ such that $\{h(x_1), h(x_2)\} \subseteq S'$. Then we can conclude that the simplex $\{h(x_1), h(x_2)\} \in s$.

Since these maps do not modify the action of a function over vertices, it is clear that they are inverses and are both natural in c and s. □

Intuitively, S_{fl} and $Flag$ describe a way to transform between simplicial complexes and nonnested flag covers. However, while flagification may be the most information preserving way to form a nonnested flag cover from a simplicial complex, it is not the only functor from **SCpx** to **Cov**. We can also form a nonnested flag cover from the connected components of a simplicial complex:

Proposition 6.10. *Consider the map* $\pi_0 : \mathbf{SCpx} \to \mathbf{Cov}$ *that acts as the identity on morphisms and sends* S_X *to the tuple* $(X, \mathcal{C}_X)$ *where* $\mathcal{C}_X$ *is the set of connected components of* S_X*. The map* π_0 *is a functor.*

Proof. Since π_0 acts as the identity on morphisms, identity and composition are trivially preserved. Next, note that for any S_X the tuple $\pi_0 S_X = (X, \mathcal{C}_X)$ is a partition of X and therefore a nonnested flag cover of X.

Next, consider any simplicial map $f : S_X \to S_Y$ in **SCpx** and pair of points $\{x_1, x_n\}$ that is contained by a set in $\pi_0 S_X$. There must exist a sequence of points $x_1, x_2, \cdots, x_n$ in the underlying set of S_X such that each $\{x_i, x_{i+1}\}$ is a simplex in S_X. Therefore for each $\{f(x_i), f(x_{i+1})\}$ in the sequence of points $f(x_1), f(x_2), \cdots, f(x_n)$ in the underlying set of S_Y it must be that $\{f(x_i), f(x_{i+1})\}$ is also a simplex in S_Y. This implies that $\{f(x_1), f(x_n)\}$ is also contained by a set in $\pi_0 S_Y$. Therefore $\pi_0 f$ is a morphism in **Cov**. □

6.1.3.2 Fuzzy Flagification

We can generalize the relationships between **Cov** and **SCpx** that we explored in Section 6.1.3.1 to relationships between **FCov** (Definition 2.71) and **FSCpx** (Definition 2.60). In particular, we can build on Definition 6.7 and Definition 6.8 to define the functors:

$$(S_{fl} \circ \_) : \mathbf{FCov} \to \mathbf{FSCpx}$$
$$(Flag \circ \_) : \mathbf{FSCpx} \to \mathbf{FCov}$$

We can put these functors together to form the following functor on fuzzy simplicial complexes:

Definition 6.11. *The functor FlagCpx converts any fuzzy simplicial complex into a fuzzy simplicial flag complex.*

$$FlagCpx : \mathbf{FSCpx} \to \mathbf{FSCpx}$$
$$FlagCpx = (S_{fl} \circ \_) \circ (Flag \circ \_)$$

6.1.4 The Realization and Singular Set Functors

Spivak (2012) and McInnes et al. (2018) adapt the realization and singular set functors from algebraic topology to form the finite realization and finite singular set functors that transform between fuzzy simplicial sets and finite ubermetric spaces. In this subsection we further adapt these functors to transform between fuzzy simplicial complexes and finite ubermetric spaces.

To start, consider the following functor:

Proposition 6.12. *Consider the map* $Pair : \mathbf{UMet} \to \mathbf{FSCpx}$ *that sends the ubermetric space* $(X, d_X) \in \mathbf{UMet}$ *to the fuzzy simplicial complex* $F_X : (0, 1]^{op} \to \mathbf{SCpx}$ *where for* $a \in (0, 1]^{op}$*,* $F_X(a)$ *is a simplicial complex with underlying set* X *whose* 1*-simplices are the pairs* $\{x_1, x_2\} \subseteq X$ *such that:*

$$d_X(x_1, x_2) \leq -\log(a)$$

$F_X(a)$ *has no* n*-simplices for* $n > 1$*. For any nonexpansive map*

$$h : (X, d_X) \to (Y, d_Y)$$

Pair maps h to the natural transformation whose component at each a is h. Pair is a functor.

Proof. *Pair* trivially preserves identity and composition, so we just need to show that for any nonexpansive map

$$h : (X, d_X) \to (Y, d_Y)$$

and $a \in (0, 1]$, the function h is a simplicial map from $Pair(X, d_X)(a)$ to $Pair(Y, d_Y)(a)$. Since neither $Pair(X, d_X)(a)$ nor $Pair(Y, d_Y)(a)$ contain any n-simplices where $n > 1$, we simply need to demonstrate that h preserves 1-simplices. Now suppose $\{x_1, x_2\}$ is a simplex in $Pair(X, d_X)(a)$. Then by the definition of *Pair* we have:

$$d_X(x_1, x_2) \leq -\log(a)$$

which by the nonexpansiveness of h implies that:

$$d_Y(h(x_1), h(x_2)) \leq -\log(a)$$

so $\{h(x_1), h(x_2)\}$ is a simplex in $Pair(Y, d_Y)(a)$ □

We use $-\log$ because it is a monotonically decreasing function from $(0, 1]$ to $[0, \infty)$. That is, if $d_X(x_1, x_2) = 0$, then the strength of the simplex $\{x_1, x_2\}$ in the fuzzy simplicial complex $Pair\ (X, d_X)$ is 1, whereas when $d_X(x_1, x_2)$ approaches ∞, the strength of the simplex $\{x_1, x_2\}$ approaches 0.

We can now use *Pair* and *FlagCpx* (Definition 6.11) to define an adaptation of the finite singular set functor (Spivak, 2012; McInnes et al., 2018) that maps ubermetric spaces to fuzzy simplicial complexes:

Definition 6.13. *The functor FinSing maps ubermetric spaces to fuzzy simplicial complexes as follows:*

$$FinSing : \mathbf{UMet} \to \mathbf{FSCpx}$$
$$FinSing = FlagCpx \circ Pair$$

For $a \in (0, 1]^{\text{op}}$, $FinSing(X, d_X)(a)$ is a flag complex in which the set:

$$S = \{x_1, x_2, \cdots, x_n\} \subseteq X$$

forms a simplex in $FinSing(X, d_X)(a)$ if all pairwise distances between points in the set are less than $-\log(a)$. This is the $-\log(a)$-Vietoris-Rips Complex of (X, d_X) (Definition 2.54).

We can now define an adaptation of the realization functor that maps fuzzy simplicial complexes to ubermetric spaces.

Definition 6.14. *Given a fuzzy simplicial complex $F_X : (0, 1]^{op} \to \mathbf{SCpx}$ in $\mathbf{FSCpx}$ with underlying set X the strength graph $\mathcal{G}_{F_X}$ of F_X is the graph with underlying set X and an edge of length:*

$$\inf\{-\log(a) \mid a \in (0, 1], \{x_1, x_2\} \in F_X(a)\}$$

between each pair $x_1, x_2 \in X$.

Proposition 6.15. *Given a fuzzy simplicial complex $F_X : (0,1]^{op} \to \mathbf{SCpx}$ in $\mathbf{FSCpx}$ with underlying set X we can define a functor $FinReal : \mathbf{FSCpx} \to \mathbf{UMet}$ that sends the fuzzy simplicial complex $F_X : (0,1]^{op} \to \mathbf{SCpx}$ with underlying set X to the ubermetric space (X, d_X) where for any points $x_1, x_2 \in X$ the distance $d_X(x_1, x_2)$ is the length of the shortest path from x_1 to x_2 in the strength graph $\mathcal{G}_{F_X}$ of F_X. On morphisms, $FinReal$ sends a natural transformation $\mu : F_X \to F_Y$ to the function determined by μ's actions on the underlying set of F_X.*

Proof. To start, note that $FinReal$ trivially preserves the identity and composition since natural transformations compose along their components.

Next, we can show for any $F_X \in \mathbf{FSCpx}$ with underlying set X the tuple $(X, d_X) = FinReal(F_X)$ is an ubermetric space. First, d_X is trivially reflexive. Second, for any $x \in X$ the length of the path from x to x is 0 so $d_X(x, x) = 0$. Next, for any $x_1, x_2, x_3 \in X$ note that we can always construct a path in the strength graph $\mathcal{G}_{F_X}$ of F_X from x_1 to x_3 by concatenating the paths from x_1 to x_2 and from x_2 to x_3. Therefore we can conclude that

$$d_X(x_1, x_3) \leq d_X(x_1, x_2) + d_X(x_2, x_3)$$

and (X, d_X) satisfies the triangle inequality.

Finally, we will show that if $(X, d_X) = FinReal(F_X)$ and $(Y, d_Y) = FinReal(F_Y)$ then the map

$$f : (X, d_X) \to (Y, d_Y)$$

determined by μ's actions on the underlying set of F_X is nonexpansive. Consider any $x_1 \in X, x_n \in X$ such that $d_X(x_1, x_n) \leq \infty$ and suppose that $x_1, x_2, \cdots, x_n$ is the shortest path from x_1 to x_n in the strength graph $\mathcal{G}_{F_X}$ of F_X. For any edge of length $-\log(a)$ from x_i to x_{i+1} in this path the simplex $\{x_i, x_{i+1}\}$ must be contained in $F_X(a)$. Since f is a simplicial map from $F_X(a)$ to $F_Y(a)$, it must be that the simplex $\{f(x_i), f(x_{i+1})\}$ is contained in $F_Y(a)$. This implies that the edge connecting $f(x_i)$ and $f(x_{i+1})$ in $\mathcal{G}_{F_Y}$ has length no greater than $-\log(a)$, which implies that the shortest path from $f(x_1)$ to $f(x_n)$ in $\mathcal{G}_{F_Y}$ has length no greater than $d_X(x_1, x_n)$. Therefore f is nonexpansive. □

Intuitively, $FinReal\ F_X$ is an ubermetric space where the distance between the points x_1, x_2 is determined by the length of the path from x_1 to x_2 in a graph we construct from F_X.

Proposition 6.16. *There exists an adjunction $FinReal \dashv FinSing$.*

Proof. We will show that there exists an isomorphism:

$$\mathbf{UMet}[FinReal\ c, m] \simeq \mathbf{FSCpx}[c, FinSing\ m]$$

that is natural in $m \in Ob(\mathbf{UMet})$ and $c \in Ob(\mathbf{FSCpx})$.

Consider first the function:

$$f_{c,m} : \mathbf{UMet}[FinReal\ c, m] \to \mathbf{FSCpx}[c, FinSing\ m]$$

that sends a nonexpansive map $h : FinReal\ c \to m$ to the map that applies h to the vertices of c. We will show that this is a natural transformation in $\mathbf{FSCpx}[c, FinSing\ m]$

whose components are simplicial maps. Since $FinSing\ m$ is a flag complex, we only need to show that this map preserves 1-simplices at each $a \in (0,1]^{\text{op}}$. For any $a \in (0,1]^{\text{op}}$ suppose there exists a 1-simplex $\{x_1, x_2\}$ in $c(a)$. Then the distance between x_1 and x_2 in $FinReal\ c$ must be no greater than $-\log(a)$. Since h is nonexpansive, this implies that the distance between $h(x_1)$ and $h(x_2)$ in m must be no greater than $-\log(a)$ as well. Since the n-simplices in $(FinSing\ (X, d_X))(a)$ are the n-element sets of points in X with pairwise distance $\leq -\log(a)$, we can therefore conclude that $\{h(x_1), h(x_2)\}$ is a simplex in $(FinSing\ m)(a)$. This implies that h is a natural transformation in $\mathbf{FSCpx}[c, FinSing\ m]$ whose components are simplicial maps.

Next, consider the function:

$$g_{c,m} : \mathbf{FSCpx}[c, FinSing\ m] \to \mathbf{UMet}[FinReal\ c, m]$$

that sends a natural transformation $h : c \to FinSing\ m$ whose components are simplicial maps to the function that is defined by the action of h on the underlying set of $FinReal\ c$. We will show that this is a nonexpansive map in $\mathbf{UMet}[FinReal\ c, m]$. Write $FinReal\ c = (X, d_X)$ and consider any $x_1, x_n \in X$ where $d_X(x_1, x_n) \leq \infty$. By the definition of $FinReal$ there exists some sequence $x_1, x_2, \cdots, x_n \in X$ such that:

$$d_X(x_1, x_n) = \sum_{i=1}^{i=n} \inf\{-\log(a) \ \mid \ a \in (0,1], \{x_i, x_{i+1}\} \in c(a)\}$$

Since for each $a \in (0,1]^{\text{op}}$ the map h is a simplicial map from $c(a)$ to $(FinSing\ m)(a)$, for each pair x_i, x_{i+1} in this sequence we have that:

$$\inf\{-\log(a) \ \mid \ a \in (0,1], \{h(x_i), h(x_{i+1})\} \in (FinSing\ m)(a)\} \leq \inf\{-\log(a) \ \mid \ a \in (0,1], \{x_i, x_{i+1}\} \in c(a)\}$$

Therefore the shortest path from x_1 to x_n in the strength graph $\mathcal{G}_{FinSing\ m}$ of $FinSing\ m$ must have length less than $d_X(x_1, x_n)$. Now by the definition of $FinSing$ the shortest path from x_1 to x_n in $\mathcal{G}_{FinSing\ m}$ must be the edge connecting x_1 to x_n. Therefore the 1-simplex $\{x_1, x_n\}$ in $FinSing\ m$ must have strength no less than a where $-\log(a) = d_X(x_1, x_n)$ and therefore the distance between x_1 and x_n in m is no greater than $d_X(x_1, x_n)$. This implies that h is nonexpansive.

It is clear that both of these functions are natural in c and s and that they are inverses. □

6.1.5 Maximal Linkage and Single Linkage

Recall that the single linkage functor $\mathcal{SL} : \mathbf{UMet} \to \mathbf{FCov}$ (Definition 3.11) maps the ubermetric space (X, d_X) to the hierarchical cover of X such that the points $x_1, x_n \in X$ lie in the same cluster with strength at least a if there exists a sequence of points:

$$x_1, x_2, \cdots, x_{n-1}, x_n$$

such that the largest distance between any adjacent points in the sequence is no larger than $-\log(a)$.

Recall also that the maximal linkage functor $\mathcal{ML} : \mathbf{UMet} \to \mathbf{FCov}$ (Definition 3.14) maps the ubermetric space (X, d_X) to the hierarchical cover of X such that the points

$x_1, x_2 \in X$ lie in the same cluster with strength at least a if the distance between them is no larger than $-\log(a)$.

Given an ubermetric space $(X, d_X) \in \mathbf{UMet}$, we can generate the single linkage clustering and maximal linkage clustering by first forming the Vietoris-Rips complex (Definition 2.54) with the $FinSing$ functor and then taking the connected components and maximal simplices of this complex respectively. That is, we can write:

$$\mathcal{ML} = (Flag \circ \_) \circ FinSing$$
$$\mathcal{SL} = (\pi_0 \circ \_) \circ FinSing$$

The functors $(Flag \circ \_)$ (Definition 6.8) and $(\pi_0 \circ \_)$ (Proposition 6.10) respectively map the functor $F_X : (0,1]^{\mathrm{op}} \to \mathbf{SCpx}$ in $\mathbf{FSCpx}$ to:

$$Flag \circ F_X : (0,1]^{\mathrm{op}} \to \mathbf{Cov} \qquad \text{and} \qquad \pi_0 \circ F_X : (0,1]^{\mathrm{op}} \to \mathbf{Cov}$$

in $\mathbf{FCov}$.

Proposition 6.17. *$\mathcal{ML}$ and $\mathcal{SL}$ are non-trivial overlapping hierarchical $\mathbf{UMet}$-clustering functors.*

Proof. First, we note that $\mathcal{ML}$ and $\mathcal{SL}$ are overlapping hierarchical $\mathbf{UMet}$-clustering functors. Given an ubermetric space $(X, d_X) \in \mathbf{UMet}$, the underlying set of $FinSing\ (X, d_X)$ is X. Since both $(Flag \circ \_)$ and $(\pi_0 \circ \_)$ map a fuzzy simplicial complex with underlying set X to a fuzzy nonnested flag cover of X, $\mathcal{ML}(\_)(a)$ and $\mathcal{SL}(\_)(a)$ commute with the forgetful functor.

Next, $\mathcal{ML}$ and $\mathcal{SL}$ are non-trivial since for $a \in (0,1]^{\mathrm{op}}$ the flat overlapping $\mathbf{UMet}$ clustering functors $\mathcal{ML}(\_)(a)$ and $\mathcal{SL}(\_)(a)$ have the clustering parameter $-\log(a)$. □

Single linkage always generates a partition of X, whereas maximal linkage does not.

Both single and maximal linkage use $-\log$ to convert from strengths in $\mathbf{FPart}$ to distances in $\mathbf{UMet}$. We can generalize this as follows:

Definition 6.18. *Given a subcategory $\mathbf{D}$ of $\mathbf{UMet}$, a non-trivial overlapping hierarchical $\mathbf{D}$-clustering functor $H : \mathbf{D} \to \mathbf{FCov}$ is in standard form if for all $\epsilon, \delta \in \mathbb{R}_{\geq 0}$ and ubermetric spaces $(X, d_X) \in \mathbf{UMet}$ we have:*

$$H(X, d_X + \epsilon)(e^{-\delta}) \simeq H(X, d_X)(e^{-\delta+\epsilon})$$

Proposition 6.19. *$\mathcal{SL}$ and $\mathcal{ML}$ are in standard form.*

Proof. First, note that for $\epsilon, \delta \in \mathbb{R}_{\geq 0}, (X, d_X) \in \mathbf{UMet}$ we have that

$$\mathcal{ML}(X, d_X + \epsilon)(e^{-\delta})$$

will group together all points $x_i, x_j \in X$ where:

$$d_X(x_i, x_j) + \epsilon \leq -\log(e^{-\delta}) = \delta$$

and

$$\mathcal{ML}(X, d_X)(e^{-\delta+\epsilon})$$

will group together all points $x_i, x_j \in X$ where:

$$d_X(x_i, x_j) \leq -\log(e^{-\delta+\epsilon}) = \delta - \epsilon$$

and so:

$$\mathcal{ML}(X, d_X + \epsilon)(e^{-\delta}) \simeq \mathcal{ML}(X, d_X)(e^{-\delta+\epsilon})$$

Next, note that

$$\mathcal{SL}(X, d_X + \epsilon)(e^{-\delta})$$

will group together all points $x_1, x_n \in X$ where there exists a chain of points $x_1, x_2, \cdots, x_n \in X$ such that:

$$d_X(x_i, x_{i+1}) + \epsilon \leq -\log(e^{-\delta}) = \delta$$

and

$$\mathcal{SL}(X, d_X)(e^{-\delta+\epsilon})$$

will group together all points $x_1, x_n \in X$ where there exists a chain of points $x_1, x_2, \cdots, x_n \in X$ such that:

$$d_X(x_i, x_{i+1}) \leq -\log(e^{-\delta+\epsilon}) = \delta - \epsilon$$

and so:

$$\mathcal{SL}(X, d_X + \epsilon)(e^{-\delta}) \simeq \mathcal{SL}(X, d_X)(e^{-\delta+\epsilon})$$

□

Unlike with single linkage it is possible to reconstruct an ubermetric space from its hierarchical maximal linkage clustering:

Proposition 6.20. *There exists an adjunction* $FinReal \circ (S_{fl} \circ \_) \dashv \mathcal{ML}$.

Proof. Since:

$$\mathcal{ML} = (Flag \circ \_) \circ FinSing$$

we can express this as:

$$FinReal \circ (S_{fl} \circ \_) \dashv (Flag \circ \_) \circ FinSing$$

which holds by the composition of adjunctions since $FinReal \dashv FinSing$ by Proposition 6.16 and $(S_{fl} \circ \_) \dashv (Flag \circ \_)$ by Proposition 6.9. □

Intuitively, single linkage and maximal linkage clustering lie on two ends of a spectrum of clustering refinement. Any other non-trivial hierarchical clustering functor lies between them. Formally, we can make the following claim, which is inspired by Theorem 8 in (Culbertson et al., 2016):

Proposition 6.21. *Suppose:*

$$H : \mathbf{UMet} \to \mathbf{FCov}$$

is a non-trivial hierarchical clustering functor such that for all $a \in (0, 1]$, the functor:

$$H(\_)(a) : \mathbf{UMet} \to \mathbf{Cov}$$

has clustering parameter $\delta_{H,a}$. Next, define the following functor:

$$W_H : (0, 1]^{op} \to (0, 1]^{op}$$
$$W_H(a) = e^{-\delta_{H,a}}$$

Note that:

$$-\log(W_H(a)) = -\log(e^{-\delta_{H,a}}) = \delta_{H,a}$$

Then there exist natural transformations with identity functions as components from:

$$\mathcal{ML}(\_)(W_H(\_)) : \mathbf{UMet} \to \mathbf{FCov}$$

to H and from H to:

$$\mathcal{SL}(\_)(W_H(\_)) : \mathbf{UMet} \to \mathbf{FCov}$$

Therefore H lies between $\mathcal{ML}(\_)(W_H(\_))$ and $\mathcal{SL}(\_)(W_H(\_))$ along any sequence $F_1 \to F_2 \to \cdots \to F_n$ of clustering functors and natural transformations with identity functions as components.

Proof. First, we construct a natural transformation:

$$\mu_{\mathcal{ML}} : \mathcal{ML}(\_)(W_H(\_)) \to H$$

by demonstrating that for all $(X, d_X) \in \mathbf{UMet}$ and all $a \in (0, 1]$, the identity function on X is a morphism in **Cov** from $\mathcal{ML}(X, d_X)(W_H(a))$ to $H(X, d_X)(a)$. Consider the set:

$$\{x_1, x_2, \cdots, x_n\} \in \mathcal{ML}(X, d_X)(W_H(a))$$

The maximum distance between any two points in this set must be less than $\delta_{H,a}$. Therefore, for any x_i, x_j in this collection, there exists a morphism in **UMet** from $\Lambda^1_{\delta_{H,a}}$ to (X, d_X) that sends the two elements of $\Lambda^1_{\delta_{H,a}}$ to x_i and x_j respectively. Since H is a functor and $H(\Lambda^1_{\delta_{H,a}})$ is a single two-point cluster, this implies that there is a set in the nonnested flag cover $H(X, d_X)(a)$ that contains both x_i and x_j. Since this logic holds for all pairs of points

$$x_i, x_j \in \{x_1, x_2, \cdots, x_n\}$$

and $H(X, d_X)(a)$ is a flag cover, there must exist a set $S \in H(X, d_X)(a)$ such that:

$$\{x_1, x_2, \cdots, x_n\} \subseteq S$$

which implies that the identity map on X is a morphism in **Cov** from $\mathcal{ML}(X, d_X)(W_H(a))$ to $H(X, d_X)(a)$.

Next, we construct a natural transformation:

$$\mu_{\mathcal{SL}} : H \to \mathcal{SL}(\_)(W_H(\_))$$

by demonstrating that for all $a \in (0, 1]$, the identity function on X is a morphism in **Cov** from $H(X, d_X)(a)$ to $\mathcal{SL}(X, d_X)(W_H(a))$. Suppose there exist $x_1, x_2 \in X$ such that there is no $S \in \mathcal{SL}(X, d_X)(W_H(a))$ where $\{x_1, x_2\} \subseteq S$. Suppose also that ϵ is the smallest distance between any pair of points $x_1', x_2' \in X$ where x_1' shares a cluster with x_1 in $\mathcal{SL}(X, d_X)(W_H(a))$ and x_2' shares a cluster with x_2 in $\mathcal{SL}(X, d_X)(W_H(a))$. Note that $\epsilon > \delta_{H,a}$.

Now write $\Lambda_\epsilon^1 = \{\lambda_1, \lambda_2\}$ and consider the function

$$f : (X, d_X) \to \Lambda_\epsilon^1$$

that maps all x_1' that share a cluster in $\mathcal{SL}(X, d_X)(W_H(a))$ with x_1 to λ_1, all x_2' that share a cluster in $\mathcal{SL}(X, d_X)(W_H(a))$ with x_2 to λ_2, and all other points to λ_2. This function is nonexpansive and therefore a morphism in **UMet** since there exists no pair of points $x_1', x_2' \in X$ such that $f(x_1') \neq f(x_2')$ and $d_X(x_1', x_2') < \epsilon$.

Since H is a functor, it must map f to a morphism in **FCov**. Now since H has a clustering parameter of $\delta_{H,a}$ and $\epsilon > \delta_{H,a}$, the functor H maps Λ_ϵ^1 to two singleton clusters. Therefore there cannot be any S in $H(X, d_X)(a)$ such that $\{x_1, x_2\} \subseteq S$. We can therefore conclude that the identity function on X is a morphism in **Cov** from $H(X, d_X)(a)$ to $\mathcal{SL}(X, d_X)(W_H(a))$. □

6.2 Flattening Multiparameter Hierarchical Clustering Functors

In Section 3.3.1.6 we discussed some ways in which hierarchical clustering algorithms can be extended to multiple parameters. Since applying a multiparameter hierarchical clustering algorithm with different hyperparameter values may produce different partitions, we can view such algorithms as mapping a finite metric space (X, d_X) to a function from the hyperparameter space to the space of partitions of X.

In this section we will characterize and study multiparameter hierarchical clustering algorithms with partially ordered hyperparameter spaces. This perspective allows us to guarantee that the clustering algorithms we study preserve both nonexpansive maps between metric spaces and the ordering of the hyperparameter space. We focus entirely on nonoverlapping clustering for simplicity.

As we discussed in Section 3.3.1.6, it can be difficult to work with multiscale clusterings in practice. For this reason it can be helpful to flatten the clustering, or convert it to a single partition. Many popular clustering algorithms, such as HDBSCAN (Campello et al.) and ToMATo (Chazal et al., 2013), include a flattening step in which the most important clusters of a dataset are those which exist at multiple scales. In this section we will introduce a general strategy that operates based on this intuition and evaluate it on real data. Concretely:

- We generalize nonoverlapping hierarchical **D**-clustering functors to multiple hyperparameters and demonstrate in Proposition 6.27 how the robust single linkage algorithm fits into this framework.

- We describe an algorithm for flattening a multiparameter hierarchical clustering and show in Table 4 that this algorithm can outperform choosing the optimal hyperparameter value on real data.
- We introduce a Bayesian update algorithm in Definition 6.33 for learning a distribution over clustering hyperparameters from data.
- We show in Theorem 6.35 that the composition of this Bayesian update algorithm and this flattening algorithm is consistent.

In this section we will exclusively consider functors out of **Met** rather than **UMet** for clarity.

6.2.1 Multiparameter Hierarchical Clustering

Many nonoverlapping flat clustering algorithms are configured by a hyperparameter vector that governs their behavior. Two examples are the δ-single linkage functor (Definition 3.8) and (γ, δ)-robust single linkage functor (Definition 3.10). Both functors map a metric space $(X, d_X) \in \mathbf{Met}$ to the set of connected components of a simplicial complex constructed from (X, d_X) and the functor hyperparameters.

More generally, in the case that a nonoverlapping flat clustering algorithm's hyperparameter vector is an element of a partial order O we can represent the output of such an algorithm with a functor $O \to \mathbf{Part}$.

Definition 6.22. *Given a partial order O, an O-partition of the set X is a functor:*

$$F_X : O \to \mathbf{Part}$$

such that for all $a \in O$ we have that $(U \circ F_X)(a) = X$ where $U : \mathbf{Part} \to \mathbf{Set}$ is the forgetful functor. That is, for any morphism $a' \geq a$ in O, the function $F_X(a' \geq a)$ is the identity function $x \mapsto x$ on X.

We can form a category of O-partitions as follows:

Definition 6.23. *Given a partial order O, the objects in the category $\mathbf{Part}^{\overline{O}}$ are O-partitions (functors). The morphisms in $\mathbf{Part}^{\overline{O}}$ are natural transformations.*

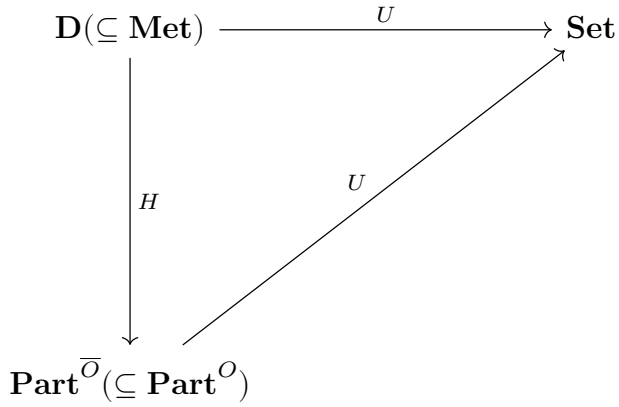
$\mathbf{Part}^{\overline{O}}$ is a subcategory of the category $\mathbf{Part}^{O}$ of functors from O to **Part** and natural transformations between them. Note also that **FPart** (Definition 2.72) is equivalent to $\mathbf{Part}^{\overline{(0,1]^{op}}}$. We can further define the following:

Definition 6.24. *Given an O-partition F_X of the set X we call X the underlying set of F_X.*

Note that there also exists a forgetful functor $U : \mathbf{Part}^{\overline{O}} \to \mathbf{Set}$ that maps $F_X : O \to \mathbf{Part}$ to its underlying set X and that any natural transformation in $\mathbf{Part}^{\overline{O}}$ from the O-partition $F_X : O \to \mathbf{Part}$ to the O-partition $F_Y : O \to \mathbf{Part}$ of the sets X and Y is fully specified by a function $f : X \to Y$.

Definition 6.25. *Given a partial order O and a subcategory $\mathbf{D}$ of $\mathbf{Met}$, a nonoverlapping O-hierarchical $\mathbf{D}$-clustering functor is a functor $H : \mathbf{D} \to \mathbf{Part}^{\overline{O}}$ that commutes with the forgetful functors from $\mathbf{D}$ and $\mathbf{Part}^{\overline{O}}$ into $\mathbf{Set}$. In the case that $\mathbf{D}$ is unspecified we simply call H a nonoverlapping O-hierarchical clustering functor.*

That is, for any nonoverlapping O-hierarchical $\mathbf{D}$-clustering functor $H : \mathbf{D} \to \mathbf{Part}^{\overline{O}}$ the following diagram commutes:



For example, hierarchical single linkage $\mathcal{SL} : \mathbf{Met} \to \mathbf{FPart}$ (Definition 3.11) is a nonoverlapping $(0,1]^{\text{op}}$-hierarchical $\mathbf{Met}$-clustering functor.

One convenient property for an O-hierarchical $\mathbf{D}$-clustering functor is the following:

Definition 6.26. *The O-partition F_X of the set X is collection complete if for each $x \in X$ there exists $a \in O$ such that $\{x\} \in F_X(a)$.*

For convenience, we will also call the nonoverlapping O-hierarchical $\mathbf{D}$-clustering functor H collection complete if for each $(X, d_X) \in \mathbf{D}$ the O-partition

$$H(X, d_X)(\_) : O \to \mathbf{Part}$$

is collection complete.

This definition is similar in spirit to Definition 6.6, except that this is a condition on the behavior of $H(X, d_X)(\_) : O \to \mathbf{Part}$ rather than the behavior of $H(\_)(a) : \mathbf{D} \to \mathbf{Part}$.

Recall the (γ, δ)-robust single linkage algorithm (Definition 3.10). Since the density estimation parameter γ in this algorithm behaves similarly to the scale parameter δ over which single linkage clustering varies, a natural question is whether we can express this algorithm in this framework. This would both simplify the theoretical presentation of the algorithm and enable us to take advantage of the structure exposed by varying the scale parameter.

Proposition 6.27. *The robust single linkage functor:*

$$\mathcal{SL}_{\mathcal{R}} : \mathbf{Met}_{bij} \to \mathbf{Part}^{\overline{(0,1]^{op} \times (0,1]^{op}}}$$

is a collection complete $(0,1]^{op} \times (0,1]^{op}$-hierarchical $\mathbf{Met}_{bij}$-clustering functor where $\mathcal{SL}_{\mathcal{R}}(X, d_X)(a_1, a_2)$ is the partition of X in which the distinct points x_1, x_n are in the same cluster if and only if there exists some sequence of points:

$$x_1, x_2, \cdots, x_{n-1}, x_n$$

such that for all (x_i, x_{i+1}) in this sequence we have:

$$d_X^{a_1}(x_i, x_{i+1}) \leq -\log(a_2)$$

where:

$$d_X^{a_1}(x_i, x_{i+1}) = max(d_X(x_i, x_{i+1}), \mu_{X_{a_1}}(x_i), \mu_{X_{a_1}}(x_{i+1}))$$

and $\mu_{X_{a_1}}(x_i)$ *is the distance from* x_i *to its* $\lfloor a_1 * |X| \rfloor$*th nearest neighbor.*

For any nonexpansive map

$$f : (X, d_X) \to (Y, d_Y)$$

in $\mathbf{Met}_{bij}$ *the functor* $\mathcal{SL}_{\mathcal{R}}$ *maps* f *to the natural transformation whose component at each* a *is* f*.*

Proof. To start, note that $\mathcal{SL}_{\mathcal{R}}$ trivially preserves identity and composition.

Next, we will show that for any $(X, d_X) \in \mathbf{Met}_{bij}$ the map:

$$\mathcal{SL}_{\mathcal{R}}(X, d_X) : (0,1]^{\text{op}} \times (0,1]^{\text{op}} \to \mathbf{Part}$$

is an object in $\mathbf{Part}^{\overline{(0,1]^{\text{op}} \times (0,1]^{\text{op}}}}$. $\mathcal{SL}_{\mathcal{R}}(X, d_X)$ trivially commutes with the forgetful functor so we simply need to show that it is a functor. Consider some:

$$(a_1', a_2') \geq (a_1, a_2) \in (0,1]^{\text{op}} \times (0,1]^{\text{op}}$$

pair $x_1, x_n \in X$, and any sequence $x_1, x_2, \cdots, x_n$ in X. For all $i \leq n$ we have that

$$d_X^{a_1}(x_i, x_{i+1}) \leq d_X^{a_1'}(x_i, x_{i+1})$$

and that:

$$-\log(a_2') \leq -\log(a_2)$$

so $d_X^{a_1'}(x_i, x_{i+1}) \leq -\log(a_2')$ implies that $d_X^{a_1}(x_i, x_{i+1}) \leq -\log(a_2)$. Therefore the identity function on X is a morphism in **Part** from $\mathcal{SL}_{\mathcal{R}}(X, d_X)(a_1', a_2')$ to $\mathcal{SL}_{\mathcal{R}}(X, d_X)(a_1, a_2)$. This implies that $\mathcal{SL}_{\mathcal{R}}(X, d_X) : (0,1]^{\text{op}} \times (0,1]^{\text{op}} \to \mathbf{Part}$ is a functor.

Next, we will show that for any bijective nonexpansive map

$$f : (X, d_X) \to (Y, d_Y)$$

in $\mathbf{Met}_{bij}$ and any $(a_1, a_2) \in (0,1]^{\text{op}} \times (0,1]^{\text{op}}$ the map f is a consistent map from $\mathcal{SL}_{\mathcal{R}}(X, d_X)(a_1, a_2)$ to $\mathcal{SL}_{\mathcal{R}}(Y, d_Y)(a_1, a_2)$. Consider some pair of distinct point $x_1, x_n \in X$ that are in the same cluster in $\mathcal{SL}_{\mathcal{R}}(X, d_X)(a_1, a_2)$. There must exist some sequence $x_1, x_2, \cdots, x_n$ in X where for all (x_i, x_{i+1}) in this sequence we have:

$$d_X^{a_1}(x_i, x_{i+1}) \leq -\log(a_2)$$

Now since f is nonexpansive we have that:

$$d_Y(f(x_i), f(x_{i+1})) \leq d_X(x_i, x_{i+1})$$

Furthermore, since f is bijective it must be that $|X| = |Y|$ and therefore:

$$\mu_{Y_{a_1}}(f(x_i)) \leq \mu_{X_{a_1}}(x_i)$$
$$\mu_{Y_{a_1}}(f(x_{i+1})) \leq \mu_{X_{a_1}}(x_{i+1})$$

Together this implies that:

$$d_Y^{a_1}(f(x_i), f(x_{i+1})) \leq d_X^{a_1}(x_i, x_{i+1})$$

Therefore for all $(f(x_i), f(x_{i+1}))$ in the sequence $f(x_1), f(x_2), \cdots, f(x_n)$ in Y it must be that:

$$d_Y^{a_1}(f(x_i), f(x_{i+1})) \leq -\log(a_2)$$

and therefore $(f(x_1), f(x_n))$ are in the same cluster in $\mathcal{SL}_{\mathcal{R}}(Y, d_Y)(a_1, a_2)$. We can therefore conclude that f is a consistent map from $\mathcal{SL}_{\mathcal{R}}(X, d_X)(a_1, a_2)$ to $\mathcal{SL}_{\mathcal{R}}(Y, d_Y)(a_1, a_2)$.

Finally, we will show that $\mathcal{SL}_{\mathcal{R}}$ is collection complete. For any $(X, d_X) \in \mathbf{Met}_{bij}$ consider the O-partition:

$$\mathcal{SL}_{\mathcal{R}}(X, d_X) : (0, 1]^{\text{op}} \times (0, 1]^{\text{op}} \to \mathbf{Part}$$

and consider some $x \in X$. By the definition of a metric space it must be that for all $x' \neq x$ in X, $d_X(x, x') > 0$ and therefore:

$$d_X(x, x') > -\log(1)$$

Therefore for any choice of $a_1 \in (0, 1]^{\text{op}}$ we have that $\{x\} \in H(X, d_X)(a_1, 1)$ so $\mathcal{SL}_{\mathcal{R}}$ is collection complete.

□

Although $\mathcal{SL}_{\mathcal{R}}$ is a nonoverlapping $(0, 1]^{\text{op}} \times (0, 1]^{\text{op}}$-hierarchical $\mathbf{Met}_{bij}$-clustering functor, it is not a nonoverlapping $(0, 1]^{\text{op}} \times (0, 1]^{\text{op}}$-hierarchical $\mathbf{Met}$-clustering functor because it includes a k-nearest neighbor computation that is sensitive to $|X|$.

6.2.2 Multiparameter Flattening

Consider the following structure:

Definition 6.28. *The partition collection of the O-partition $F_X : O \to \mathbf{Part}$ of the set X is the set S_X of all subsets $S \subseteq X$ such that there exists some $a \in O$ where $S \in F_X(a)$.*

In order to flatten an O-partition F_X of X we simply need to choose a partition of X from its partition collection S_X. One way to do this is to weigh the elements of S_X according to a model of the importance of different regions of O. In this section we will only consider O that are Borel measurable, so we can represent this model with a probability measure μ over $(O, \mathcal{B}(O))$. This probabilistic interpretation will be useful in Section 6.2.3 when we learn this model from data.

We can then view the flattening algorithm as choosing a partition $\mathbb{P}_X$ of X where $\mathbb{P}_X \subseteq S_X$ and $\mathbb{P}_X$ maximizes the μ-expectation of the function that maps $a \in O$ to the number of sets $s_X \in \mathbb{P}_X$ that are also in $F_X(a)$. Formally, the algorithm chooses $\mathbb{P}_X$ to maximize:

$$\int_{a \in O} |\{s_X \mid s_X \in \mathbb{P}_X, s_X \in F_X(a)\}| \, d\mu$$

If μ is uniform this is similar to the topologically motivated HDBSCAN by McInnes and Healy (2017).

We can reduce this selection problem to a binary integer program.

Definition 6.29. *An binary integer program is a tuple:*

$$(n, m, c, A, u)$$

where $n, m \in \mathbb{N}$, c is an m-element real valued vector, u is an n-element real valued vector and A is an $n \times m$ real valued matrix.

The solution set of the binary integer program (n, m, c, A, u) is the set S of all m-element $\{0, 1\}$-valued vectors v that maximize $c^T v$ subject to:

$$Av \leq u$$

Proposition 6.30. *Suppose we have a probability measure μ over O, an O-partition $F_X : O \to \mathbf{Part}_{bij}$ of the set X, and a total order on the partition collection S_X of F_X such that we can write the elements of S_X (subsets of X) with the notation:*

$$S_X = \left\{ s_{1_X}, s_{2_X}, \cdots, s_{n_X} \right\}$$

The μ-binary integer program of F_X is the tuple:

$$(|S_X|, |S_X|, c, A, u)$$

where c, u are $|S_X|$-element real valued vectors and A is a real valued $|S_X| \times |S_X|$ matrix where:

$$u[i] = |S_X|$$

$$c[i] = \int_{\{a \ | \ s_{i_X} \in F_X(a)\}} d\mu$$

$$A[i, j] = \begin{cases} |S_X| & i = j \\ 1 & i \neq j, \ s_{i_X} \cap s_{j_X} \neq \emptyset \\ 0 & else \end{cases}$$

If F_X is collection complete there exists some v in the solution set of the μ-binary integer program of F_X for which the set $S_X(v) \subseteq S_X$ of all $s_{i_X} \in S_X$ where $v[i] = 1$ is the partition of X that maximizes the following function across all partitions of X:

$$\int_{a \in O} \left| \{ s_{i_X} \ | \ s_{i_X} \in S_X(v), s_{i_X} \in F_X(a) \} \right| d\mu$$

Proof. Suppose

$$(|S_X|, |S_X|, c, A, u)$$

is the μ-binary integer program of F_X. For any partition $\mathbb{P}_X$ of X define the m-element $\{0, 1\}$-valued vector v to have 1 in position i for each set $s_{i_X} \in S_X$ that $\mathbb{P}_X$ contains.

Note that $S_X(v) = \mathbb{P}_X$. We can therefore write:

$$
\begin{aligned}
& c^T v \\
= \quad & \{\text{Dot product}\} \\
& \sum_{s_{i_X} \in \mathbb{P}_X} c_i \\
= \quad & \{\text{Definition of } c_i\} \\
& \sum_{s_{i_X} \in \mathbb{P}_X} \int_{\{a \mid s_{i_X} \in F_X(a)\}} d\mu \\
= \quad & \{\text{Rearrange integral}\} \\
& \int_{a \in O} \left| \{ s_{i_X} \mid s_{i_X} \in \mathbb{P}_X, s_{i_X} \in F_X(a) \} \right| d\mu
\end{aligned}
$$

Next, consider any row $A[i]$ in A. Since this row has $|S_X|$ elements, the value $|S_X|$ in position i, the value 1 in each position j where $i \neq j,\ s_{i_X} \cap s_{j_X} \neq \emptyset$ and 0 in all other positions it must be that:

$$A[i]^T v > |S_X|$$

if and only if $v[i] = 1$ and there exists some j where $i \neq j,\ s_{i_X} \cap s_{j_X} \neq \emptyset$ and $v[j] = 1$. However, since the sets in $\mathbb{P}_X$ must be nonoverlapping this cannot be the case. We can therefore conclude that:

$$Av \leq u$$

Therefore for any v^* in the solution set of the μ-binary integer program of F_X we have:

$$c^T v \leq c^T v^*$$

and so:

$$
\begin{aligned}
& \int_{a \in O} \left| \{ s_{i_X} \mid s_{i_X} \in \mathbb{P}_X, s_{i_X} \in F_X(a) \} \right| d\mu \leq \\
& \int_{a \in O} \left| \{ s_{i_X} \mid s_{i_X} \in S_X(v^*), s_{i_X} \in F_X(a) \} \right| d\mu
\end{aligned}
$$

We now show that there exists some v^* in the solution set of the μ-binary integer program of F_X such that $S_X(v^*)$ is a partition of X.

First, note that this solution set is not empty, since the vector of all zeros satisfies the constraint $Av \leq u$.

Next, consider any v^* in this solution set and any distinct $s_{i_X}, s_{j_X} \in S_X$ where $s_{i_X} \cap s_{j_X} \neq \emptyset$. The ith row of A will have $|S_X|$ in position i and 1 in position j. This implies that if $v^*[i] = 1$ then $A[i]^T v^* \leq |S_X|$ if and only if $v^*[j] = 0$. Therefore it cannot be that $s_{i_X}, s_{j_X} \in S_X(v^*)$ and we can conclude that any pair of sets in $S_X(v^*)$ must have an empty intersection.

Next, consider any v^* in the solution set of the μ-binary integer program of F_X and suppose there exists some $x \in X$ that is not contained by any set in $S_X(v^*)$. Since F_X is collection complete, the set $s_{i_X} = \{x\}$ must be in S_X and:

$$\int_{\{a \mid s_{i_X} \in F_X(a)\}} d\mu \geq 0$$

Therefore we can set $v^*[i] = 1$ without decreasing $c^T v^*$ and still satisfy:

$$Av^* \leq u$$

We can repeat this process for each $x \in X$ that is not contained by any set in $S_X(v^*)$ to form a vector v^{**} in the solution set of the μ-binary integer program of F_X such that $S_X(v^{**})$ is a partition of X.

□

For example, if μ is uniform then the connected components of the Vietoris-Rips filtration of (X, d_X) that have the largest differences between their birth and death times will be a maximal solution to the μ-binary integer program of $\mathcal{SL}(X, d_X)$.

6.2.2.1 Implementing Multiparameter Flattening

Given a nonoverlapping O-hierarchical clustering functor H and a distribution μ over O we can use Monte Carlo integration to implement the following algorithm:

procedure MULTIPARAMETERFLATTENING($H, \mu, (X, d_X), n$)
    Initialize an empty list L of sets
    Repeat n times:
        Sample the hyperparameter a according to μ
        Add each set in $H(X, d_X)(a)$ to L
    Define S_X to be the list of unique elements of L
    Define the $|S_X|$-element vector c where c_i is the number of times the set s_{i_X} appears in L
    Set u, A as in Proposition 6.30
    Solve the binary integer program $(|S_X|, |S_X|, c, A, u)$

We include an example of this procedure on GitHub (Shiebler, 2020a) that builds on the HDBSCAN implementation by McInnes and Healy (2017). Recall that HDBSCAN (Campello et al.) is a density-based clustering algorithm that uses a similar strategy to robust single linkage but performs better in practice.

In Table 4 we demonstrate that applying this procedure and solving the resulting binary integer program can perform better than choosing an optimal parameter value.

Frequency that Multiparameter Flattening Outperforms Best α on the Fashion MNIST Dataset	Frequency that Multiparameter Flattening Outperforms Best α on the 20 Newsgroups Dataset
0.912 ($\pm$0.025)	0.768 ($\pm$0.038)

Table 4: *We compare the performance of applying multiparameter flattening to HDBSCAN (Campello et al.; McInnes and Healy, 2017) with simply running HDBSCAN with the value of the distance scaling parameter α that achieves the best Adjusted Rand Score. In this experiment, we select 500 bootstrap samples of the Fashion MNIST (Xiao et al., 2017) and 20 Newsgroups (Lang, 1995) datasets, apply both clustering strategies to each bootstrap sample, and compare the clustering produced with the ground truth categorization. We use the scikit-learn implementation of the Adjusted Rand Score (Pedregosa et al., 2011) to perform this comparison. We then compute the frequency with which multiparameter flattening performs better (has a higher Adjusted Rand Score with ground truth) than choosing the optimal value of α. The win rates and two standard error confidence bounds from the 500 experiments are shown. We see that the multiparameter flattening procedure performs consistently better, which suggests that it may be a good option for unsupervised learning applications such as data visualization or preprocessing. The code to run these experiments is on GitHub (Shiebler, 2021a).*

6.2.3 Multiparameter Flattening with Supervision

One of the most important components in the multiparameter flattening algorithm is the choice of distribution μ over O. For example, if $O = (0,1]^{\text{op}}$ and μ is the Dirac distribution at a then $\mathcal{SL}(X, d_X)(a)$ will be a solution to the μ-binary integer program of $\mathcal{SL}(X, d_X)$.

We can leverage the importance of μ to enable the flattening procedure to learn from labeled data. That is, given a finite metric space (X, d_X), a partition $\mathbb{P}_X$ of X (the labels), and a nonoverlapping O-hierarchical clustering functor H, we can use Bayes rule to assign higher weight to the regions of O in which $H(X, d_X)$ and $\mathbb{P}_X$ are more similar.

Definition 6.31. *Suppose that $\mathbf{P}_X$ is the finite set of all partitions of the finite set X and that $(O, \mathcal{B}(O), \mu_0)$ is the probability space constructed from the Borel measurable space $(O, \mathcal{B}(O)$ and the uniform measure $\mu_0 : \mathcal{B}(O) \to [0,1]$. Suppose also that we have a map:*

$$\gamma_X(\_ \mid \_) : \mathbf{P}_X \times O \to [0,1]$$

such that for each $a \in O$ the map:

$$\gamma_X(\_ \mid a) : \mathbf{P}_X \to [0,1]$$

is a probability mass function over the finite set $\mathbf{P}_X$ and that for each $\mathbb{P}_X \in \mathbf{P}_X$ the map:

$$\gamma_X(\mathbb{P}_X \mid \_) : O \to [0,1]$$

is Lebesgue integrable.

Then given a partition $\mathbb{P}_X$ of X (the labels) we can use Bayes rule to define a probability measure:

$$\mu_{\mathbb{P}_X} : \mathcal{B}(O) \to [0,1]$$
$$\mu_{\mathbb{P}_X}(\sigma) = \frac{\int_{a \in \sigma} \gamma_X(\mathbb{P}_X \mid a)\ d\mu_0}{\int_{a \in O} \gamma_X(\mathbb{P}_X \mid a)\ d\mu_0}$$

We call $\mu_{\mathbb{P}_X}$ the Bayes measure of γ_X at $\mathbb{P}_X$.

There are several ways that we can construct γ_X from a nonoverlapping O-hierarchical clustering functor H. Intuitively, we want $\gamma_X(\mathbb{P}_X \mid a)$ to be large when $H(X, d_X)(a)$ and $\mathbb{P}_X$ are similar. The simplest choice would be:

$$\gamma_X(\mathbb{P}_X \mid a) = \begin{cases} 1 & H(X, d_X)(a) = \mathbb{P}_X \\ 0 & else \end{cases}$$

but a more robust strategy would be to build on the Rand score (Definition 2.75), which measures how well two partitions agree on each pair of distinct points.

Proposition 6.32. *Suppose (X, d_X) is a finite metric space and $H(X, d_X)$ is a nonoverlapping O-hierarchical clustering functor. Then we can define the $H(X, d_X)$-Rand mass function over the set $\mathbf{P}_X$ of all partitions of X to be:*

$$\gamma_X(\_ \mid \_) : \mathbf{P}_X \times O \to [0, 1]$$
$$\gamma_X(\mathbb{P}_X \mid a) = \frac{RI(\mathbb{P}_X, H(X, d_X)(a))}{\sum_{\mathbb{P}'_X \in \mathbf{P}_X} RI(\mathbb{P}'_X, H(X, d_X)(a))}$$

where RI is the Rand score (Definition 2.75).

For each $a \in O$ the function:

$$\gamma_X(\_ \mid a) : \mathbf{P}_X \to [0, 1]$$

is a probability mass function over the finite set $\mathbf{P}_X$. Similarly, for each $\mathbb{P}_X \in \mathbf{P}_X$ the map:

$$\gamma_X(\mathbb{P}_X \mid \_) : O \to [0, 1]$$

is Lebesgue integrable.

Proof. To start, note that for any $a \in O$ the map:

$$\gamma_X(\_ \mid a) : \mathbf{P}_X \to [0, 1]$$

is a probability mass function since by definition for all $a \in O$ we have that:

$$\sum_{\mathbb{P}_X \in \mathbf{P}_X} \gamma_X(\mathbb{P}_X \mid a) = \frac{\sum_{\mathbb{P}_X \in \mathbf{P}_X} RI(\mathbb{P}_X, H(X, d_X)(a))}{\sum_{\mathbb{P}'_X \in \mathbf{P}_X} RI(\mathbb{P}'_X, H(X, d_X)(a))} = 1$$

Next, note that since the set $\mathbf{P}_X$ is finite the function:

$$H(X, d_X) : O \to \mathbf{P}_X$$

can only take on a finite number of values, which implies that the function:

$$\gamma_X(\mathbb{P}_X \mid \_) : O \to [0, 1]$$
$$\gamma_X(\mathbb{P}_X \mid a) = \frac{RI(\mathbb{P}_X, H(X, d_X)(a))}{\sum_{\mathbb{P}'_X \in \mathbf{P}_X} RI(\mathbb{P}'_X, H(X, d_X)(a))}$$

can also only take on a finite number of values in $[0, 1]$, so this function is continuous almost everywhere and therefore Lebesgue integrable. □

Intuitively, the Bayes measure (Definition 6.31) of the $H(X, d_X)$-Rand mass function γ_X at $\mathbb{P}_X$ places higher weight on the $a \in O$ where $H(X, d_X)(a)$ is closer to $\mathbb{P}_X$. In Figure 15 we explore the distribution of samples from the Bayes measure of the $\mathcal{SL}(X, d_X)$-Rand mass function where (X, d_X) is a real-world dataset.

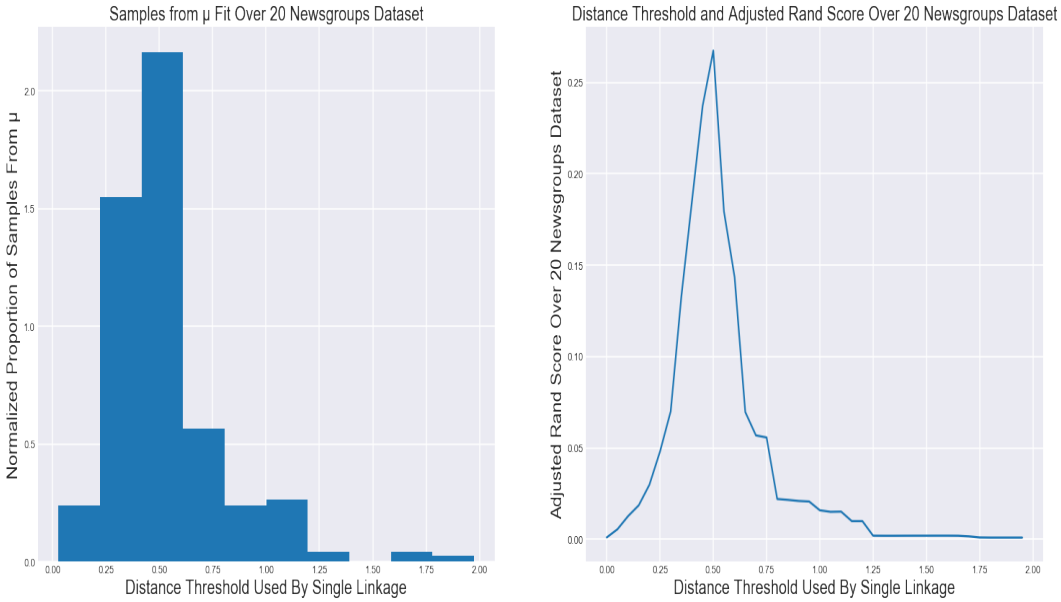


Figure 15: *Suppose (X, d_X) is the 20-Newsgroups dataset and $\mathbb{P}_X$ is the partition of X determined by the class labels on X. Suppose also that μ is the Bayes measure of the $\mathcal{SL}(X, d_X)$-Rand mass function at $\mathbb{P}_X$. In this case samples from μ are distance thresholds for single linkage. We see that these samples tend to be drawn from the region where the adjusted Rand score (Definition 2.76) between $\mathcal{SL}(X, d_X)$ and $\mathbb{P}_X$ is highest. The code to generate this figure can be found on GitHub (Shiebler, 2020b).*

We can now build on these constructions and use Bayesian updating to learn a posterior distribution over O from multiple observations (Bloom and Orloff, 2014) :

Definition 6.33. *Suppose we have a nonoverlapping O-hierarchical clustering functor H and a sequence of tuples*

$$S = (X_1, d_{X_1}, \mathbb{P}_{X_1}), (X_2, d_{X_2}, \mathbb{P}_{X_2}), \cdots$$

where each $(X_i, d_{X_i}) \in S$ is a finite metric space and each $\mathbb{P}_{X_i}$ is a partition of X_i.

Then the i-Rand-Bayes measure of (S, H) is:

$$\mu_i(\sigma) : \mathcal{B}(O) \to [0, 1] \tag{2}$$

$$\mu_i(\sigma) = \frac{\int_{a \in \sigma} \gamma_{X_i}(\mathbb{P}_{X_i} \mid a) \, d\mu_{i-1}}{\int_{a \in O} \gamma_{X_i}(\mathbb{P}_{X_i} \mid a) \, d\mu_{i-1}} \tag{3}$$

where γ_{X_i} is the $H(X_i, d_{X_i})$-Rand mass function (Proposition 6.32) and $\mu_0 : \mathcal{B}(O) \to [0, 1]$ is the uniform measure.

Next, we describe how we can use a target parameter to construct a sampling distribution from which we can draw the samples in S:

Proposition 6.34. *Given $d \in \mathbb{N}$ suppose we have a compact region $R \subseteq [0,1]^d$ and a nonoverlapping O-hierarchical clustering functor H. For any $k \in \mathbb{N}$ define $\lambda_{R^k} : \mathcal{B}(R^k) \to [0,1]$ to be the uniform probability measure on R^k.*

Now suppose P_R is the set of pairs $(X, \mathbb{P}_X)$ of finite k-element subsets $X \subset R$ and partitions $\mathbb{P}_X$ of X and assume that the function:

$$\gamma\_(\_ \mid a) : P_R \to [0,1]$$

that maps $(X, \mathbb{P}_X) \in P_R$ to the value of the $H(X, d_X)$-Rand mass function γ_X (Proposition 6.32) at $\mathbb{P}_X$ is Lebesgue integrable.

Then for each $a \in O$ we can define the following probability measure on $(P_R, \mathcal{B}(P_R))$, which we call the (H, a)-Rand probability measure.

$$\eta_a : \mathcal{B}(P_R) \to [0,1]$$
$$\eta_a(\sigma) = \int_{(X,\mathbb{P}_X)\in\sigma} \gamma_X(\mathbb{P}_X \mid a)\ \lambda_{R^k}$$

Proof. For any $a \in O$ by the definition of Lebesgue integration we have that η_a is trivially countably additive and $\eta_a(\emptyset) = 0$. Furthermore we have:

$$\begin{aligned}
&\eta_a(P_R)\\
=\quad &\{\text{Definition of } \eta\}\\
&\int_{(X,\mathbb{P}_X)\in P_R} \gamma_X(\mathbb{P}_X \mid a)\ \lambda_{R^k}\\
=\quad &\{\text{Rewrite } P_R\}\\
&\int_{X\in R^k} \sum_{\mathbb{P}_X\in\mathbf{P}_X} \gamma_X(\mathbb{P}_X \mid a)\ \lambda_{R^k}\\
=\quad &\{\text{Definition of } \gamma_X\}\\
&\int_{X\in R^k} \frac{\sum_{\mathbb{P}_X\in\mathbf{P}_X} RI(\mathbb{P}_X, H(X,d_X)(a))}{\sum_{\mathbb{P}'_X\in\mathbf{P}_X} RI(\mathbb{P}'_X, H(X,d_X)(a))}\ \lambda_{R^k}\\
=\quad &\{\text{Simplify}\}\\
&\int_{X\in R^k} \lambda_{R^k}\\
=\quad &\{\text{Compute the integral}\}\\
&1
\end{aligned}$$

□

Finally, we will show that the composition of the Bayesian update algorithm (Definition 6.33) and the multiparameter flattening algorithm (Proposition 6.30) is consistent. When we choose a target parameter $a^* \in O$ and draw a sequence of samples S according to η_{a^*}, the optimal solution to the μ_n-binary integer program of $H(X, d_X)$ will almost surely be $H(X, d_X)(a^*)$.

Theorem 6.35. *Given $d \in \mathbb{N}$ suppose we have a compact region $R \subseteq [0,1]^d$ and a nonoverlapping O-hierarchical clustering functor H. For any $k \in \mathbb{N}$ define $\lambda_{R^k} : \mathcal{B}(R^k) \to [0,1]$ to be the uniform probability measure on R^k and $\mu_0 : \mathcal{B}(O) \to [0,1]$ to be the uniform probability measure on O. Furthermore, define d_R to be Euclidean distance in $[0,1]^d$.*

Suppose that the conditions in Proposition 6.34 are satisfied, and that:

- *H is R-smooth: for any k-element $X \subset R$ there exists a subset $S \subset O$ such that $\mu_0(S) = 0$ and for any $a \in O - S$ there exists a neighborhood B_a of a where for $a' \in B_a$ we have that:*

$$H(X, d_R)(a) = H(X, d_R)(a')$$

- *H is R-identifiable: For any distinct $a, a' \in O$ we have that:*

$$\eta_a \neq \eta_{a'}$$

where η_a is the (H, a)-Rand probability measure (Proposition 6.34).

Now suppose we sample $a_* \in O$ according to μ_0 and for each $i > 0$ sample $(X_i, \mathbb{P}_{X_i})$ according to the (H, a^*)-Rand probability measure η_{a^*}. We can therefore define the sequence:

$$S = (X_1, d_R, \mathbb{P}_{X_1}), (X_2, d_R, \mathbb{P}_{X_2}), \cdots$$

and set μ_i to be the i-Rand-Bayes measure of (S, H) (Definition 6.33).

Then for any k-element $X \subset R$ there almost surely exists some m such that for $n \geq m$, the partition $H(X, d_R)(a_*)$ is an optimal solution to the μ_n-binary integer program of $H(X, d_R)$ (Proposition 6.30).

Proof. We will use the formulation of Doob's theorem (Doob, 1949) by Miller (2018) to prove this. To start, we show that Assumption 2.1 in (Miller, 2018) holds:

- Since H is R-smooth, it must be that for any k-element $X \subset R$ and partition $\mathbb{P}_X$ the function:

$$\gamma_X(\mathbb{P}_X \mid \_) : O \to [0, 1]$$

is continuous almost everywhere, and therefore measurable. This implies that for any $\sigma \in \mathcal{B}(P_R)$ the function:

$$\eta_\_(\sigma) : O \to [0, 1]$$

is measurable as well.

- Since H is R-identifiable, we have that:

$$a \neq a' \implies \eta_a \neq \eta_{a'}$$

We can now use Theorem 2.4 in (Miller, 2018). For any ball B_{a_*} centered at a_*, and $\epsilon > 0$, there μ_0-almost surely, η_a-almost surely exists some m such that for $n \geq m$ we have that $\mu_n(B_{a_*}) \geq 1 - \epsilon$. Therefore, for any k-element $X \subset R$ no solution to the μ_n-binary integer program of $H(X, d_R)$ can be improved by including sets that only exist in partitions of X formed from $H(X, d_R)(a')$ where $a' \notin B_{a_*}$. Since H is R-smooth, this implies that the partition $H(X, d_R)(a_*)$ is almost surely an optimal solution to the μ_n-binary integer program of $H(X, d_R)$.

□

6.2.4 Thoughts on Flattening Multiparameter Hierarchical Clustering Algorithms

The multiparameter flattening procedure is very general, and can support a broad range of multiparameter hierarchical clustering algorithms. Because this procedure is consistent with the Bayesian update algorithm, we can define the following supervised clustering algorithm from a multiparameter hierarchical clustering algorithm:

- Given a set of training samples (tuples of finite metric spaces and partitions), apply the Bayesian update algorithm to learn a distribution over the multiparameter hierarchical clustering parameter space.
- Given a finite metric space (X, d_X) apply the multiparameter flattening procedure with this distribution to derive a partition of X.

However, one issue with the multiparameter flattening procedure is that it reduces to a binary integer program, which can be very slow to solve. This makes it difficult to use this algorithm for large scale problems.

6.3 Functorial Manifold Learning

In this section we develop a functorial perspective on manifold learning. We first characterize manifold learning algorithms as functors that map metric spaces to optimization objectives and factor through hierachical clustering functors. We then use this characterization to prove refinement bounds on manifold learning loss functions and construct a hierarchy of manifold learning algorithms based on their invariants. We express several popular manifold learning algorithms as functors at different levels of this hierarchy, including metric multidimensional scaling, isomap, and UMAP. Next, we use interleaving distance to study the stability of a broad class of manifold learning algorithms. We present bounds on how closely the embeddings these algorithms produce from noisy data approximate the embeddings they would learn from noiseless data. Finally, we use our framework to derive a set of novel manifold learning algorithms, which we experimentally demonstrate are competitive with the state of the art.

6.3.1 Background and Overview

Suppose we have a metric space $(X, d_X) \in \mathbf{Met}$ that we assume has been sampled from some larger space $\mathbf{X}$ according to some probability measure $\mu_{\mathbf{X}}$ over $\mathbf{X}$. Manifold learning algorithms like isomap (Tenenbaum et al., 2000), metric multidimensional scaling (Abdi, 2007), and UMAP (McInnes et al., 2018) construct $\mathbb{R}^a$-embeddings for the points in X, which we interpret as coordinates for the support of $\mu_{\mathbf{X}}$. These techniques are based on the assumption that this support can be well-approximated with a manifold. For example, a common assumption is that d_X is a noisy approximation of geodesic distance on the support of $\mu_{\mathbf{X}}$, and that smaller distances between points are be more likely to be reliable than larger distances. Manifold learning algorithms like Laplacian eigenmaps, which ignore all pairs of points with distances greater than some threshold δ, exploit this assumption (Belkin and Niyogi, 2003).

When $\mathbf{X}$ is $\mathbb{R}^b$ for $b > a$ we can interpret a manifold learning algorithm as a dimensionality reduction algorithm that maps vectors in $\mathbb{R}^b$ to vectors in $\mathbb{R}^a$. However, it is critical for manifold learning algorithms to be able to generate different outputs based

on the choice of the set $X \subset \mathbf{X}$. If we attempt to construct a single $f : \mathbb{R}^b \to \mathbb{R}^a$ and use it to reduce the dimensionality of the points in all sets $X \subset \mathbb{R}^b$ we can run into serious problems. For example, Landweber et al. (2016) show that any continuous map $\mathbb{R}^b \to \mathbb{R}^a$ may collapse arbitrarily distant regions to a single point.

We can use functoriality to explore two aspects of manifold learning algorithms:

- **Equivariance**: A manifold learning algorithm is equivariant under a transformation if applying that transformation to its inputs results in an corresponding transformation of its outputs. Understanding the equivariance of a transform lets us understand how it will behave on new types of data.

- **Stability**: The stability of a manifold learning algorithm captures how well the embeddings it generates on noisy data approximate the embeddings it would generate on noiseless data. Understanding the stability of a transform helps us predict its performance on real-world applications.

In Section 6.3.2, we demonstrate that manifold learning algorithms can be expressed as optimization problems defined on top of overlapping hierarchical clusterings. That is, we can express these algorithms in terms of the composition of:

- A strategy for clustering points at different distance scales

- An order preserving transformation from a clustering of points to a loss function

We formalize this relationship in terms of the composition of functors between categories of metric spaces, overlapping hierarchical clusterings, and optimization problems. This allows us to formally extend clustering theory into manifold learning theory.

In Section 6.3.3 we build on clustering theory to demonstrate that every manifold learning objective lies on a spectrum based on the criterion by which embeddings are penalized for being too close together or too far apart. In Section 6.3.4 we introduce a hierarchy of manifold learning algorithms and categorize algorithms based on the dataset transformations under which they are equivariant. In Section 6.3.5 we provide several examples of this categorization. We show that UMAP is equivariant under isometries and both isomap and metric multidimensional scaling are equivariant under nonexpansive maps. Identifying these equivariants is helpful for identifying the circumstances under which we would expect our algorithms to generalize. For example, while adding points to a dataset will modify the isomap objective in a predictable way, this will completely disrupt the UMAP objective. This is caused by the fact that UMAP uses a local distance rescaling procedure that is density-sensitive, and is therefore not equivariant under injective or surjective nonexpansive maps.

In Section 6.3.6 we use interleaving distance to study the stability of a broad class of manifold learning algorithms. Due to the importance of this topic, many other authors have researched the stability properties of manifold learning algorithms. For example, Bailey (2012) explores adaptations of principal components analysis (PCA) to noisy data, and Gerber et al. (2007) demonstrate that Laplacian eigenmaps has nicer stability properties than isomap. However, we believe that this is the first work that uses interleaving distance to formalize a manifold learning stability property. We present novel bounds on how well the embeddings that algorithms in this class learn on noisy data approximate the embeddings they learn on noiseless data.

In Section 6.3.7 we build on this theory to describe a strategy for deriving novel algorithms from existing manifold learning algorithms. As an example we derive the

novel single linkage scaling algorithm, which projects samples in the same connected component of the Vietoris-Rips complex as close together as possible. We also present experimental results demonstrating the efficacy of this algorithm.

6.3.2 Manifold Learning

Definition 6.36. *Given a choice of $m \in \mathbb{N}$, an m-embedding of the finite set X is a function:*

$$\gamma : X \to \mathbb{R}^m$$

that assigns an m-element real valued vector to each element of X.

Given a choice of $m \in \mathbb{N}$ and a metric space $(X, d_X) \in \mathbf{Met}$, a manifold learning algorithm returns some $\gamma : X \to \mathbb{R}^m$. In this work we focus on manifold learning algorithms that operate by solving a special kind of optimization problem:

Definition 6.37. *An embedding optimization problem is a tuple $(X, m, \mathbb{l})$ where X is a finite set, m is a positive integer and $\mathbb{l}$ is a real valued function:*

$$\mathbb{l} : (X \to \mathbb{R}^m) \to \mathbb{R}$$

which we will call the loss function.

Solving an embedding optimization problem involves finding an m-embedding

$$\gamma : X \to \mathbb{R}^m$$

that minimizes its loss function:

Definition 6.38. *The solution set of the embedding optimization problem $(X, m, \mathbb{l})$ is the set of all $\gamma : X \to \mathbb{R}^m$ that minimize $\mathbb{l}(\gamma)$.*

The solution set of an embedding optimization problem may be empty if $\mathbb{l}$ does not have any global minima. We will particularly focus on the following subclass of embedding optimization problems:

Definition 6.39. *An pairwise embedding optimization problem is a tuple (X, m, l_X) where l_X is a real valued function:*

$$l_X : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R}$$

such that the tuple $(X, m, \mathbb{l})$ where

$$\mathbb{l}(\gamma) = \sum_{x_i, x_j \in X} l_X(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|)$$

is an embedding optimization problem.

Intuitively each term

$$l_X(x_i, x_j, \_) : \mathbb{R}_{\geq 0} \to \mathbb{R}$$

in a pairwise embedding optimization problem acts as a penalty on the distance between the embeddings of x_i and x_j. We can now define manifold learning problems in terms of pairwise embedding optimization problems:

Definition 6.40. *A manifold learning problem is a function that maps the metric space* $(X, d_X) \in \mathbf{Met}$ *to a pairwise embedding optimization problem* (X, m, l_X).

This definition does not include any specification of how the optimization problem is solved (gradient descent, reduction to an eigenproblem, etc).

An example manifold learning problem is metric multidimensional scaling:

Definition 6.41. *The metric multidimensional scaling manifold learning problem maps the metric space* $(X, d_X) \in \mathbf{Met}$ *to the pairwise embedding optimization problem* (X, m, l_X) *where:*

$$l_X : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R}$$
$$l_X(x_i, x_j, \delta) = (d_X(x_i, x_j) - \delta)^2$$

Intuitively, optimizing the metric multidimensional scaling objective for some (X, d_X) involves finding an m-embedding $\gamma : X \to \mathbb{R}^m$ that minimizes:

$$\sum_{x_i, x_j \in X} (d_X(x_i, x_j) - \|\gamma(x_i) - \gamma(x_j)\|)^2$$

That is, we need to find a $\gamma : X \to \mathbb{R}^m$ that minimizes the Frobenius norm of the difference between the Euclidean distance matrix of $\gamma(X)$ and the d_X distance matrix. In general we can solve this problem with gradient descent, but under certain conditions on (X, d_X) this problem reduces to an eigenproblem.

We will particularly focus on a special case of manifold learning problems:

Definition 6.42. *We say that a pair of pairwise embedding optimization problems*

$$(X, m, l_X), (Y, m, l_Y)$$

have isomorphic solutions when there exists a bijection $f : X \to Y$ *such that:*

- *for any solution* $\gamma_Y : Y \to \mathbb{R}^m$ *to* (Y, m, l_Y) *the map* $\gamma_Y \circ f : X \to \mathbb{R}^m$ *is a solution to* (X, m, l_X)
- *for any solution* $\gamma_X : X \to \mathbb{R}^m$ *to* (X, m, l_X) *the map* $\gamma_X \circ f^{-1} : Y \to \mathbb{R}^m$ *is a solution to* (Y, m, l_Y)

Definition 6.43. *A manifold learning problem is isometry invariant if it maps any pair of metric spaces* $(X, d_X), (Y, d_X) \in \mathbf{Met}$ *with an isometry between them to a pair of pairwise embedding optimization problems* $(X, m, l_X), (Y, m, l_Y)$ *with isomorphic solutions.*

Intuitively, isometry invariant manifold learning algorithms do not change their output based on the identity of the points in X. A particularly useful property of isometry invariant manifold learning problems is that they factor through hierarchical clustering algorithms.

Proposition 6.44. *An isometry invariant manifold learning problem* M *factors into the composite* $L \circ H$ *where* H *is an overlapping hierarchical clustering functor (Definition 6.5) and* L *is a function that maps the output of* H *to a pairwise embedding optimization problem, such that the images of* M *and* $L \circ H$ *on any metric space* $(X, d_X) \in \mathbf{Met}$ *have isomorphic solutions.*

Proof. Recall the maximal linkage overlapping hierarchical clustering functor $\mathcal{ML}$ (Definition 3.14) that maps the metric space $(X, d_X) \in \mathbf{Met}$ to the fuzzy nonnested flag cover $F_X : (0,1]^{\text{op}} \to \mathbf{Part}$ of X such that for $a \in (0,1]^{\text{op}}$ the cover $F_X(a)$ is the maximally linked sets of the relation R in which $x_i \, R \, x_j$ if:

$$d_X(x_i, x_j) \leq -\log(a)$$

Consider also the function *Real* that maps the fuzzy nonnested flag cover $F_X : (0,1]^{\text{op}} \to \mathbf{Part}$ of the set X to the metric space $(X, d'_X) \in \mathbf{Met}$ in which:

$$d'_X(x_i, x_j) = \inf\{-\log(a) \mid \exists S \in F_X(a), x_i, x_j \in S\}$$

It is easy to see that $\mathit{Real} \circ \mathcal{ML}$ is an isometry on metric spaces. Therefore, for any isometry invariant manifold learning problem M and metric space $(X, d_X) \in \mathbf{Met}$, the pairwise embedding optimization problems $M(X, d_X)$ and

$$((M \circ \mathit{Real}) \circ \mathcal{ML})(X, d_X)$$

have isomorphic solutions. □

Intuitively, Proposition 6.44 holds because manifold learning problems generate loss functions from a finite metric space by grouping points in the space together based on their distances.

In order to use this property to adapt clustering theorems into manifold learning theorems we will introduce a target category of pairwise embedding optimization problems and replace functions with functors from **Met** into this category. We restrict the domain of these functors to **Met** rather than **UMet** since it is difficult to meaningfully create an optimization objective from an ubermetric space in which the distance between two points may be infinite.

To start, suppose we construct a category in which objects are tuples (X, l_X) of a finite set X and a real valued function l_X with signature:

$$l_X : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R}$$

and morphisms from (X, l_X) to (Y, l_Y) are functions $f : X \to Y$ such that for $x_i, x_j \in X$ and $\delta \in \mathbb{R}_{\geq 0}$ we have:

$$l_X(x_i, x_j, \delta) \leq l_Y(f(x_i), f(x_j), \delta)$$

Given a choice of m we can view each object (X, l_X) in this category as a pairwise embedding optimization problem (X, m, l_X). However, this category is not quite expressive enough to serve as our target category for manifold learning functors. Recall the metric multidimensional scaling manifold learning problem, which maps the metric space $(X, d_X) \in \mathbf{Met}$ to the pairwise embedding optimization problem (X, m, l_X) where:

$$l_X(x_i, x_j, \delta) = (d_X(x_i, x_j) - \delta)^2$$

Since l_X does not vary monotonically with $d_X(x_i, x_j)$, it is clear that this manifold learning problem is not a functor from **Met**. However, we can express l_X as the sum of a term that increases monotonically in $d_X(x_i, x_j)$ and a term that decreases monotonically in $d_X(x_i, x_j)$:

$$l_X(\delta) = (d_X(x_i, x_j) - \delta)^2 = \left(\delta^2 + d_X(x_i, x_j)^2\right) - \left(2\delta d_X(x_i, x_j)\right)$$

We will see in Section 6.3.5 that the embedding optimization problems associated with many common manifold learning algorithms decompose similarly. We can build on this insight to create a new category of manifold learning loss functions:

Definition 6.45. *We define the category* **Loss** *to be the category in which objects are tuples* (X, c_X, e_X) *of a finite set* X *and functions:*

$$c_X : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R}$$
$$e_X : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R}$$

The morphisms from (X, c_X, e_X) *to* (Y, c_Y, e_Y) *in* **Loss** *are functions* $f : X \to Y$ *such that for* $x_i, x_j \in X$ *and* $\delta \in \mathbb{R}_{\geq 0}$ *we have:*

$$c_Y(f(x_i), f(x_j), \delta) \leq c_X(x_i, x_j, \delta)$$
$$e_X(x_i, x_j, \delta) \leq e_Y(f(x_i), f(x_j), \delta)$$

Given a choice of m, each object (X, c_X, e_X) in **Loss** corresponds to the pairwise embedding optimization problem (X, m, l_X) where:

$$l_X : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R}$$
$$l_X(x_i, x_j, \delta) = c_X(x_i, x_j, \delta) + e_X(x_i, x_j, \delta)$$

Now recall that the preorder category $(0, 1]^{\text{op}}$ has elements in the set $(0, 1]$ as objects with the order relation $\geq$ as morphisms. Similarly to our representation of overlapping hierarchical clustering functors as maps into a category **FCov** of functors $(0, 1]^{\text{op}} \to$ **Cov** (Definition 6.5), we will represent manifold learning algorithms as mapping into a category **FLoss** of functors $(0, 1]^{\text{op}} \to$ **Loss**:

Definition 6.46. *A fuzzy loss tuple is a functor* $F_X : (0, 1]^{op} \to$ **Loss** *that commutes with the forgetful functor* $U :$ **Loss** $\to$ **Set** *that maps* (X, c_X, e_X) *to* X*. That is, for any morphism* $a' \geq a$ *in* $(0, 1]^{op}$*, the function* $F_X(a' \geq a)$ *is the identity function* $x \mapsto x$ *on* X*. We call* X *the underlying set of* F_X*.*

We use the following notation to express a fuzzy loss tuple:

$$F_X(a) = (X, c_{F_X(a)}, e_{F_X(a)})$$

Definition 6.47. *The objects in the category* **FLoss** *are fuzzy loss tuples (Definition 6.46):*

$$F_X(a) = (X, c_{F_X(a)}, e_{F_X(a)})$$

such that for any $x_i, x_j \in X, \delta \in \mathbb{R}_{\geq 0}$ *the functions:*

$$c_{F_X(\_)}(x_i, x_j, \delta) : (0, 1]^{op} \to \mathbb{R}$$
$$e_{F_X(\_)}(x_i, x_j, \delta) : (0, 1]^{op} \to \mathbb{R}$$

are Lebesgue integrable. The morphisms in **FLoss** *are natural transformations.*

Given any object $F_X \in$ **FLoss** we can construct an embedding optimization problem $(X, m, \mathbb{l}_{F_X})$ where $\mathbb{l}_{F_X}$ is defined as follows:

Definition 6.48. *Given a functor $F_X \in \mathbf{FLoss}$ we call the following sum the F_X-loss:*

$$\mathbb{l}_{F_X} : (X \to \mathbb{R}^m) \to \mathbb{R}$$
$$\mathbb{l}_{F_X}(\gamma) = \sum_{x_i, x_j \in X} \int_{a \in (0,1]} c_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) + e_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|)\ da$$

We can now give our definition of a manifold learning functor:

Definition 6.49. *Given the subcategories $\mathbf{D} \subseteq \mathbf{Met}, \mathbf{D}' \subseteq \mathbf{FCov}$, the composition:*

$$L \circ H : \mathbf{D} \to \mathbf{FLoss}$$

forms a $\mathbf{D}$-manifold learning functor if $H : \mathbf{D} \to \mathbf{D}'$ is an overlapping hierarchical $\mathbf{D}$-clustering functor and $L : \mathbf{D}' \to \mathbf{FLoss}$ is a functor that maps a fuzzy nonnested flag cover with underlying set X to some $F_X \in \mathbf{FLoss}$ with underlying set X.

Intuitively, a manifold learning functor:

$$\mathbf{D} \xrightarrow{H} \mathbf{D}' \xrightarrow{L} \mathbf{FLoss}$$

factors through a hierarchical clustering functor and sends (X, d_X) to:

$$F_X : (0,1]^{\text{op}} \to \mathbf{Loss}$$
$$F_X(a) = (X, c_{F_X(a)}, e_{F_X(a)})$$

Given a choice of m each manifold learning functor M corresponds to a manifold learning problem that maps (X, d_X) to $(X, m, \mathbb{l}_{M(X,d_X)})$ where $\mathbb{l}_{M(X,d_X)}$ is the $M(X, d_X)$-loss (Definition 6.48).

We will primarily work with manifold learning functors in the following form:

Definition 6.50. *Given a manifold learning functor $M = L \circ H$, we say that M is in standard form if H is in standard form (Definition 6.18).*

Intuitively, a manifold learning functor in standard form uses $-\log$ to convert from strengths in **FLoss** to distances in **Met**, similar to our characterizations of single linkage (Definition 3.11) and maximal linkage (Definition 3.14).

6.3.3 A Spectrum of Manifold Learning Functors

In Proposition 6.21 we demonstrated that the hierarchical single linkage and maximal linkage clustering functors lie on two ends of a spectrum of clustering refinement. We can use functoriality and Definition 6.49 to adapt this to a theorem about manifold learning:

Theorem 6.51. *Suppose $L \circ H$ is a $\mathbf{Met}$-manifold learning functor such that H is a non-trivial overlapping hierarchical $\mathbf{Met}$-clustering functor (Definition 6.6) and for all $a \in (0,1]$, the functor:*

$$H(\_)(a) : \mathbf{Met} \to \mathbf{Cov}$$

has clustering parameter $\delta_{H,a}$. Then for any $(X, d_X) \in \mathbf{Met}, a \in (0,1]^{op}$ the identity map on X forms the $\mathbf{FLoss}$-morphisms:

$$(L \circ \mathcal{ML})(X, d_X)(e^{-\delta_{H,a}}) \xrightarrow{id_X} (L \circ H)(X, d_X)(a) \xrightarrow{id_X} (L \circ \mathcal{SL})(X, d_X)(e^{-\delta_{H,a}})$$

Proof. By Proposition 6.21, for any (X, d_X) the identity map on X forms the **FCov**-morphisms:

$$\mathcal{ML}(X, d_X)(W_H(\_)) \xrightarrow{id_X} H(X, d_X)(\_) \xrightarrow{id_X} \mathcal{SL}(X, d_X)(W_H(\_))$$

where $W_H(a) = e^{-\delta_{H,a}}$. The statement then holds by the functoriality of $L \circ H$ over **Met**. □

Intuitively, Proposition 6.51 states that every manifold learning functor maps (X, d_X) to a loss that is both no more interconnected than the loss that does not distinguish points within the same connected component of the Vietoris-Rips complex and no less interconnected than the loss that treats each pair of points independently.

There are many manifold learning functors that lie between these extremes. In particular, for any functor $L : \mathbf{FCov} \to \mathbf{Loss}$ and sequence of clustering functors:

$$\mathcal{ML} \to H_1 \to H_2 \to \cdots \to H_n \to \mathcal{SL}$$

whose outputs refine each other, we can apply functoriality to form a sequence of manifold learning functors:

$$L \circ \mathcal{ML} \to L \circ H_1 \to \cdots \to L \circ H_n \to L \circ \mathcal{SL}$$

For example, consider the family $\mathcal{L}_k$ of overlapping hierarchical clustering functors by Culbertson et al. (2016).

Definition 6.52. *For $k \in \mathbb{N}$, the functor $\mathcal{L}_k : \mathbf{Met} \to \mathbf{FCov}$ maps $(X, d_X)(a)$ to the maximal linked sets of the relation R_a where for any $x_1, x_k \in X$ we have $x_1 \, R_a \, x_k$ if there is a sequence:*

$$x_1, x_2 \cdots, x_{k-1}, x_k$$

in X where:

$$d_X(x_i, x_{i+1}) \leq -\log(a)$$

The functor $L \circ \mathcal{L}_k$ therefore maps (X, d_X) to a loss that distinguishes only between points whose shortest path in the Vietoris-Rips complex is longer than k. For $k > 1$ this loss is more interconnected than $L \circ \mathcal{ML}$ and less interconnected than $L \circ \mathcal{SL}$. This also suggests a recipe for generating new manifold learning algorithms:

- Identify the manifold learning problem associated with the algorithm and express it in the form $L \circ H$.
- Form the more interconnected functor $L \circ \mathcal{SL}$, the less interconnected functor $L \circ \mathcal{ML}$, or any of the functors along the spectrum $L \circ \mathcal{L}_k$.

In Section 6.3.7 we empirically demonstrate how this procedure can affect the performance and behavior of the learned embeddings.

6.3.4 Characterizing Manifold Learning Problems

Similarly to how Carlsson and Mémoli (2013) characterize clustering algorithms in terms of their functoriality over different subcategories of metric spaces, we can characterize manifold learning algorithms based on the subcategory $\mathbf{D} \subseteq \mathbf{Met}$ over which they are functorial.

We have already introduced the class of isometry invariant manifold learning problems. It is easy to see that any $\mathbf{Met}_{isom}$-manifold learning functor is isometry invariant.

Next, for many manifold learning algorithms the loss function decomposes into the sum of a term e that decreases as distances increase and a term c that increases as distances increase. We call these algorithms expansive-contractive. Any $\mathbf{Met}_{bij}$-manifold learning functor is expansive-contractive.

We can further break down this characterization:

Definition 6.53. *An expansive-contractive manifold learning problem is extensible if the expansive term e is nonpositive and the contractive term c is nonnegative.*

Proposition 6.54. *Any $\mathbf{Met}_{sur}$-manifold learning functor M that maps the one-point metric space $(\{*\}, 0)$ to some:*

$$F_* : (0,1]^{op} \to \mathbf{Loss}$$
$$F_* = (X, c_{F_*(a)}, e_{F_*(a)})$$

where $c_{F_*(a)}(*, *, \delta)$ is nonnegative and $e_{F_*(a)}(*, *, \delta)$ is nonpositive is extensible.

Proof. Consider the unique surjective nonexpansive map from $f : (X, d_X) \to (\{*\}, 0)$. Since M is a functor out of $\mathbf{Met}_{sur}$ for all $x_i, x_j \in X, \delta \in \mathbb{R}_{\geq 0}$, we have:

$$0 \leq c_{M(*)(a)}(*, *, \delta) \leq c_{M(X,d_X)(a)}(x_i, x_j, \delta)$$
$$e_{M(X,d_X)(a)}(x_i, x_j, \delta) \leq e_{M(*)(a)}(*, *, \delta) \leq 0$$

□

6.3.5 Examples

In this section we explore examples of manifold learning functors out of some of the categories we discussed in Section 6.3.4.

6.3.5.1 Metric Multidimensional Scaling (Met-Manifold Learning Functor)

The most straightforward strategy for learning embeddings is to minimize the difference between the pairwise distance matrix of the original space and the pairwise Euclidean distance matrix of the learned embeddings. Metric multidimensional scaling (Definition 6.41) does exactly this. Recall that given a metric space $(X, d_X) \in \mathbf{Met}$, the metric multidimensional scaling embedding optimization problem is $(X, m, \mathbb{l})$ where:

$$\mathbb{l}(\gamma) = \sum_{x_i, x_j \in X} (d_X(x_i, x_j) - \|\gamma(x_i) - \gamma(x_j)\|)^2$$

When the distance matrix of the metric space is double-centered (mean values of rows/-columns are zero) this is the same objective that principal components analysis (PCA) optimizes (Abdi and Williams, 2010).

We can now define the metric multidimensional scaling functor:

Proposition 6.55. *Consider the map:*

$$MDS : \mathbf{FCov} \to \mathbf{FLoss}$$

that maps the fuzzy nonnested flag cover $H : (0,1]^{op} \to \mathbf{Cov}$ *with underlying set* X *to* $F : (0,1]^{op} \to \mathbf{Loss}$ *where:*

$$F_X(a) = (X, c_{F_X(a)}, e_{F_X(a)})$$

and:

$$c_{F_X(a)} : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R} \qquad e_{F_X(a)} : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R}$$

$$c_{F_X(a)}(x_i, x_j, \delta) = \begin{cases} \delta^2 & \exists S \in H(a),\ x_i, x_j \in S \\ \delta^2 + 2\delta^2 \left(1/W(x_i, x_j) - 1/a\right) & else \end{cases}$$

$$e_{F_X(a)}(x_i, x_j, \delta) = \begin{cases} 0 & \exists S \in H(a),\ x_i, x_j \in S \\ \frac{2\log(W(x_i,x_j))}{W(x_i,x_j)} - \frac{2\log(a)}{a} & else \end{cases}$$

where:

$$W(x_i, x_j) = \sup_{\geq 0}\{a \mid a \in (0,1],\ \exists S \in H(a),\ x_i, x_j \in S\}$$

MDS is a functor, and $MDS \circ \mathcal{ML}$ *is a standard form* **Met***-manifold learning functor that maps the metric space* $(X, d_X) \in \mathbf{Met}$ *to the metric multidimensional scaling embedding optimization problem.*

Proof. First, we need to show that $MDS : \mathbf{FCov} \to \mathbf{FLoss}$ is a functor. Consider the fuzzy nonnested flag covers H_X and $H_{X'}$ in **FCov** with underlying sets X, X' respectively and the **FCov**-morphism between them defined by the function f. Suppose:

$$MDS(H_X)(a) = (X, c_{F_X(a)}, e_{F_X(a)})$$
$$MDS(H_{X'})(a) = (X', c_{F_{X'}(a)}, e_{F_{X'}(a)})$$

We must show that the function $f : X \to X'$ is a **Loss**-morphism from $(X, c_{F_X(a)}, e_{F_X(a)})$ to $(X', c_{F_{X'}(a)}, e_{F_{X'}(a)})$. Suppose:

$$W_{H_X}(x_i, x_j) = \sup_{\geq 0}\{a \mid a \in (0,1],\ \exists S \in H_X(a),\ x_i, x_j \in S\}$$
$$W_{H_{X'}}(x'_i, x'_j) = \sup_{\geq 0}\{a \mid a \in (0,1],\ \exists S \in H_{X'}(a),\ x'_i, x'_j \in S\}$$

For any:

$$a \in (0,1],\ x_i, x_j \in X,\ \exists S \in H_X(a),\ x_i, x_j \in S$$

by definition of f:

$$\exists S \in H_{X'}(a),\ f(x_i), f(x_j) \in S$$

and therefore:

$$W_{H_X}(x_i, x_j) \leq W_{H_{X'}}(f(x_i), f(x_j))$$

which further implies:

$$\delta^2 + 2\delta^2 \left(1/W_{H_{X'}}(f(x_i), f(x_j)) - 1/a\right) \leq \delta^2 + 2\delta^2 \left(1/W_{H_X}(x_i, x_j) - 1/a\right)$$

Combined with the fact that $c_{F_{X'}(a)}$ is increasing in a and hits its else condition only when $c_{F_X(a)}$ hits its else condition we can conclude:

$$c_{F_{X'}(a)}(f(x_i), f(x_j), \delta) \leq c_{F_X(a)}(x_i, x_j, \delta)$$

Similarly, since:

$$\frac{2\log(W_{H_X}(x_i, x_j))}{W_{H_X}(x_i, x_j)} - \frac{2\log(a)}{a} \leq \frac{2\log(W_{H_{X'}}(f(x_i), f(x_j)))}{W_{H_{X'}}(f(x_i), f(x_j))} - \frac{2\log(a)}{a}$$

and $e_{F_X(a)}$ is decreasing in a and misses its else condition only when $e_{F_{X'}(a)}$ misses its else condition we can conclude:

$$e_{F_X(a)}(x_i), x_j, \delta) \leq e_{F_{X'}(a)}(f(x_i), f(x_j), \delta)$$

Therefore $f : X \to X'$ is a **Loss**-morphism from $(X, c_{F_X(a)}, e_{F_X(a)})$ to $(X', c_{F_{X'}(a)}, e_{F_{X'}(a)})$. Since $MDS : \mathbf{FCov} \to \mathbf{FLoss}$ trivially preserves the identity and composition, we can conclude that it is a functor and $MDS \circ \mathcal{ML}$ is a **Met**-manifold learning functor. Since $\mathcal{ML}$ is in standard form (Proposition 6.19), then $MDS \circ \mathcal{ML}$ is in standard form as well (Definition 6.50).

Next, we show that $MDS \circ \mathcal{ML}$ maps $(X, d_X) \in \mathbf{Met}$ to the metric multidimensional scaling embedding optimization problem. Define:

$$F = (MDS \circ \mathcal{ML})(X, d_X)$$

and note that:

$$W(x_i, x_j) = \sup_{\geq 0}\{a \mid a \in (0, 1],\ \exists S \in \mathcal{ML}(X, d_X)(a),\ x_i, x_j \in S\} = e^{-d_X(x_i, x_j)}$$

The F_X-loss $\mathbb{l}_{F_X}(\gamma)$ is as follows:

$$
\begin{aligned}
&\mathbb{l}_{F_X} : (X \to \mathbb{R}^m) \to \mathbb{R} \\
&\mathbb{l}_{F_X}(\gamma) \\
=\quad &\{\text{Definition of } \mathbb{l}_{F_X}\} \\
&\sum_{x_i,x_j \in X} \int_{a \in (0,1]} e_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) + c_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|)\ da \\
=\quad &\{\text{Definitions of } e_{F_X(a)} \text{ and } c_{F_X(a)}\} \\
&\sum_{x_i,x_j \in X} \|\gamma(x_i) - \gamma(x_j)\|^2 + \left(\int_{a \in (W(x_i,x_j),1]} \frac{2log(W(x_i,x_j))}{W(x_i,x_j)} - \frac{2log(a)}{a}\ da \right) + \\
&\qquad \left(2\|\gamma(x_i) - \gamma(x_j)\|^2 \int_{a \in (W(x_i,x_j),1]} \frac{1}{W(x_i,x_j)} - \frac{1}{a}\ da \right) \\
=\quad &\{\text{Compute integrals}\} \\
&C + \sum_{x_i,x_j \in X} \|\gamma(x_i) - \gamma(x_j)\|^2 + log(W(x_i,x_j))^2 + 2\|\gamma(x_i) - \gamma(x_j)\|^2 log(W(x_i,x_j)) \\
=\quad &\{\text{Definition of } W\} \\
&C + \sum_{x_i,x_j \in X} \|\gamma(x_i) - \gamma(x_j)\|^2 + d_X(x_i,x_j)^2 - 2\|\gamma(x_i) - \gamma(x_j)\|^2 d_X(x_i,x_j) \\
=\quad &\{\text{Simplify quadratic}\} \\
&C + \sum_{x_i,x_j \in X} \left(\|\gamma(x_i) - \gamma(x_j)\| - d_X(x_i,x_j)\right)^2
\end{aligned}
$$

where C is a constant factor. □

6.3.5.2 Isomap (Met-Manifold Learning Functor)

For many real world datasets it is the case that the distances between nearby points are more reliable and less noisy than the distances between far away points. The isomap algorithm Tenenbaum et al. (2000) redefines the distances between far apart points in terms of the distances between near points. Given a metric space $(X, d_X) \in \mathbf{Met}$, the isomap embedding optimization problem is $(X, m, \mathbb{l})$ where:

$$\mathbb{l}(\gamma) = \sum_{x_i,x_j \in X} (d'_X(x_i,x_j) - \|\gamma(x_i) - \gamma(x_j)\|)^2$$

such that $d'_X(x_i,x_j)$ is the length of the shortest path between x_i and x_j in the graph in which there exists an edge of length $d_X(x,x')$ between each pair of points $(x,x') \in X$ with $d_X(x,x') \leq \delta$.

Proposition 6.56. *For any $\delta \in \mathbb{R}_{\geq 0}$, there exists a hierarchical* **Met***-clustering functor $IsoCluster_\delta$ such that the* **Met***-manifold learning functor $MDS \circ IsoCluster_\delta$ maps the metric space $(X, d_X) \in \mathbf{Met}$ to the isomap embedding optimization problem.*

Proof. First, define the δ-graph of (X, d_X) to be the graph in which the vertices are the points in X and there exists an edge of length $d_X(x,x')$ between each pair of points

$(x, x') \in X$ with $d_X(x, x') \leq \delta$. Now define $IsoCluster_\delta : \mathbf{Met} \to \mathbf{FCov}$ such that $IsoCluster_\delta(X, d_X)(a)$ is the collection of maximally linked sets of the relation R_a, where for $x, x' \in X$ we have $x \, R_a \, x'$ if there exists a path of length no larger than $-\log(a)$ in the δ-graph of (X, d_X). We will show that $IsoCluster_\delta$ is a hierarchical **Met**-clustering functor.

Consider the nonexpansive map $f : (X, d_X) \to (Y, d_Y)$ and say that for some $a \in (0, 1]^{\text{op}}, x, x' \in X$ there exists:

$$S \in IsoCluster_\delta(X, d_X)(a), \;\; x, x' \in S$$

Then there exists a sequence

$$x = x_1, x_2, \cdots, x_{n-1}, x_n = x'$$

such that:

$$\begin{aligned} &\max_{i=1\cdots n-1} d_X(x_i, x_{i+1}) \leq \delta \\ &\sum_{i=1\cdots n-1} d_X(x_i, x_{i+1}) \leq -\log(a) \end{aligned}$$

which implies that:

$$\begin{aligned} &\max_{i=1\cdots n-1} d_Y(f(x_i), f(x_{i+1})) \leq \delta \\ &\sum_{i=1\cdots n-1} d_Y(f(x_i), f(x_{i+1})) \leq -\log(a) \end{aligned}$$

which in turn implies that:

$$\exists S' \in IsoCluster(Y, d_Y)(a), f(x), f(x') \in S'$$

Since $IsoCluster_\delta$ trivially preserves the identity and acts as the identity on the underlying set, we can conclude that $IsoCluster_\delta$ is a hierarchical **Met**-clustering functor.

Next, we will show that the manifold learning functor $MDS \circ IsoCluster_\delta$ maps (X, d_X) to the isomap embedding optimization problem. First define:

$$\begin{aligned} &W(x_i, x_j) = \\ &\sup_{\geq 0}\{a \;\mid\; a \in (0, 1], \; \exists S \in IsoCluster_\delta(X, d_X)(a), \; x_i, x_j \in S\} = \\ &e^{-d'_X(x_i, x_j)} \end{aligned}$$

where $d'_X(x_i, x_j)$ is the smallest δ' such that there exists a path of length no greater than δ' between x_i and x_j in the δ-graph of (X, d_X). Now if we define:

$$F_X = (MDS \circ IsoCluster_\delta)(X, d_X)$$

then following the same steps as in the proof of Proposition 6.55 we have:

$$\mathbb{I}_{F_X}(\gamma) = C + \sum_{x_i, x_j \in X} \left(\|\gamma(x_i) - \gamma(x_j)\| - d'_X(x_i, x_j)\right)^2$$

where C is a constant factor. □

6.3.5.3 k-Path Scaling (Met-Manifold Learning Functor)

Given a metric space $(X, d_X) \in \mathbf{Met}$ and $k \in \mathbb{N}$, suppose $d'_X(x_i, x_j)$ is the smallest δ such that there exists a path of $\leq k$ edges between x_i and x_j in the δ-Vietoris-Rips complex of (X, d_X). Then the **Met**-manifold learning functor $MDS \circ \mathcal{L}_k$ where $\mathcal{L}_k$ is defined as in Definition 6.52 maps (X, d_X) to the k-Path Scaling embedding optimization problem $(X, m, \mathbb{l})$ where:

$$\mathbb{l}(\gamma) = \sum_{x_i, x_j \in X} (d'_X(x_i, x_j) - \|\gamma(x_i) - \gamma(x_j)\|)^2$$

An algorithm to use this embedding optimization problem to find embeddings for a metric space $(X, d_X) \in \mathbf{Met}$ would look like:

procedure k-$PathScaling((X, d_X), m, k)$
    Initialize the $|X| \times |X|$ matrix D to all zeros
    $\forall i, j \leq |X|$
        $x_1 \leftarrow x_i$
        $x_k \leftarrow x_j$
        $D[i, j] \leftarrow \min\{\delta \mid \exists\, x_1, x_2, \cdots, x_{k-1}, x_k, d_X(x_i, x_{i+1}) \leq \delta\}$
    $\gamma \leftarrow \min_{\gamma: X \to \mathbb{R}^m} \sum_{x_i, x_j \in X} (D[i, j] - \|\gamma(x_i) - \gamma(x_j)\|)^2$
    Return γ

6.3.5.4 k-Vertex-Connected Scaling ($\mathbf{Met}_{inj}$-Manifold Learning Functor)

Culbertson et al. (2016) explore an overlapping clustering algorithm based on k-vertex-connected subgraphs. We can adapt their construction to a collection of overlapping hierarchical clustering functors:

Definition 6.57. *For $k \in \mathbb{N}$ the overlapping hierarchical clustering functor $\mathcal{VL}_k$ maps the metric space $(X, d_X) \in \mathbf{Met}$ to the fuzzy nonnested flag cover F_X of X such that for $a \in (0, 1]^{op}$ the cover $F_X(a)$ is the set of maximal k-vertex-connected subgraphs of the $-\log(a)$-Vietoris-Rips complex of (X, d_X).*

Note that $\mathcal{VL}_1 = \mathcal{SL}$ and $lim_{k \to \infty} \mathcal{VL}_k = \mathcal{ML}$. Note also that for $k > 1$ the map $\mathcal{VL}_k$ is functorial over $\mathbf{Met}_{inj}$ but not all of **Met** since a non-injective map may split a k-vertex-linked subgraph.

Now given a metric space $(X, d_X) \in \mathbf{Met}$ and $k \in \mathbb{N}$, suppose $d'_X(x_i, x_j)$ is the smallest δ such that there exists a k-vertex-connected subgraph of the δ-Vietoris-Rips complex of (X, d_X) that contains both x_i and x_j. Then the $\mathbf{Met}_{inj}$-manifold learning functor $MDS \circ \mathcal{VL}_k$ maps (X, d_X) to the k-Vertex Connected Scaling embedding optimization problem $(X, m, \mathbb{l})$ where:

$$\mathbb{l}(\gamma) = \sum_{x_i, x_j \in X} (d'_X(x_i, x_j) - \|\gamma(x_i) - \gamma(x_j)\|)^2$$

Unlike $MDS \circ \mathcal{L}_k$, for $k > 1$ the map $MDS \circ \mathcal{VL}_k$ is not functorial over all of **Met** since $\mathcal{VL}_k$ is not functorial over **Met**.

6.3.5.5 UMAP ($\mathbf{Met}_{isom}$-Manifold Learning Functor)

Recall the following constructions by McInnes et al. (2018):

Definition 6.58. *An extended-pseudometric space (X, d_X) is a tuple of a set X and a function $d_X : X \times X \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ such that:*

- $d_X(x_1, x_2) = d_X(x_2, x_1)$
- $d_X(x, x) = 0$
- $d_X(x_1, x_2) + d_X(x_2, x_3) \geq d_X(x_1, x_3)$ *or* $d_X(x_1, x_3) = \infty$

where for all $x \in \mathbb{R}_{\geq 0}$ we have:

$$x \leq \infty \qquad \infty + x = x + \infty = \infty$$

Definition 6.59. *In the category* **EPMet** *objects are finite extended-pseudometric spaces and morphisms are nonexpansive maps.*

Given a finite metric space $(X, d_X) \in \mathbf{Met}$, the UMAP algorithm builds a local finite extended-pseudometric space around each point in X, converts each local extended-pseudometric space to a fuzzy simplicial complex, and minimizes a loss function based on a fuzzy union of these fuzzy simplicial complexes. Given a finite metric space $(X, d_X) \in \mathbf{Met}$, the UMAP embedding optimization problem is $(X, m, \mathbb{l})$ where $\mathbb{l}$ is the fuzzy cross-entropy:

$$\mathbb{l}(\gamma) = \sum_{x_i, x_j \in X} W(x_i, x_j) \log\left(\frac{W(x_i, x_j)}{e^{-\|\gamma(x_i) - \gamma(x_j)\|}}\right) + (1 - W(x_i, x_j)) \log\left(\frac{1 - W(x_i, x_j)}{1 - e^{-\|\gamma(x_i) - \gamma(x_j)\|}}\right)$$

and $W(x_i, x_j)$ is the weight of the fuzzy union of the 1-simplices connecting x_i and x_j in the Vietoris-Rips complexes formed from the $|X|$ local finite extended-pseudometric spaces (X, d_{x_i}) where:

$$d_{x_i}(x_j, x_k) = \begin{cases} d_X(x_j, x_k) - \min_{l=1\cdots n} d_X(x_i, x_l) & x_i = x_j \text{ or } x_i = x_k \\ \infty & \text{else} \end{cases}$$

In order to construct UMAP as a manifold learning functor we will first describe the overlapping hierarchical clustering functor that it factors through:

Proposition 6.60. *There exists a hierarchical $\mathbf{Met}_{isom}$-clustering functor*

$$\mathit{FuzzySimplex} : \mathbf{Met}_{isom} \to \mathbf{FCov}_{bij}$$

that decomposes into the composition of four functors that:

1. *Build a local extended-pseudometric space around each point in X*
2. *Convert each local extended-pseudometric space to a fuzzy simplicial complex*
3. *Take a fuzzy union of these fuzzy simplicial complexes*

4. Convert the resulting fuzzy simplicial complex to a fuzzy nonnested flag cover

Proof. Before we begin, we will show that the category $\mathbf{FSCpx}_{bij}$ (Definition 2.60) of fuzzy simplicial complexes and bijective simplicial maps between them is finitely co-complete.

For some finite category $\mathbf{C}$ consider a functor of the form $F : \mathbf{C} \to \mathbf{FSCpx}_{bij}$. Define the fuzzy simplicial complex $F_c : (0,1]^{\text{op}} \to \mathbf{SCpx}_{bij}$ in $\mathbf{FSCpx}_{bij}$ to map $a \in (0,1]^{\text{op}}$ to the simplicial complex whose set of n-simplices is:

$$\cup_{o \in Ob(\mathbf{C})} F(o)(a)[n]$$

where $F(o)(a)[n]$ is the set of n-simplices in the simplicial complex $F(o)(a) \in \mathbf{SCpx}_{bij}$ and $Ob(\mathbf{C})$ is the set of objects in $\mathbf{C}$. It is clear that this is the minimal fuzzy simplicial complex such that there exists a natural transformation from each fuzzy simplicial complex $F(o), o \in Ob(\mathbf{C})$ into this fuzzy simplicial complex, so F_c is the colimit of F and $\mathbf{FSCpx}_{bij}$ is finitely co-complete.

Now we can proceed to prove Proposition 6.60.

For any $N, N' \in \mathbb{N}$ such that $N \neq N'$, the size N metric spaces and the size N' metric spaces have no morphisms between them in $\mathbf{Met}_{isom}$. Therefore, we can uniquely define *FuzzySimplex* by defining a separate functor:

$$FuzzySimplex_N : \mathbf{Met}_{isom}(N) \to \mathbf{FCov}_{bij}$$

for each $N \in \mathbb{N}$, where $\mathbf{Met}_{isom}(N)$ is the subcategory of $\mathbf{Met}_{isom}$ where objects are restricted to metric spaces (X, d_X) with cardinality N.

To start, denote the N-element discrete category $\mathbf{N}$ and define the following functor for step 1 (build a local extended-pseudometric space around each point):

$$LocalMetric_N : \mathbf{Met}_{isom}(N) \to \mathbf{EPMet}_{bij}^{\mathbf{N}}$$

This functor sends the N-element metric space (X, d_X) to the functor $F : \mathbf{N} \to \mathbf{EPMet}_{bij}$ that maps $i \in \mathbf{N}$ to (X, d_{x_i}) where:

$$d_{x_i}(x_j, x_k) = \begin{cases} d_X(x_j, x_k) - \min_{l=1 \cdots n} d_X(x_i, x_l) & x_i = x_j \text{ or } x_i = x_k \\ \infty & \text{else} \end{cases}$$

$LocalMetric_N$ sends the function $f : (X, d_X) \to (Y, d_Y)$ to the natural transformation in which each component is f. Since f is an isometry, we have that:

- $d_X(x_j, x_k) = d_Y(f(x_j), f(x_k))$
- If $x_i = x_j$ then $f(x_i) = f(x_j)$
- If $x_i \neq x_j$ then $f(x_i) \neq f(x_j)$

Together this implies that for each $x_i \in X$ and pair $x_j, x_k \in X$:

$$
\begin{aligned}
& d_{x_i}(x_j, x_k) \\
= \quad & \{\text{Definition of } d_{x_i}\} \\
& \begin{cases} d_X(x_j, x_k) - \min_{l=1\cdots n} d_X(x_i, x_l) & x_i = x_j \text{ or } x_i = x_k \\ \infty & \text{else} \end{cases} \\
= \quad & \{f \text{ is an isometry}\} \\
& \begin{cases} d_Y(f(x_j), f(x_k)) - \min_{l=1\cdots n} d_Y(f(x_i), f(x_l)) & f(x_i) = f(x_j) \text{ or } f(x_i) = f(x_k) \\ \infty & \text{else} \end{cases} \\
= \quad & \{\text{Definition of } d_{f(x_i)}\} \\
& d_{f(x_i)}(f(x_j), f(x_k))
\end{aligned}
$$

and therefore this natural transformation is a morphism in $\mathbf{EPMet}_{bij}^{\mathbf{N}}$. Since $LocalMetric_N$ trivially preserves composition and the identity, it is a functor. For step 2 (convert each local extended-pseudometric space to a fuzzy simplicial complex), we will adapt the $FinSing$ functor from Definition 6.13 to form the functor:

$$(FinSing \circ \_)_N : \mathbf{EPMet}_{bij}^{\mathbf{N}} \to \mathbf{FSCpx}_{bij}^{\mathbf{N}}$$

which maps the functor $F : \mathbf{N} \to \mathbf{EPMet}_{bij}$ to the functor $(FinSing \circ F) : \mathbf{N} \to \mathbf{FSCpx}_{bij}$. For step 3 (take a fuzzy union of these fuzzy simplicial complexes), we apply the colimit functor:

$$colim_N : \mathbf{FSCpx}_{bij}^{\mathbf{N}} \to \mathbf{FSCpx}_{bij}$$

which sends an indexed set of fuzzy simplicial complexes in $\mathbf{FSCpx}_{bij}^{\mathbf{N}}$ to its logical fuzzy union. This functor exists since the category of fuzzy simplicial complexes and bijective simplicial maps is finitely co-complete. In a logical fuzzy union the strength of a simplex is defined to be its maximum strength among the complexes we are adjoining[2] .

For step 4 (convert the resulting fuzzy simplicial complex to a fuzzy nonnested flag cover), we use the following functor from Definition 6.8:

$$(Flag \circ \_) : \mathbf{FSCpx} \to \mathbf{FCov}$$

Since $Flag$ maps bijective simplicial maps to bijections, the image of this functor over $\mathbf{FSCpx}_{bij}$ is $\mathbf{FCov}_{bij}$.

Now we can compose steps 1-4 and apply a coproduct over $N \in \mathbb{N}$ to extend this to the following functor from $\mathbf{Met}_{isom}$ to $\mathbf{FCov}_{bij}$:

$$FuzzySimplex = \coprod_{N \in \mathbb{N}} (Flag \circ \_) \circ colim_N \circ (FinSing \circ \_)_N \circ LocalMetric_N$$

We now show $FuzzySimplex$ is a hierarchical $\mathbf{Met}_{isom}$-clustering functor. Since $FuzzySimplex$ is by definition a functor, we simply need to show for any (X, d_X) that $FuzzySimplex(X, d_X)$

[2]This is different from the probabilistic simplicial complex union that the UMAP python code uses (McInnes et al., 2018).

is a fuzzy nonnested flag cover of X. First note that for any object $o \in \mathbf{N}$, the underlying set of the following fuzzy simplicial complex is X:

$$((FinSing \circ \_)_N \circ LocalMetric_N)(X, d_X)(o)$$

Therefore the underlying set of the following fuzzy simplicial complex is X as well:

$$(colim_N \circ (FinSing \circ \_)_N \circ LocalMetric_N)(X, d_X)$$

This implies that $FuzzySimplex(X, d_X)$ is a fuzzy nonnested flag cover of X.

□

Next, we introduce the functor $FCE : \mathbf{FCov}_{bij} \to \mathbf{FLoss}$ that we will compose *FuzzySimplex* with to construct the functor UMAP:

Proposition 6.61. *There exists a functor* $FCE : \mathbf{FCov}_{bij} \to \mathbf{FLoss}$ *such that the composition*

$$FCE \circ FuzzySimplex : \mathbf{Met}_{isom} \to \mathbf{FLoss}$$

is a $\mathbf{Met}_{isom}$*-manifold learning functor that maps the metric space* $(X, d_X) \in \mathbf{Met}_{isom}$ *to the UMAP embedding optimization problem.*

Proof. Define $FCE : \mathbf{FCov}_{bij} \to \mathbf{FLoss}$ to map the fuzzy nonnested flag cover $H_X : (0,1]^{\text{op}} \to \mathbf{Cov}_{bij}$ with underlying set X to $F_X : (0,1]^{\text{op}} \to \mathbf{Loss}$ where:

$$F_X(a) = (X, c_{F_X(a)}, e_{F_X(a)})$$

and:

$$c_{F_X(a)} : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R} \qquad e_{F_X(a)} : X \times X \times \mathbb{R}_{\geq 0} \to \mathbb{R}$$

$$c_{F_X(a)}(x_i, x_j, \delta) = \begin{cases} 0 & \delta = 0 \text{ or } \exists S \in H_X(a),\ x_i, x_j \in S\}) \\ -\log(1 - e^{-\delta}) & \text{else} \end{cases}$$

$$e_{F_X(a)}(x_i, x_j, \delta) = \begin{cases} -\log(e^{-\delta}) & \exists S \in H_X(a),\ x_i, x_j \in S\}) \\ 0 & \text{else} \end{cases}$$

We will show that $FCE \circ FuzzySimplex$ is an $\mathbf{Met}_{isom}$-manifold learning functor that maps any metric space $(X, d_X) \in \mathbf{Met}_{isom}$ to the UMAP embedding optimization problem over the distance matrix of d_X.

First, we need to show that $FCE : \mathbf{FCov}_{bij} \to \mathbf{FLoss}$ is a functor. Consider the fuzzy nonnested flag covers H_X and $H_{X'}$ in $\mathbf{FCov}_{bij}$ with underlying sets X, X' respectively such that there exists a morphism f in $\mathbf{FCov}_{bij}$ between them (a natural transformation with bijective components). Suppose:

$$FCE(H_X)(a) = F_X(a) = (X, c_{F_X(a)}, e_{F_X(a)})$$
$$FCE(H_{X'})(a) = F_{X'}(a) = (X', c_{F_{X'}(a)}, e_{F_{X'}(a)})$$

Since each component of f is bijective, it must be that $|X|' = |X|$. Now for each:

$$a \in (0,1], x_i, x_j \in X, \exists S \in H_X(a),\ x_i, x_j \in S$$

by definition:

$$\exists S \in H_{X'}(a),\ f(x_i), f(x_j) \in S$$

Now note that for all $\delta > 0$ we have $-\log(1 - e^{-\delta}) > 0$. Since $c_{F_{X'}(a)}$ hits its else condition only when $c_{F_X(a)}$ hits its else condition, we can conclude:

$$c_{F_{X'}(a)}(f(x_i), f(x_j), \delta) \leq c_{F_X(a)}(x_i, x_j, \delta)$$

Next, note that for all $\delta \geq 0$ we have $-\log(e^{-\delta}) \geq 0$. Since $e_{F_X(a)}$ misses its else condition only when $e_{F_{X'}(a)}$ misses its else condition, we can conclude:

$$e_{F_X(a)}(x_i, x_j, \delta) \leq e_{F_{X'}(a)}(f(x_i), f(x_j), \delta)$$

Therefore $f : X \to X'$ is a **Loss**-morphism from $(X, c_{F_X(a)}, e_{F_X(a)})$ to $(X', c_{F_{X'}(a)}, e_{F_{X'}(a)})$. Since $FCE : \mathbf{FCov}_{bij} \to \mathbf{FLoss}$ trivially preserves the identity and composition, we can conclude that it is a functor and $FCE \circ FuzzySimplex$ is a $\mathbf{Met}_{isom}$-manifold learning functor.

Next, we will show that $FCE \circ FuzzySimplex$ maps (X, d_X) to the UMAP embedding optimization problem. Define

$$F_X = (FCE \circ FuzzySimplex)(X, d_X)$$

We have that the F_X-loss $\mathbb{l}_{F_X}(\gamma)$ is:

$$\begin{aligned}
&\mathbb{l}_{F_X} : (X \to \mathbb{R}^m) \to \mathbb{R} \\
&\mathbb{l}_{F_X}(\gamma) \\
=\ & \{\text{Definition of } \mathbb{l}_{F_X}\} \\
&\sum_{x_i, x_j \in X} \int_{a \in (0,1]} c_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) + e_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) da \\
=\ & \{\text{Rewrite integral}\} \\
&\sum_{\substack{x_i, x_j \in X \\ \|\gamma(x_i) - \gamma(x_j)\| > 0}} \int_{a \in (0,1]} c_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) + e_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) da \\
=\ & \{\text{Definitions of } e_{F_X(a)} \text{ and } c_{F_X(a)}\} \\
&\sum_{\substack{x_i, x_j \in X \\ \|\gamma(x_i) - \gamma(x_j)\| > 0}} \int_{a \in (W(x_i,x_j),1]} -log(1 - e^{-\|\gamma(x_i) - \gamma(x_j)\|})\ da - \int_{a \in (0, W(x_i,x_j)]} log(e^{-\|\gamma(x_i) - \gamma(x_j)\|})\ da \\
=\ & \{\text{Compute integral}\} \\
&\sum_{\substack{x_i, x_j \in X \\ \|\gamma(x_i) - \gamma(x_j)\| > 0}} -(1 - W(x_i, x_j))log(1 - e^{-\|\gamma(x_i) - \gamma(x_j)\|}) - W(x_i, x_j)log(e^{-\|\gamma(x_i) - \gamma(x_j)\|}) \\
=\ & \{\text{Rearrange terms}\} \\
&C + \sum_{\substack{x_i, x_j \in X \\ \|\gamma(x_i) - \gamma(x_j)\| > 0}} (1 - W(x_i, x_j))log\left(\frac{1 - W(x_i, x_j)}{1 - e^{-\|\gamma(x_i) - \gamma(x_j)\|}}\right) + W(x_i, x_j)log\left(\frac{W(x_i, x_j)}{e^{-\|\gamma(x_i) - \gamma(x_j)\|}}\right)
\end{aligned}$$

where:

$$W(x_i, x_j) = \sup_{\geq 0}\{a \mid a \in (0,1],\ \exists S \in \mathit{FuzzySimplex}(X, d_X)(a),\ x_i, x_j \in S\}$$

is the weight of the fuzzy 1-simplex connecting x_i and x_j and:

$$C = \sum_{x_i, x_j \in X} (1 - W(x_i, x_j))log(1 - W(x_i, x_j)) + W(x_i, x_j)log(W(x_i, x_j))$$

is a constant. □

Since the UMAP distance rescaling procedure does not preserve nonexpansive maps, even if a map from (X, d_X) to $(X', d_{X'})$ is nonexpansive, this will not necessarily be the case for all of the local extended-pseudometric spaces (X, d_{x_i}) that we build from (X, d_X) and $(X', d_{X'})$. For this reason $\mathit{FCE} \circ \mathit{FuzzySimplex}$ is not functorial over $\mathbf{Met}_{bij}$.

6.3.6 Stability of Manifold Learning Algorithms

We can use this functorial perspective on manifold learning to reason about the stability of manifold learning algorithms under dataset noise. In order to do this, we will use the interleaving distance (Definition 3.16). Recall that the interleaving distance measures the distance between functors $F : \mathbb{R}_{\geq 0} \to \mathbf{C}$.

In this section we will work with the following subcategories of **Loss** and **FLoss**:

Definition 6.62. $\mathbf{Loss_{id}}$ *is the preorder subcategory of* **Loss** *in which we write:*

$$(X, c_X, e_X) \leq_{\mathbf{Loss_{id}}} (X, c'_X, e'_X)$$

to indicate that the identity function on X is a morphism in **Loss** *from (X, c_X, e_X) to (X, c'_X, e'_X).*

Definition 6.63. $\mathbf{FLoss_{id}}$ *is the subcategory of* **FLoss** *in which morphisms are limited to natural transformations with components in* $\mathbf{Loss_{id}}$.

In order to study interleavings between functors in $\mathbf{FLoss_{id}}$ whose domain is $(0,1]^{\mathrm{op}}$ rather than $\mathbb{R}_{\geq 0}$, we will say that the functors F_X, G_X are ϵ_*-interleaved when there is an ϵ-interleaving (Definition 3.15) between the functors $F_X \circ r$ and $G_X \circ r$ where $r(\delta) = e^{-\delta}$. We will also write:

$$d_{I_*}(F_X, G_X) = d_I(F_X \circ r, G_X \circ r)$$

where d_I is the interleaving distance.

Proposition 6.64. *Given a functor $M : \mathbf{Met}_{id} \to \mathbf{FLoss_{id}}$ such that the functor $\iota \circ M = L \circ H$ is a standard form $\mathbf{Met}_{id}$-manifold learning functor (where $\iota : \mathbf{FLoss_{id}} \hookrightarrow \mathbf{FLoss}$ is the inclusion functor) and a pair of metric spaces $(X, d_X), (X, d'_X) \in \mathbf{Met}_{id}$ such that:*

$$f : (X, d_X + \epsilon) \leq_{\mathbf{Met}_{id}} (X, d'_X)$$
$$g : (X, d'_X + \epsilon) \leq_{\mathbf{Met}_{id}} (X, d_X)$$

in **D**, *we have:*

$$d_{I_*}(M(X, d_X), M(X, d'_X)) \leq \epsilon$$

Proof. By Definition 6.18 we have that:

$$H(X, d_X + \epsilon)(e^{-\delta}) \simeq H(X, d_X)(e^{-\delta+\epsilon})$$

so by functoriality for any $\delta \in \mathbb{R}_{\geq 0}$ we have that:

$$H(X, d'_X)(e^{-\delta+\epsilon}) \simeq H(X, d'_X + \epsilon)(e^{-\delta}) \leq_{\mathbf{FCov}_{id}} H(X, d_X)(e^{-\delta})$$
$$H(X, d_X)(e^{-\delta+\epsilon}) \simeq H(X, d_X + \epsilon)(e^{-\delta}) \leq_{\mathbf{FCov}_{id}} H(X, d'_X)(e^{-\delta})$$

Therefore since $M = L \circ H$ by functoriality we also have that:

$$M(X, d'_X)(e^{-\delta+\epsilon}) \leq_{\mathbf{FLoss}_{\mathbf{id}}} M(X, d_X)(e^{-\delta})$$
$$M(X, d_X)(e^{-\delta+\epsilon}) \leq_{\mathbf{FLoss}_{\mathbf{id}}} M(X, d'_X)(e^{-\delta})$$

and therefore:

$$(M(X, d'_X) \circ r)(\delta - \epsilon) \leq_{\mathbf{FLoss}_{\mathbf{id}}} (M(X, d_X) \circ r)(\delta)$$
$$(M(X, d_X) \circ r)(\delta - \epsilon) \leq_{\mathbf{FLoss}_{\mathbf{id}}} (M(X, d'_X) \circ r)(\delta)$$

Since $\mathbf{Loss}_{\mathbf{id}}$ is a preorder, this implies that:

$$M(X, d_X) \circ r : \mathbb{R}_{\geq 0} \to \mathbf{Loss}_{\mathbf{id}} \qquad M(X, d'_X) \circ r : \mathbb{R}_{\geq 0} \to \mathbf{Loss}_{\mathbf{id}}$$

are ϵ-interleaved and therefore $M(X, d_X), M(X, d'_X)$ are ϵ_*-interleaved. □

Proposition 6.64 is similar in spirit to previous results that use the Gromov-Hausdorff distance between metric spaces to bound the bottleneck or homotopy interleaving distances between their corresponding Vietoris-Rips complexes (Chazal et al., 2014; Bubenik and Scott, 2014; Scoccola, 2020; Blumberg and Lesnick, 2017). We can use this to prove the following:

Theorem 6.65. *Suppose we have a functor $M : \mathbf{Met}_{id} \to \mathbf{FLoss}_{\mathbf{id}}$ such that $\iota \circ M = L \circ H$ is an extensible standard form $\mathbf{Met}_{id}$-manifold learning functor, a pair of metric spaces $(X, d_X), (X, d'_X) \in \mathbf{Met}_{id}$ satisfying the conditions in Proposition 6.64, the embedding map $\gamma_{M(X,d_X)} : X \to \mathbb{R}^m$ that minimizes $\mathbb{l}_{M(X,d_X)}$ and the embedding map $\gamma_{M(X,d'_X)} : X \to \mathbb{R}^m$ that minimizes $\mathbb{l}_{M(X,d'_X)}$. Then if the following conditions hold for all $a \in (0, 1]^{op}, x_i, x_j \in X, \delta \geq 0$:*

$$|c_{M(X,d_X)(a)}(x_i, x_j, \delta)| \leq \frac{K_{\mathbf{c}}}{2}| \qquad |c_{M(X,d'_X)(a)}(x_i, x_j, \delta)| \leq \frac{K_{\mathbf{c}}}{2}$$
$$|e_{M(X,d_X)(a)}(x_i, x_j, \delta)| \leq \frac{K_{\mathbf{e}}}{2} \qquad |e_{M(X,d'_X)(a)}(x_i, x_j, \delta)| \leq \frac{K_{\mathbf{e}}}{2}$$

where:

$$M(X, d_X)(a) = (X, c_{M(X,d_X)(a)}, e_{M(X,d_X)(a)})$$
$$M(X, d'_X)(a) = (X, c_{M(X,d'_X)(a)}, e_{M(X,d'_X)(a)})$$

we have that:

$$\mathbb{l}_{M(X,d_X)}\left(\gamma_{M(X,d'_X)}\right) \leq \mathbb{l}_{M(X,d_X)}\left(\gamma_{M(X,d_X)}\right) + K_{\mathbf{c}}|X|^2(1 - e^{-\epsilon}) + K_{\mathbf{e}}|X|^2(e^{\epsilon} - 1)$$

If for all $x_i, x_j \in X$ the function $e_{M(X,d_X)(a)}(x_i, x_j, \delta)$ is constant in δ (such as for any M that factors as $M = MDS \circ H$) we have:

$$\mathbb{l}_{M(X,d_X)}\left(\gamma_{M(X,d'_X)}\right) \leq \mathbb{l}_{M(X,d_X)}\left(\gamma_{M(X,d_X)}\right) + K_{\mathbf{c}}|X|^2(1 - e^{-\epsilon})$$

Proof. By using Proposition 6.64, we see that in order to prove Theorem 6.65 we simply need to show that if F_X, G_X are ϵ^*-interleaved functors in $\mathbf{FLoss_{id}}$ such that the map $\gamma_{F_X} : X \to \mathbb{R}^m$ minimizes $\mathbb{l}_{F_X}$, the map $\gamma_{G_X} : X \to \mathbb{R}^m$ minimizes $\mathbb{l}_{G_X}$, and for all $a \in (0,1]^{\text{op}}$:

$$\begin{aligned}
&c_{F_X(a)} \geq 0 \qquad && c_{G_X(a)} \geq 0 \\
&e_{F_X(a)} \leq 0 \qquad && e_{G_X(a)} \leq 0 \\
&|c_{F_X(a)}| \leq \frac{K_{\mathbf{c}}}{2} \qquad && |c_{G_X(a)}| \leq \frac{K_{\mathbf{c}}}{2} \\
&|e_{F_X(a)}| \leq \frac{K_{\mathbf{e}}}{2} \qquad && |e_{G_X(a)}| \leq \frac{K_{\mathbf{e}}}{2}
\end{aligned}$$

then we have:

$$\mathbb{l}_{F_X}(\gamma_{G_X}) \leq \mathbb{l}_{F_X}(\gamma_{F_X}) + K_{\mathbf{c}}|X|^2(1 - e^{-\epsilon}) + K_{\mathbf{e}}|X|^2(e^{\epsilon} - 1)$$

And that in the special case where for all $x_i, x_j \in X, a \in (0,1]^{\text{op}}$ the function:

$$e_{F_X(a)}(x_i, x_j, \_) : \mathbb{R}_{\geq 0} \to \mathbb{R}$$

is constant in δ we have:

$$\mathbb{l}_{F_X}(\gamma_{G_X}) \leq \mathbb{l}_{F_X}(\gamma_{F_X}) + K_{\mathbf{c}}|X|^2(1 - e^{-\epsilon})$$

Now for brevity we will write:

$$\begin{aligned}
\mathbf{e}_{F_X(a)}(\gamma) &= \sum_{x_i, x_j \in X} e_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) \\
\mathbf{c}_{F_X(a)}(\gamma) &= \sum_{x_i, x_j \in X} c_{F_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) \\
\mathbf{e}_{G_X(a)}(\gamma) &= \sum_{x_i, x_j \in X} e_{G_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|) \\
\mathbf{c}_{G_X(a)}(\gamma) &= \sum_{x_i, x_j \in X} c_{G_X(a)}(x_i, x_j, \|\gamma(x_i) - \gamma(x_j)\|)
\end{aligned}$$

By the definition of ϵ-interleaving we have the following for any $\gamma : X \to \mathbb{R}^m$.

$$\begin{aligned}
&\mathbf{c}_{G_X(a * e^{-\epsilon})}(\gamma) \leq \mathbf{c}_{F_X(a)}(\gamma) \\
&\mathbf{e}_{F_X(a)}(\gamma) \leq \mathbf{e}_{G_X(a * e^{-\epsilon})}(\gamma) \\
&\mathbf{c}_{F_X(a * e^{-\epsilon})}(\gamma) \leq \mathbf{c}_{G_X(a)}(\gamma) \\
&\mathbf{e}_{G_X(a)}(\gamma) \leq \mathbf{e}_{F_X(a * e^{-\epsilon})}(\gamma)
\end{aligned}$$

Now we can conclude that:

$$
\begin{aligned}
& \mathbb{I}_{F_X}(\gamma_{G_X}) \\
= \quad & \{\text{Definition of } \mathbb{I}_{F_X}\} \\
& \int_{a\in(0,1]} \mathbf{c}_{F_X(a)}(\gamma_{G_X})\ da + \int_{a\in(0,1]} \mathbf{e}_{F_X(a)}(\gamma_{G_X})\ da \\
\leq \quad & \{\text{Apply the } \star \text{ steps below}\} \\
& e^{-\epsilon} \int_{a\in(0,1]} \mathbf{c}_{G_X(a)}(\gamma_{G_X})\ da + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) + \int_{a\in(0,1]} \mathbf{e}_{F_X(a)}(\gamma_{G_X})\ da \\
\leq \quad & \{\text{Apply the } \star\star \text{ steps below}\} \\
& e^{-\epsilon} \int_{a\in(0,1]} \mathbf{c}_{G_X(a)}(\gamma_{G_X})\ da + e^{\epsilon} \int_{a\in(0,1]} \mathbf{e}_{G_X(a)}(\gamma_{G_X})\ da + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) + (e^{\epsilon}-1)\frac{K_{\mathbf{e}}}{2}|X|^2 \\
\leq \quad & \{\mathbf{c}_{G_X(a)}(\gamma_{G_X}) \text{ is nonnegative and } \mathbf{e}_{G_X(a)}(\gamma_{G_X}) \text{ is nonpositive}\} \\
& \left(\int_{a\in(0,1]} \mathbf{c}_{G_X(a)}(\gamma_{G_X}) + \mathbf{e}_{G_X(a)}(\gamma_{G_X})\ da \right) + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) + (e^{\epsilon}-1)\frac{K_{\mathbf{e}}}{2}|X|^2 \\
\leq \quad & \{\text{Optimality of } \gamma_{G_X}\} \\
& \left(\int_{a\in(0,1]} \mathbf{c}_{G_X(a)}(\gamma_{F_X}) + \mathbf{e}_{G_X(a)}(\gamma_{F_X})\ da \right) + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) + (e^{\epsilon}-1)\frac{K_{\mathbf{e}}}{2}|X|^2 \\
\leq \quad & \{\text{Apply the } \star \text{ steps below}\} \\
& \int_{a\in(0,1]} e^{-\epsilon}\mathbf{c}_{F_X(a)}(\gamma_{F_X}) + \mathbf{e}_{G_X(a)}(\gamma_{F_X})\ da + K_{\mathbf{c}}|X|^2(1-e^{-\epsilon}) + (e^{\epsilon}-1)\frac{K_{\mathbf{e}}}{2}|X|^2 \\
\leq \quad & \{\text{Apply the } \star\star \text{ steps below}\} \\
& \int_{a\in(0,1]} e^{-\epsilon}\mathbf{c}_{F_X(a)}(\gamma_{F_X}) + e^{\epsilon}\mathbf{e}_{F_X(a)}(\gamma)\ da + K_{\mathbf{c}}|X|^2(1-e^{-\epsilon}) + (e^{\epsilon}-1)K_{\mathbf{e}}|X|^2 \\
\leq \quad & \{\text{Definition of } \mathbb{I}_{F_X}\} \\
& \mathbb{I}_{F_X}(\gamma_{F_X}) + K_{\mathbf{c}}|X|^2(1-e^{-\epsilon}) + (e^{\epsilon}-1)K_{\mathbf{e}}|X|^2
\end{aligned}
$$

And in the special case where for all $x_i, x_j \in X, a \in (0,1]^{\text{op}}$ the function:

$$e_{F_X(a)}(x_i, x_j, \_) : \mathbb{R}_{\geq 0} \to \mathbb{R}$$

is constant in δ we have:

$$
\begin{aligned}
& \mathbb{I}_{F_X}(\gamma_{G_X}) \\
= \quad & \{\text{Definition of } \mathbb{I}_{F_X}\} \\
& \int_{a\in(0,1]} \mathbf{c}_{F_X(a)}(\gamma_{G_X})\ da + \int_{a\in(0,1]} \mathbf{e}_{F_X(a)}(\gamma_{G_X})\ da \\
\leq \quad & \{\text{Apply the } \star \text{ steps below}\} \\
& e^{-\epsilon} \int_{a\in(0,1]} \mathbf{c}_{G_X(a)}(\gamma_{G_X})\ da + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) + \int_{a\in(0,1]} \mathbf{e}_{F_X(a)}(\gamma_{G_X})\ da \\
\leq \quad & \{\text{Optimality of } \gamma_{G_X}\} \\
& e^{-\epsilon} \int_{a\in(0,1]} \mathbf{c}_{G_X(a)}(\gamma_{F_X})\ da + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) + \int_{a\in(0,1]} \mathbf{e}_{F_X(a)}(\gamma_{F_X})\ da \\
\leq \quad & \{\text{Apply the } \star \text{ steps below}\} \\
& e^{-2\epsilon} \int_{a\in(0,1]} \mathbf{c}_{F_X(a)}(\gamma_{F_X})\ da + K_{\mathbf{c}}|X|^2(1-e^{-\epsilon}) + \int_{a\in(0,1]} \mathbf{e}_{F_X(a)}(\gamma_{F_X})\ da \\
\leq \quad & \{\text{Definition of } \mathbb{I}_{F_X}\} \\
& \mathbb{I}_{F_X}(\gamma_{F_X}) + K_{\mathbf{c}}|X|^2(1-e^{-\epsilon})
\end{aligned}
$$

The steps marked with $\star$ are by the following:

$$
\begin{aligned}
& \int_{a\in(0,1]} \mathbf{c}_{F_X(a)}(\gamma)\ da \\
\leq \quad & \{\text{Definition of } K_{\mathbf{c}}\} \\
& \int_{a\in(0,1]} \mathbf{c}_{F_X(a)}(\gamma)\ da + \left[-\int_{a\in(e^{-\epsilon},1]} \mathbf{c}_{F_X(a)}(\gamma)\ da + \int_{a\in(e^{-\epsilon},1]} \frac{K_{\mathbf{c}}}{2}|X|^2\ da \right] \\
= \quad & \{\text{Compute integral}\} \\
& \int_{a\in(0,1]} \mathbf{c}_{F_X(a)}(\gamma)\ da - \int_{a\in(e^{-\epsilon},1]} \mathbf{c}_{F_X(a)}(\gamma)\ da + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) \\
= \quad & \{\text{Rearrange integrals}\} \\
& \int_{a\in(0,e^{-\epsilon}]} \mathbf{c}_{F_X(a)}(\gamma)\ da + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) \\
= \quad & \{\text{Integration by substitution}\} \\
& e^{-\epsilon} \int_{a\in(0,1]} \mathbf{c}_{F_X(a*e^{-\epsilon})}(\gamma)\ da + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon}) \\
\leq \quad & \{\text{Interleaving implies } \mathbf{c}_{F_X(a*e^{-\epsilon})}(\gamma) \leq \mathbf{c}_{G_X(a)}(\gamma)\} \\
& e^{-\epsilon} \int_{a\in(0,1]} \mathbf{c}_{G_X(a)}(\gamma)\ da + \frac{K_{\mathbf{c}}}{2}|X|^2(1-e^{-\epsilon})
\end{aligned}
$$

The steps marked with $\star\star$ are by the following:

$$
\begin{aligned}
& \int_{a\in(0,1]} \mathbf{e}_{F_X(a)}(\gamma)\ da \\
\leq\ & \{\text{Interleaving implies } \mathbf{e}_{F_X(a)}(\gamma) \leq \mathbf{e}_{G_X(a*e^{-\epsilon})}(\gamma)\} \\
& \int_{a\in(0,1]} \mathbf{e}_{G_X(a*e^{-\epsilon})}(\gamma)\ da \\
=\ & \{\text{Integration by substitution}\} \\
& \frac{1}{e^{-\epsilon}} \int_{a\in(0,e^{-\epsilon}]} \mathbf{e}_{G_X(a)}(\gamma)\ da \\
=\ & \{\text{Rearrange integral}\} \\
& \frac{1}{e^{-\epsilon}} \left(\int_{a\in(0,1]} \mathbf{e}_{G_X(a)}(\gamma)\ da - \int_{a\in(e^{-\epsilon},1]} \mathbf{e}_{G_X(a)}(\gamma)\ da \right) \\
\leq\ & \{\text{Definition of } K_{\mathbf{e}}\} \\
& \frac{1}{e^{-\epsilon}} \left(\int_{a\in(0,1]} \mathbf{e}_{G_X(a)}(\gamma)\ da + \int_{a\in(e^{-\epsilon},1]} \frac{K_{\mathbf{e}}}{2}|X|^2\ da \right) \\
=\ & \{\text{Compute integral}\} \\
& \frac{1}{e^{-\epsilon}} \left(\int_{a\in(0,1]} \mathbf{e}_{G_X(a)}(\gamma)\ da + (1-e^{-\epsilon})\frac{K_{\mathbf{e}}}{2}|X|^2 \right) \\
=\ & \{\text{Distribute the } \tfrac{1}{e^{-\epsilon}} = e^{\epsilon} \text{ term}\} \\
& e^{\epsilon} \int_{a\in(0,1]} \mathbf{e}_{G_X(a)}(\gamma)\ da + (e^{\epsilon}-1)\frac{K_{\mathbf{e}}}{2}|X|^2
\end{aligned}
$$

□

For a very simple example, consider multidimensional scaling without dimensionality reduction. Suppose we have some finite set X and two embedding maps:

$$\gamma : X \to \mathbb{R}^a \qquad \gamma' : X \to \mathbb{R}^a$$

such that we can define metric spaces $(X, d_X), (X, d'_X)$ where:

$$d_X(x_i, x_j) = \|\gamma(x_i) - \gamma(x_j)\| \qquad d'_X(x_i, x_j) = \|\gamma'(x_i) - \gamma'(x_j)\|$$

Now suppose $M = MDS \circ \mathcal{ML}$. In this case γ, γ' respectively minimize $\mathbb{I}_{M(X,d_X)}$ and $\mathbb{I}_{M(X,d'_X)}$ and we have:

$$\mathbb{I}_{M(X,d_X)}(\gamma) = \mathbb{I}_{M(X,d'_X)}(\gamma') = 0$$

In this case Theorem 6.65 simply bounds the average value of the following quantity across all i, j:

$$(\|\gamma(x_i) - \gamma(x_j)\| - \|\gamma'(x_i) - \gamma'(x_j)\|)^2$$

in terms of the largest value of this quantity across all x_i, x_j.

These bounds apply to a very general class of manifold learning algorithms, including topologically unstable algorithms like isomap (Balasubramanian, 2002). As an example, for some $n \in \mathbb{N}, r > 0$ define:

$$\delta = 2r \sin\left(\frac{2\pi}{2n}\right)$$

Now suppose we have a finite ordered set X and an embedding map $\gamma_2 : X \to \mathbb{R}^2$ such that each $\gamma_2(x_i)$ lies on a radius $r + \epsilon$ zero-centered circle in $\mathbb{R}^2$ and for each pair x_i, x_{i+1} we have:

$$\|\gamma_2(x_{i+1}) - \gamma_2(x_i)\| = \delta$$

For any i, j where $|j - i| > 1$ we have:

$$\|\gamma_2(x_i) - \gamma_2(x_j)\| > \delta$$

Now define the metric space (X, d_X) such that:

$$d_X(x_i, x_j) = \|\gamma_2(x_i) - \gamma_2(x_j)\|$$

and consider using isomap to project this metric space onto $\mathbb{R}^1$. Given the functor $M = MDS \circ IsoCluster_\delta$ for any embedding map $\gamma_1 : X \to \mathbb{R}$ such that:

$$\gamma_1(x_{i+1}) - \gamma_1(x_i) = \delta$$

we have:

$$\mathbb{I}_{M(X,d_X)}(\gamma_1) = 0$$

Now suppose that we instead apply isomap to a noisy view of (X, d_X): that is, some metric space (X, d'_X) where:

$$d'_X(x_i, x_j) = \|\gamma'_2(x_i) - \gamma'_2(x_j)\|$$

where $\gamma'_2 : X \to \mathbb{R}^2$ is an embedding map such that for all i, j we have:

$$|d_X(x_i, x_j) - d'_X(x_i, x_j)| \leq \epsilon$$

Then for any $\gamma_1 : X \to \mathbb{R}$ that minimizes $\mathbb{I}_{M(X,d_X)}$, and $\gamma'_1 : X \to \mathbb{R}$ that minimizes $\mathbb{I}_{M(X,d'_X)}$, Theorem 6.65 bounds the average squared difference between $\|\gamma_1(x_i) - \gamma_1(x_j)\|$ and $\|\gamma'_1(x_j) - \gamma'_1(x_j)\|$.

6.3.7 Experiments in Functorial Recombination

One benefit of the functorial perspective on manifold learning is that it provides a natural way to produce new manifold learning algorithms by recombining the components of existing algorithms. Suppose we are able to express two existing manifold learning algorithms M_1, M_2 in this framework such that:

$$M_1 = L_1 \circ H_1$$
$$M_2 = L_2 \circ H_2$$

where H_1, H_2 are hierarchical clustering functors. Then we can use the compositionality of functors to define the manifold learning algorithms $L_2 \circ H_1$ or $L_1 \circ H_2$. We can use this procedure to combine the strengths of multiple algorithms in a way that preserves functoriality (and therefore also stability by Theorem 6.65).

For example, one of the largest contributors to UMAP's practical effectiveness is the density-aware local metric rescaling. This counteracts the non-uniformity of the data distribution (McInnes et al., 2018), and is captured in the *FuzzySimplex* functor. We can combine this strength with the information preserving properties of the *MDS* functor. That is, if we compose the *FuzzySimplex* functor from Proposition 6.60 with *MDS* we form the $\mathbf{Met}_{isom}$-manifold learning functor $MDS \circ FuzzySimplex$ that maps (X, d_X) to the embedding optimization problem $(X, m, \mathbb{I})$ where:

$$\mathbb{I}(\gamma) = \sum_{x_i, x_j \in X} (-\log(\alpha_{ij}) - \|\gamma(x_i) - \gamma(x_j)\|)^2$$

and α_{ij} is the strength of the fuzzy simplex that UMAP forms between x_i and x_j. We can express this algorithm in pseudocode as follows:

procedure MDS-FuzzySimplex($(X, d_X), m$)
    Initialize the $|X| \times |X|$ matrix D to all zeros
    $\forall i, j \leq |X|$
        $D[i, j] \leftarrow d_X(x_i, x_j) - \max(\min_{k=1\cdots|X|} d_X(x_i, x_k), \min_{k=1\cdots|X|} d_X(x_j, x_k))$
    $\gamma \leftarrow \min_{\gamma: X \to \mathbb{R}^m} \sum_{x_i, x_j \in X} (D[i, j] - \|\gamma(x_i) - \gamma(x_j)\|)^2$
    Return γ

For a more illustrative example, consider a DNA recombination task in which we attempt to match a string of DNA that has been repeatedly mutated back to the original string. One way to solve this task is to generate embeddings for each string of DNA and look at the nearest neighbors of the mutated string. We can simulate this task as follows:

1. Generate N original random sequences of DNA of length L (strings of "A", "C", "G", "T").

2. For each sequence, mutate the sequence M times to produce a mutation list, or a list of sequences which each start with an original DNA sequence and end with a final DNA sequence. There will be N total mutation lists, each of length M.

3. Collect each of the M sequences in each of these N mutation lists into a $N * M$ element finite metric space with Hamming distance (Hamming, 1950).

4. Use a manifold learning algorithm to build embeddings from each sequence in this metric space. Define the embedding distance between two sequences in this space to be the Euclidean distance between their embeddings.

5. Construct a set S_1 that contains the original sequences in each mutation list and a set S_2 that contains the final sequences in each mutation list.

6. For each sequence in S_2, use the embedding distance to compute that sequence's nearest neighbor in S_1.

7. Define the accuracy to be the percentage of sequences in S_2 for which the S_1-nearest neighbor is the original sequence in that sequences's mutation list.

A manifold learning algorithm that performs well on this task will need to take advantage of the intermediate mutation states to recognize that the first state and final state in a mutation list should be embedded as close together as possible. We can follow the procedure in Section 6.3.3 to adapt the metric multidimensional scaling algorithm $MDS \circ \mathcal{ML}$ (Section 6.3.5.1) into such an algorithm by swapping the $\mathcal{ML}$ overlapping hierarchical clustering functor for the $\mathcal{SL}$ functor to form the single linkage scaling functor:

Definition 6.66. *We define the single linkage scaling functor to be:*

$$MDS \circ \mathcal{SL} : \mathbf{Met} \to \mathbf{FLoss}$$

We describe the single linkage scaling functor with pseudocode in Algorithm 1.

Intuitively, single linkage scaling maps (X, d_X) to a loss function that corresponds to the optimization objective for metric multidimensional scaling where Euclidean distance is replaced with:

$$d_X^*(x, x') = \inf\{\delta \mid \exists x = x_1, x_2, \cdots, x_n = x' \in X, d_X(x_i, x_{i+1}) \leq \delta\}$$

Since this algorithm embeds points that are connected by a sequence in the original space as close together as possible, we expect single linkage scaling to outperform metric multidimensional scaling on this task. This is exactly what we see in Table 5. We also show the embeddings for each sequence in each list in Figure 16. All code from the experiments in this section can be found on GitHub (Shiebler, 2020b).

Algorithm 1 Single Linkage Scaling

procedure SINGLELINKAGESCALING($(X, d_X), m$)
  Initialize the $|X| \times |X|$ matrix D to all zeros
  $\forall x_i, x_j \in X$
    $D[i, j] \leftarrow \inf\{\delta \mid \exists x_i = x_1, x_2, \cdots, x_n = x_j \in X, d_X(x_k, x_{k+1}) \leq \delta\}$
  $\gamma \leftarrow \min_{\gamma : X \to \mathbb{R}^m} \sum_{x_i, x_j \in X} (D[i, j] - \|\gamma(x_i) - \gamma(x_j)\|)^2$
  Return γ

6.3.8 Closing Thoughts on Functorial Manifold Learning

In this section we have explored a categorical framework for manifold learning. By defining an algorithm as a functor from a category of metric spaces, we can explicitly express the kind of dataset transformations under which it is equivariant. We show that for many popular manifold learning algorithms, including metric multidimensional scaling and isomap, the optimization objective changes in a predictable way as we modify the metric space.

The functorial perspective also suggests a new strategy for exploratory data analysis with manifold learning. Since we can decompose manifold learning algorithms into two components (clustering and loss), we can examine how slight variations of the clustering

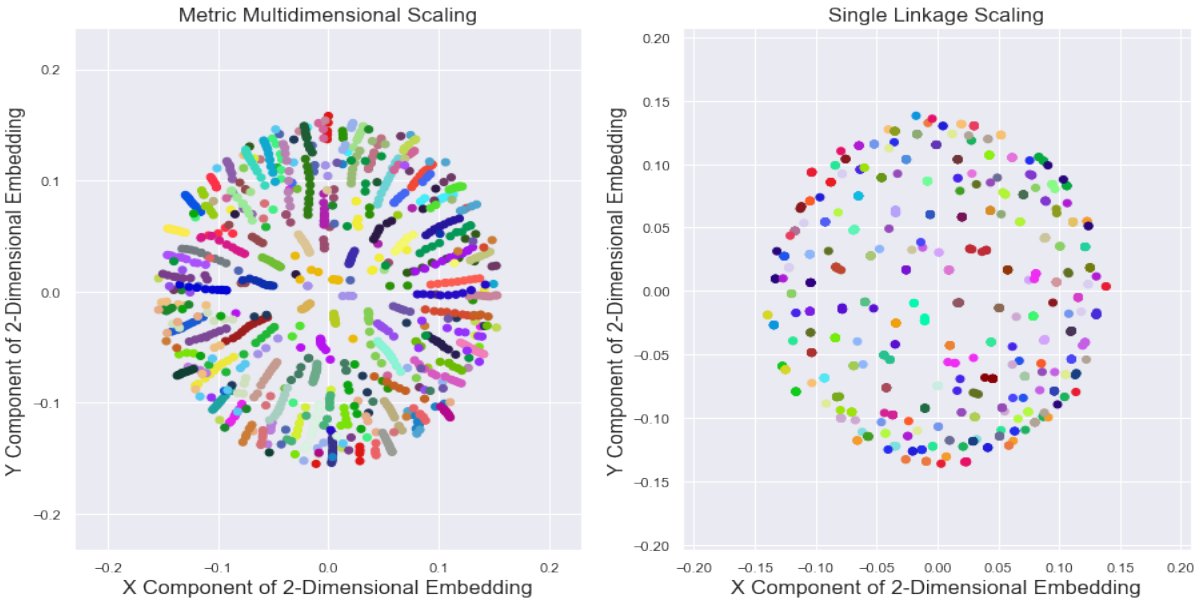


Figure 16: *Embeddings of DNA sequences from the DNA recombination task with* $L = 1000, N = 100, M = 10$. *Each color indicates a unique DNA sequence mutation list. Single linkage scaling (MDS* $\circ$ $\mathcal{SL}$*) on the right embeds sequences in the same mutation list more closely together than metric multidimensional scaling (MDS* $\circ$ $\mathcal{ML}$*) on the left.*

Algorithm	Accuracy $N = 100$ $M = 10$	Accuracy $N = 100$ $M = 20$	Accuracy $N = 200$ $M = 10$	Accuracy $N = 200$ $M = 20$
Metric Multidimensional Scaling Embedding Size 2	0.19 ($\pm$0.07)	0.12 ($\pm$0.06)	0.04 ($\pm$0.02)	0.02 ($\pm$0.02)
Single Linkage Scaling Embedding Size 2	0.77 ($\pm$0.08)	0.66 ($\pm$0.09)	0.32 ($\pm$0.06)	0.25 ($\pm$0.06)
Metric Multidimensional Scaling Embedding Size 5	0.96 ($\pm$0.04)	0.88 ($\pm$0.06)	0.27 ($\pm$0.06)	0.09 ($\pm$0.04)
Single Linkage Scaling Embedding Size 5	0.99 ($\pm$0.02)	0.89 ($\pm$0.06)	0.32 ($\pm$0.06)	0.12 ($\pm$0.04)

Table 5: *Accuracy on the DNA recombination task of the metric multidimensional scaling (MDS* $\circ$ $\mathcal{ML}$*) and single linkage scaling (MDS* $\circ$ $\mathcal{SL}$*) algorithms (higher numbers are better). The accuracy is the percent of the* N *mutation lists of length* M *for which the nearest neighbor of the last sequence in the mutation list among the set of all original DNA sequences is the first sequence in that mutation list. The reported numbers are means and two standard error confidence bounds. All DNA sequences are of length* $L = 1000$.

algorithm affect the learned embeddings. We show in Section 6.3.3 that every manifold learning functor $L \circ H$ lies on a spectrum of interconnectedness between $L \circ \mathcal{ML}$ and $L \circ \mathcal{SL}$, and we can form new algorithms by moving along this spectrum. For example, we see in Section 6.3.7 that replacing the $\mathcal{ML}$ functor with $\mathcal{SL}$ in the metric multidimensional scaling algorithm substantially changes the learned embeddings and improves performance on a DNA recombination task. There are also many algorithms that lie between these two options, including the k-Path Scaling and k-Vertex-Connected Scaling algorithms that we introduce in Section 6.3.5.

Another major benefit of expressing algorithms as functors is that functors preserve categorical properties like interleaving distance. This allows us to easily reason about the stability properties of both existing algorithms and new algorithms that we create by recombining functors. Other authors have used this strategy to prove stability properties of the homology of geometric filtered complexes (Chazal et al., 2014). In Section 6.3.6 we use this strategy to define bounds on how dataset noise affects optimization quality.

6.4 Extrapolating Clustering Functors with Colimits

In practice we often need to extrapolate a clustering to out of sample points. Suppose we have some finite metric space $(\mathbf{X}, d_{\mathbf{X}})$, finite subset $X \subseteq \mathbf{X}$, and a clustering of X. Our objective is to extrapolate this clustering of X to a clustering of $\mathbf{X}$.

Formally, suppose $\mathbf{D}_{\mathbf{X}} \subseteq \mathbf{Met}_{id}$ is the subcategory of $\mathbf{Met}_{id}$ (2.45) whose objects are all finite metric subspaces $(X, d_{\mathbf{X}})$ of $(\mathbf{X}, d_{\mathbf{X}})$. For any pair of subsets X, X' of $\mathbf{X}$ where $X \subseteq X'$ we have:

$$(X, d_{\mathbf{X}}) \leq_{\mathbf{D}_{\mathbf{X}}} (X', d_{\mathbf{X}})$$

Now consider an object $(X, d_{\mathbf{X}})$ in $\mathbf{D}_{\mathbf{X}}$ and a functor $F : \mathbf{D}_{\mathbf{X}} \to \mathbf{Cov}_{id}$ such that $\iota \circ F$ is an overlapping flat $\mathbf{D}_{\mathbf{X}}$-clustering functor. Our objective is to produce a cover of $(\mathbf{X}, d_{\mathbf{X}})$ by grouping the points in $\mathbf{X} - X$ into the sets in $F(X, d_{\mathbf{X}})$. Intuitively, we want to do this in a way such that if the points $x' \in X$ and $x \in \mathbf{X} - X$ would be placed into the same cluster in $F(X \cup \{x\}, d_{\mathbf{X}})$, then they will share a set in this cover of $(\mathbf{X}, d_{\mathbf{X}})$.

Proposition 6.67. *Suppose $\mathbf{D}_{X \cup \{\_\}}$ is the discrete subcategory of $\mathbf{D}_{\mathbf{X}}$ whose objects are all finite metric spaces $(X \cup \{x\}, d_{\mathbf{X}})$ where $x \in \mathbf{X} - X$. Consider the map*

$$F^* : \mathbf{D}_{X \cup \{\_\}} \to \mathbf{Cov}_{id}$$

that sends $(X \cup \{x\}, d_{\mathbf{X}})$ to the flagification of the following cover of $X \cup \{x\}$:

$$\{\{x\}\} \bigcup \left\{ \left\{ \begin{cases} S \cup \{x\} & \exists S' \in F(X \cup \{x\}, d_{\mathbf{X}}),\ S \cup \{x\} \subseteq S' \\ S & \text{else} \end{cases} \mid S \in F(X, d_{\mathbf{X}}) \right\} \right.$$

If $\iota : \mathbf{Cov}_{id} \hookrightarrow \mathbf{Cov}$ is the inclusion map then the map

$$\iota \circ F^* : \mathbf{D}_{X \cup \{\_\}} \to \mathbf{Cov}$$

is an overlapping flat $\mathbf{D}_{X \cup \{\_\}}$-clustering functor.

Proof. Since $\mathbf{D}_{X \cup \{\_\}}$ is discrete, we simply need to show that for any $(X \cup \{x\}, d_{\mathbf{X}})$ in $\mathbf{D}_{X \cup \{\_\}}$ we have that $F^*(X \cup \{x\}, d_{\mathbf{X}})$ is a nonnested flag cover of $X \cup \{x\}$.

First, note that there must exist some $S' \in F^*(X \cup \{x\}, d_{\mathbf{X}})$ that contains x since $F^*(X \cup \{x\}, d_{\mathbf{X}})$ is the flagification of a cover we can express as $\{x\} \cup \cdots$. Next, since $F(X, d_{\mathbf{X}})$ is a cover of X then any $x' \in X$ must be in a set in $F^*(X \cup \{x\}, d_{\mathbf{X}})$ as well. Therefore $F^*(X \cup \{x\}, d_{\mathbf{X}})$ is the flagification of a cover of $X \cup \{x\}$ and is therefore a nonnested flag cover. □

For any $x \in \mathbf{X} - X$ the cover $F^*(X \cup \{x\}, d_{\mathbf{X}})$ is different from $F(X, d_{\mathbf{X}})$ in one of two ways. Either:

$$F^*(X \cup \{x\}, d_{\mathbf{X}}) = \{\{x\}\} \cup F(X, d_{\mathbf{X}})$$

or for each $S \in F^*(X \cup \{x\}, d_{\mathbf{X}})$ there exists some $S' \in F(X, d_{\mathbf{X}})$ where:

$$S = S' \text{ or } S = S' \cup \{x\}$$

Proposition 6.68. *For any $x \in \mathbf{X} - X$ we have:*

$$\{\{x\}\} \cup F(X, d_{\mathbf{X}}) \leq_{\mathbf{Cov}_{id}} F^*(X \cup \{x\}, d_{\mathbf{X}}) \leq_{\mathbf{Cov}_{id}} F(X \cup \{x\}, d_{\mathbf{X}})$$

Proof. First note that for any set $S \in F(X, d_{\mathbf{X}})$ either S or $S \cup \{x\}$ are in

$$\left\{ \begin{cases} S \cup \{x\} & \exists S' \in F(X \cup \{x\}, d_{\mathbf{X}}),\ S \cup \{x\} \subseteq S' \\ S & else \end{cases} \mid S \in F(X, d_{\mathbf{X}}) \right\}$$

and therefore there exists a set in $F^*(X \cup \{x\}, d_{\mathbf{X}})$ that contains S. Next, since $F^*(X \cup \{x\}, d_{\mathbf{X}})$ is a cover of $X \cup \{x\}$ this implies that:

$$\{\{x\}\} \cup F(X, d_{\mathbf{X}}) \leq_{\mathbf{Cov}_{id}} F^*(X \cup \{x\}, d_{\mathbf{X}})$$

Next, consider some pair of points $x_1, x_2 \in X \cup \{x\}$ such that there exists some $S^* \in F^*(X \cup \{x\}, d_{\mathbf{X}})$ where $x_1 \in S^*$ and $x_2 \in S^*$. Since $F^*(X \cup \{x\}, d_{\mathbf{X}})$ is the flagification of the cover

$$\{\{x\}\} \bigcup \left\{ \begin{cases} S \cup \{x\} & \exists S' \in F(X \cup \{x\}, d_{\mathbf{X}}),\ S \cup \{x\} \subseteq S' \\ S & else \end{cases} \mid S \in F(X, d_{\mathbf{X}}) \right\}$$

there must be a set S^* in this cover such that $x_1, x_2, \in S^*$. There are two cases. First, if $S^* \in F(X, d_{\mathbf{X}})$ then since:

$$(X, d_{\mathbf{X}}) \leq_{\mathbf{D_X}} (X \cup \{x\}, d_{\mathbf{X}})$$

and $F : \mathbf{D_X} \to \mathbf{Cov}_{id}$ is a functor it must be that:

$$F(X, d_{\mathbf{X}}) \leq_{\mathbf{Cov}_{id}} F(X \cup \{x\}, d_{\mathbf{X}})$$

and therefore there exists some $S' \in F(X \cup \{x\}, d_{\mathbf{X}})$ such that $S^* \subseteq S'$. Second, if $S^* = S \cup \{x\}$ where $S \in F(X, d_{\mathbf{X}})$ then by the definition of F^* there must exist some $S' \in F(X \cup \{x\}, d_{\mathbf{X}})$ where $S \cup \{x\} \subseteq S'$. Since $F^*(X \cup \{x\}, d_{\mathbf{X}})$ and $F(X \cup \{x\}, d_{\mathbf{X}})$ are flag covers, this implies that:

$$F^*(X \cup \{x\}, d_{\mathbf{X}}) \leq_{\mathbf{Cov}_{id}} F(X \cup \{x\}, d_{\mathbf{X}})$$

□

We can think of F^* as simply adding each $x \in \mathbf{X} - X$ to certain sets in $F(X, d_{\mathbf{X}})$.

To stitch together these cluster assignments into a cover of $\mathbf{X}$, we can simply take the colimit of the functor $F^* : \mathbf{D}_{X \cup \{\_\}} \to \mathbf{Cov}_{id}$.

Proposition 6.69. *The colimit of the functor $F^* : \mathbf{D}_{X \cup \{\_\}} \to \mathbf{Cov}_{id}$ (Proposition 6.67) exists and is a nonnested flag cover of $(\mathbf{X}, d_{\mathbf{X}})$.*

Proof. Since $\mathbf{D}_{X \cup \{\_\}}$ is a discrete category and $\mathbf{Cov}_{id}$ is a preorder, the colimit of $F^* : \mathbf{D}_{X \cup \{\_\}} \to \mathbf{Cov}_{id}$ is simply the supremum of the set:

$$\{F^*(X \cup \{x\}, d_{\mathbf{X}}) \mid x \in \mathbf{X} - X\}$$

Now define $\mathcal{C}_{\mathbf{X}}$ to be the flagification of the following cover of $\mathbf{X}$:

$$\{S \mid \exists x \in \mathbf{X} - X,\ S \in F^*(X \cup \{x\}, d_{\mathbf{X}})\}$$

This is a nonnested flag cover of $\mathbf{X}$, and is therefore an object in $\mathbf{Cov}_{id}$. It is easy to see that for any $x \in \mathbf{X} - X$ we have that:

$$F^*(X \cup \{x\}, d_{\mathbf{X}}) \leq_{\mathbf{Cov}_{id}} \mathcal{C}_{\mathbf{X}}$$

Next, consider some other $\mathcal{C}'_{\mathbf{X}}$ in $\mathbf{Cov}_{id}$ such that for any $x \in \mathbf{X} - X$ we have that:

$$F^*(X \cup \{x\}, d_{\mathbf{X}}) \leq_{\mathbf{Cov}_{id}} \mathcal{C}'_{\mathbf{X}}$$

Consider also some pair of points $x_1, x_2 \in \mathbf{X}$. If $x_1 \notin X$ and $x_2 \notin X$ then for any $x \in \mathbf{X} - X$ and $S \in F^*(X \cup \{x\}, d_{\mathbf{X}})$ it cannot be that $x_1 \in S$ and $x_2 \in S$. This further implies that for any $S \in \mathcal{C}_{\mathbf{X}}$ it cannot be that $x_1 \in S$ and $x_2 \in S$.

Therefore if there exists some $S \in \mathcal{C}_{\mathbf{X}}$ where $x_1 \in S, x_2 \in S$ it must be that one of the following holds true:

1. $x_1 \in X$ and $x_2 \in X$. In this case for any choice of $x \in \mathbf{X} - X$ it must be that there exists some $S^* \in F^*(X \cup \{x\}, d_{\mathbf{X}})$ where $x_1 \in S^*, x_2 \in S^*$ and therefore there exists some $S' \in \mathcal{C}'_{\mathbf{X}}$ where $x_1 \in S', x_2 \in S'$.

2. Without loss of generality $x_1 \in X, x_2 \in \mathbf{X} - X$. In this case there exists some $S^* \in F^*(X \cup \{x_2\}, d_{\mathbf{X}})$ where $x_1 \in S^*, x_2 \in S^*$. Since:

   $$F^*(X \cup \{x_2\}, d_{\mathbf{X}}) \leq_{\mathbf{Cov}_{id}} \mathcal{C}'_{\mathbf{X}}$$

   this implies that there exists some $S' \in \mathcal{C}'_{\mathbf{X}}$ where $x_1 \in S', x_2 \in S'$.

We can therefore conclude that:

$$\mathcal{C}_{\mathbf{X}} \leq_{\mathbf{Cov}_{id}} \mathcal{C}'_{\mathbf{X}}$$

and therefore $\mathcal{C}_{\mathbf{X}}$ is the colimit of $F^* : \mathbf{D}_{X \cup \{\_\}} \to \mathbf{Cov}_{id}$. □

7 Supervised Learning and Generalization

A common problem in data science is "use this function defined over this small set to generate predictions over that larger set." Extrapolation, interpolation, statistical inference and forecasting all reduce to this problem. The Kan extension is a powerful tool in category theory that generalizes this notion.

In this Chapter we explore several applications of Kan extensions to data science. We begin by deriving a simple classification algorithm as a Kan extension and experimenting with this algorithm on real data. Next, we use the Kan extension to derive a procedure for learning clustering algorithms from labels and explore the performance of this procedure on real data. We then investigate how Kan extensions can be used to learn a general mapping from datasets of labeled examples to functions and to approximate a function in one class with a function in another class.

Some authors have begun to explore Kan extension structure in topological data analysis. For example, Bubenik et al. (2017) describe how three mechanisms for interpolating between persistence modules can be characterized as the left Kan extension, right Kan extension, and natural map from left to right Kan extension. Similarly, McCleary and Patel (2021) use Kan extensions to characterize deformations of filtrations. Furthermore, Botnan and Lesnick (2018) use Kan extensions to generalize stability results from block decomposable persistence modules to zigzag persistence modules and Curry (2013) uses Kan extensions to characterize persistent structures from the perspective of sheaf theory.

Other authors have explored the application of Kan extensions to databases. For example, in categorical formulations of relational database theory (Spivak and Wisnesky, 2015; Schultz and Wisnesky, 2017; Schultz et al., 2016), the left Kan extension can be used for data migration. Spivak and Wisnesky (2020) exploit the characterization of data migrations as Kan extensions to apply the chase algorithm from relational database theory to the general computation of the left Kan extension.

Our contributions in this section are as follows:

- We derive a simple classification algorithm as a Kan extension and demonstrate experimentally that this algorithm can learn to classify images.
- We use Kan extensions to derive a novel method for learning a clustering algorithm from labeled data and demonstrate experimentally that this method can learn to cluster images.
- We explore the structure of meta-supervised learning and use Kan extensions to derive supervised learning algorithms from sets of labeled datasets and trained functions.
- We use Kan extensions to characterize the process of approximating a function in one class with a function from another class.

All code can be found on GitHub (Shiebler, 2021b).

7.1 Applications of Kan Extensions

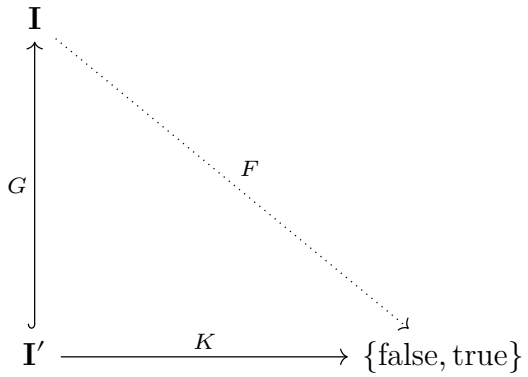
To start, recall the definitions of the left and right Kan extensions from section 2.1.2. In this section we will explore four applications of Kan extensions to generalization in machine learning:

- Section 7.2: Learn a classifier from a dataset of labeled examples.
- Section 7.3: Learn a mapping from metric spaces (X, d_X) to partitions of X.
- Section 7.4: Learn a mapping from datasets of labeled examples to functions.
- Section 7.5: Approximate a function in one class with a function from another class.

In each of these applications we first define categories $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and a functor $K : \mathbf{A} \to \mathbf{C}$ such that $\mathbf{A}$ is a subcategory of $\mathbf{B}$ and $G : \mathbf{A} \hookrightarrow \mathbf{B}$ is the inclusion functor. Then, we take the left and right Kan extensions $Lan_G K, Ran_G K$ of K along G and study their behavior. Intuitively, the more restrictive that $\mathbf{B}$ is (i.e. the more morphisms in $\mathbf{B}$) or the larger that $\mathbf{A}$ is (and therefore the more information that is stored in K) the more similar $Lan_G K, Ran_G K$ will be to each other.

7.2 Classification

We start with a simple application of Kan extensions to supervised learning. Suppose that $\mathbf{I}$ is a preorder, $\mathbf{I}' \subseteq \mathbf{I}$ is a subposet of $\mathbf{I}$, and $\{\text{false}, \text{true}\}$ is the two element preorder where false $<$ true. Suppose also that $K : \mathbf{I}' \to \{\text{false}, \text{true}\}$ is a mapping into $\{\text{false}, \text{true}\}$ and we would like to learn a monotonic function $F : \mathbf{I} \to \{\text{false}, \text{true}\}$ that approximates K on $\mathbf{I}'$. That is, K defines a finite training set of points $S = \{(x, K(x)) \mid x \in \mathbf{I}'\}$ from which we wish to learn a monotonic function $F : \mathbf{I} \to \{\text{false}, \text{true}\}$. Of course, it may not be possible to find a monotonic function that agrees with K on all the points in $\mathbf{I}'$.



If we treat $\mathbf{I}'$ as a discrete category, then K is a functor and we can solve this problem with the left and right Kan extensions of K along the inclusion functor $G : \mathbf{I}' \hookrightarrow \mathbf{I}$.

Theorem 7.1. *The left and right Kan extensions of $K : \mathbf{I}' \to \{false, true\}$ along the inclusion map $G : \mathbf{I}' \hookrightarrow \mathbf{I}$ are respectively:*

$$Lan_G K : \mathbf{I} \to \{false, true\} \qquad Ran_G K : \mathbf{I} \to \{false, true\}$$

$$Lan_G K(x) = \begin{cases} true & \exists x' \in \mathbf{I}', x' \leq x, K(x') = true \\ false & else \end{cases}$$

$$Ran_G K(x) = \begin{cases} false & \exists x' \in \mathbf{I}', x \leq x', K(x') = false \\ true & else \end{cases}$$

Proof. We first need to show that $Lan_G K, Ran_G K$ are functors. For any $x_1 \leq x_2 \in \mathbf{I}$ suppose that $Lan_G K(x_1) = \text{true}$. Then $\exists x' \in \mathbf{I}', x' \leq x_1, K(x') = \text{true}$. By transitivity we have $x' \leq x_2$, so:

$$Lan_G K(x_2) = \left(\begin{cases} \text{true} & \exists x' \in \mathbf{I}', x' \leq x_2, K(x') = \text{true} \\ \text{false} & \text{else} \end{cases} \right) = \text{true}$$

and $Lan_G K$ is therefore a functor.

Next, for any $x_1 \leq x_2 \in \mathbf{I}$ suppose that $Ran_G K(x_2) = \text{false}$. Then $\exists x' \in \mathbf{I}', x_2 \leq x', K(x') = \text{false}$. By transitivity we have $x_1 \leq x'$, so:

$$Ran_G K(x_1) = \left(\begin{cases} \text{false} & \exists x' \in \mathbf{I}', x_1 \leq x', K(x') = \text{false} \\ \text{true} & \text{else} \end{cases} \right) = \text{false}$$

and $Ran_G K$ is therefore a functor.

Next we will show that $Lan_G K$ is the left Kan extension of K along G. If for some $x' \in \mathbf{I}'$ we have that $K(x') = \text{true}$ then:

$$Lan_G K(x') = \left(\begin{cases} \text{true} & \exists x'' \in \mathbf{I}', x'' \leq x', K(x'') = \text{true} \\ \text{false} & \text{else} \end{cases} \right) = \text{true}$$

so we can conclude that $K \leq (Lan_G K \circ G)$. Now consider any other functor $M_L : \mathbf{I} \to \{\text{false}, \text{true}\}$ such that $\forall x' \in \mathbf{I}', K(x') \leq M_L(x')$. We must show that $\forall x \in \mathbf{I}, Lan_G K(x) \leq M_L(x)$. For some $x \in \mathbf{I}$ suppose $M_L(x) = \text{false}$. Then since M_L is a functor it must be that $\forall x' \in \mathbf{I}', x' \leq x, M_L(x') = \text{false}$. Since $K \leq (M_L \circ G)$, it must be that $\forall x' \in \mathbf{I}', x' \leq x, K(x') = \text{false}$. Therefore $Lan_G K(x) = \text{false}$.

Next we will show that $Ran_G K$ is the right Kan extension of K along G. If for some $x' \in \mathbf{I}'$ we have that $K(x') = \text{false}$ then:

$$Ran_G K(x') = \left(\begin{cases} \text{false} & \exists x'' \in \mathbf{I}', x' \leq x'', K(x'') = \text{false} \\ \text{true} & \text{else} \end{cases} \right) = \text{false}$$

so we can conclude that $(Ran_G K \circ G) \leq K$. Now consider any other functor $M_R : \mathbf{I} \to \{\text{false}, \text{true}\}$ such that $\forall x' \in \mathbf{I}', M_R(x') \leq K(x')$. We must show that $\forall x \in \mathbf{I}, M_R(x) \leq Ran_G K(x)$. For some $x \in \mathbf{I}$ suppose $M_R(x) = \text{true}$. Then since M_R is a functor it must be that $\forall x' \in \mathbf{I}', x \leq x', M_R(x') = \text{true}$. Since $(M_R \circ G) \leq K$, it must be that $\forall x' \in \mathbf{I}', x \leq x', K(x') = \text{true}$. Therefore $Ran_G K(x) = \text{true}$. □

In the extreme case that $Ob(\mathbf{I}') = \emptyset$, for $x \in \mathbf{I}$ we have that:

$$Lan_G K(x) = \left(\begin{cases} \text{true} & \exists x' \in \mathbf{I}', x' \leq x, K(x') = \text{true} \\ \text{false} & \text{else} \end{cases} \right) = \text{false}$$

$$Ran_G K(x) = \left(\begin{cases} \text{false} & \exists x' \in \mathbf{I}', x \leq x', K(x') = \text{false} \\ \text{true} & \text{else} \end{cases} \right) = \text{true}$$

Similarly, in the extreme case that $Ob(\mathbf{I}') = Ob(\mathbf{I})$ we have by the functoriality of K that for $x \in \mathbf{I}$ both of the following hold if and only if $K(x) = \text{true}$.

$$\exists x' \in \mathbf{I}', x' \leq x, K(x') = \text{true} \qquad \nexists x' \in \mathbf{I}', x \leq x', K(x') = \text{false}$$

Therefore in this extreme case we have:

$$Lan_G K(x) = Ran_G K(x) = K(x)$$

Now suppose that $\mathbf{I}'$ contains at least one x' such that $K(x') = \text{true}$ and at least one x' such that $K(x') = \text{false}$. In this case $Lan_G K$ and $Ran_G K$ split $\mathbf{I}$ into three regions: a region where both map all points to false, a region where both map all points to true, and a disagreement region. Note that $Ran_G K$ has no false positives on $\mathbf{I}'$ and $Lan_G K$ has no false negatives on $\mathbf{I}'$.

For example, suppose $\mathbf{I} = \mathbb{R}, \mathbf{I}' = \{1, 2, 3, 4\}$ and we have:

$$K(1) = \text{false} \qquad K(2) = \text{false} \qquad K(3) = \text{true} \qquad K(4) = \text{true}$$

Then we have that:

$$Lan_G K(x) = \left(\begin{cases} \text{true} & \exists x' \in \mathbf{I}', x' \leq x, K(x') = \text{true} \\ \text{false} & \text{else} \end{cases} \right) = \left(\begin{cases} \text{true} & x \geq 3 \\ \text{false} & \text{else} \end{cases} \right)$$

$$Ran_G K(x) = \left(\begin{cases} \text{false} & \exists x' \in \mathbf{I}', x \leq x', K(x') = \text{false} \\ \text{true} & \text{else} \end{cases} \right) = \left(\begin{cases} \text{true} & x > 2 \\ \text{false} & \text{else} \end{cases} \right)$$

In this case the disagreement region for $Lan_G K, Ran_G K$ is $(2, 3)$ and for any $x \in (2, 3)$ we have $Lan_G K(x) < Ran_G K(x)$.

As another example, suppose $\mathbf{I} = \mathbb{R}, \mathbf{I}' = \{5, 6, 7, 8\}$ and we have:

$$K(5) = \text{false} \qquad K(6) = \text{true} \qquad K(7) = \text{false} \qquad K(8) = \text{true}$$

Then we have that:

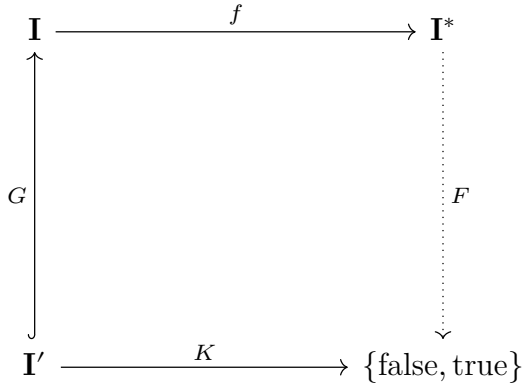
$$Lan_G K(x) = \left(\begin{cases} \text{true} & \exists x' \in \mathbf{I}', x' \leq x, K(x') = \text{true} \\ \text{false} & \text{else} \end{cases} \right) = \left(\begin{cases} \text{true} & x \geq 6 \\ \text{false} & \text{else} \end{cases} \right)$$

$$Ran_G K(x) = \left(\begin{cases} \text{false} & \exists x' \in \mathbf{I}', x \leq x', K(x') = \text{false} \\ \text{true} & \text{else} \end{cases} \right) = \left(\begin{cases} \text{true} & x > 7 \\ \text{false} & \text{else} \end{cases} \right)$$

In this case the disagreement region for $Lan_G K, Ran_G K$ is $[6, 7]$ and for any $x \in [6, 7]$ we have $Ran_G K(x) < Lan_G K(x)$.

While this approach is effective for learning very simple mappings, there are many choices of K for which $Lan_G K, Ran_G K$ do not approximate K particularly well on $\mathbf{I}'$ and therefore the disagreement region is large. In such a situation we can use a similar strategy to the one leveraged by kernel methods (Hofmann et al., 2008) and transform $\mathbf{I}$ to minimize the size of the disagreement region.

That is, we choose a preorder $\mathbf{I}^*$ and transformation $f : \mathbf{I} \to \mathbf{I}^*$ such that the size of the disagreement region for $Lan_{f \circ G} K \circ f, Ran_{f \circ G} K \circ f$ is minimized.



For example, if $\mathbf{I}^* = \mathbb{R}^a$ we can choose f to minimize the following loss:

Definition 7.2. *Suppose we have a set $\mathbf{I}' \subseteq \mathbf{I}$ and function $K : \mathbf{I}' \to \{false, true\}$ such that:*

$$\exists x', x'' \in \mathbf{I}', K(x') = true, K(x'') = false$$

Then the ordering loss l maps a function $f : \mathbf{I} \to \mathbb{R}^a$ to an approximation of the size of the disagreement region for $Lan_{f \circ G} K \circ f, Ran_{f \circ G} K \circ f$. Formally, we define the ordering loss l to be:

$$l : (\mathbf{I} \to \mathbb{R}^a) \to \mathbb{R}$$

$$l(f) = \sum_{i \leq a} \max(0,\ \max\{f(x)[i] \ \mid \ x \in \mathbf{I}', K(x) = false\} - \min\{f(x)[i] \ \mid \ x \in \mathbf{I}', K(x) = true\})$$

where $f(x)[i]$ is the ith component of the vector $f(x) \in \mathbb{R}^a$.

We can show that minimizing the ordering loss l will also minimize the size of the disagreement region:

Proposition 7.3. *The ordering loss l (Definition 7.2) is nonnegative and is only equal to 0 when $\forall x \in \mathbf{I}'$ we have:*

$$K(x) = (Lan_{f \circ G} K \circ f)(x) = (Ran_{f \circ G} K \circ f)(x)$$

Proof. First note that l must be nonnegative since each term can be expressed as $\max(0, \_)$. Next, suppose that $l(f) = 0$. Then it must be that for any $x_0, x_1 \in \mathbf{I}'$ such that $K(x_0) = \text{false}, K(x_1) = \text{true}$ we have that $f(x_0) \leq f(x_1)$. As a result, for any $x \in \mathbf{I}'$ there can only exist some $x' \in \mathbf{I}'$ where $f(x) \leq f(x'), K(x') = \text{false}$ when $K(x) = \text{false}$. Similarly, there can only exist some $x' \in \mathbf{I}'$ where $f(x') \leq f(x), K(x') = \text{true}$ when $K(x) = \text{true}$. Therefore:

$$K(x) = (Lan_{f \circ G} K \circ f)(x) = (Ran_{f \circ G} K \circ f)(x)$$

□

It is relatively straightforward to minimize the ordering loss with an optimizer like subgradient descent (Boyd et al., 2003). For example, we can implement the ordering loss in Tensorflow (Abadi et al., 2015) as follows:

```
import numpy as np
import tensorflow as tf

def get_ordering_loss(model, X, y):
    # model: Tensorflow sequential model
    # X: 2D numpy float array in which each row is a feature vector
    # y: 1D numpy boolean array of labels
    X_false = X[np.logical_not(np.array(y, dtype=bool))]
```

```
    false_preds = tf.transpose(model(X_false))

    X_true = X[np.array(y, dtype=bool)]
    true_preds = tf.transpose(model(X_true))

    return tf.reduce_sum(tf.math.maximum(0,
        tf.math.reduce_max(false_preds), axis=1) -
        tf.math.reduce_min(true_preds), axis=1)
```

In Table 6 we demonstrate that we can use this strategy to distinguish between the "T-shirt" (false) and "shirt" (true) categories in the Fashion MNIST dataset (Section 2.5). Samples in this dataset have 784 features (pixels), so we train a simple linear model $f : \mathbb{R}^{784} \to \mathbb{R}^{10}$ with Adam (Kingma and Ba, 2014) to minimize the ordering loss $l(f)$ over a training set that contains 90% of samples in the dataset. We then evaluate the performance of the left Kan classifer $Lan_{f \circ G} K \circ f$ and the right Kan classifier $Ran_{f \circ G} K \circ f$ over both this training set and a testing set that contains the remaining 10% of the dataset. We look at two metrics over both sets: the true positive rate (Definition 2.73) and the true negative rate (Definition 2.74).

As we would expect from the definition of Kan extensions, the left Kan classifier $Lan_{f \circ G} K \circ f$ has no false negatives and the right Kan classifier $Ran_{f \circ G} K \circ f$ has no false positives on the training set. The metrics on the testing set are in line with our expectations as well: the left Kan classifier has a higher true positive rate and the right Kan classifier has a higher true negative rate.

Interestingly, on this problem the right Kan classifier appears to perform better overall than the left Kan classifier. In particular, the true negative rate of the left Kan classifier is substantially worse than the true negative rate of the right Kan classifier, whereas the true positive rate of the right Kan classifier is only somewhat worse than the true positive rate of the left Kan classifier.

Model	Dataset	True Positive Rate	True Negative Rate
Left Kan Classifier	Training	1.000 ($\pm$0.000)	0.612 ($\pm$0.042)
Right Kan Classifier	Training	0.705 ($\pm$0.035)	1.000 ($\pm$0.000)
Left Kan Classifier	Testing	0.815 ($\pm$0.020)	0.593 ($\pm$0.044)
Right Kan Classifier	Testing	0.691 ($\pm$0.044)	0.837 ($\pm$0.026)

Table 6: *True positive rate (Definition 2.73) and true negative rate (Definition 2.74) of the left Kan classifier $Lan_{f \circ G} K \circ f$ and the right Kan classifier $Ran_{f \circ G} K \circ f$ where f is a linear map trained to minimize the ordering loss $l(f)$ (Definition 7.2) on the Fashion-MNIST "T-shirt" vs "shirt" task (Xiao et al., 2017). We run a bootstrap experiment by repeatedly selecting* 9000 *training samples and* 1000 *testing samples, running the training procedure, and computing true positive rate and true negative rate metrics. Mean and two standard error confidence bounds from* 10 *such bootstrap iterations are shown. The code for this experiment is on GitHub (Shiebler, 2021b).*

7.3 Clustering with Supervision

Clustering algorithms allow us to group points in a dataset together based on some notion of similarity between them. Formally, we can consider a clustering algorithm as mapping

a finite metric space (X, d_X) to a partition of X.

In most applications of clustering the points in the metric space (X, d_X) are grouped together based solely on the distances between the points and the rules embedded within the clustering algorithm itself. This is an unsupervised clustering strategy since no labels or supervision influence the algorithm output. For example, agglomerative clustering algorithms like HDBSCAN (McInnes and Healy, 2017) and single linkage partition points in X based on graphs formed from the points (vertices) and distances (edges) in (X, d_X).

However, there are some circumstances under which we have a few ground truth examples of pre-clustered training datasets and want to learn an algorithm that can cluster new data as similarly as possible to these ground truth examples. We can define the supervised clustering problem as follows. Given a collection of tuples

$$S = \{(X_1, d_{X_1}, \mathbb{P}_{X_1}), (X_2, d_{X_2}, \mathbb{P}_{X_2}), \cdots, (X_n, d_n, \mathbb{P}_{X_n})\}$$

where each (X_i, d_{X_i}) is a finite metric space and $\mathbb{P}_{X_i}$ is a partition of X_i, we would like to learn a general function f that maps a finite metric space (X, d_X) to a partition $\mathbb{P}_X$ of X such that for each $(X_i, d_{X_i}, \mathbb{P}_{X_i}) \in S$ the difference between $f(X_i, d_{X_i})$ and $\mathbb{P}_{X_i}$ is small.

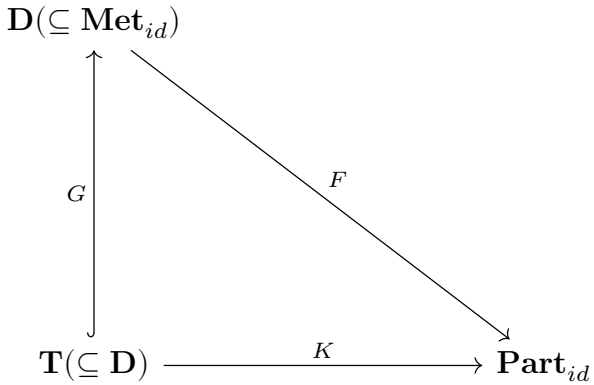
We can now frame this objective in terms of clustering functors. Recall that the categories $\mathbf{Met}_{id}$ (Definition 2.45) and $\mathbf{Part}_{id}$ (Definition 2.67) are the restrictions of $\mathbf{Met}$ (Definition 2.39) and $\mathbf{Part}$ (Definition 2.66) to preorders whose morphisms are limited to inclusion maps $\iota(x) = x$. Suppose $\iota : \mathbf{Part}_{id} \hookrightarrow \mathbf{Part}$ is the inclusion functor. Then given a subcategory $\mathbf{D} \subseteq \mathbf{Met}_{id}$ (Definition 2.45), a discrete subcategory $\mathbf{T} \subseteq \mathbf{D}$, and a functor $K : \mathbf{T} \to \mathbf{Part}_{id}$ such that:

$$\iota \circ K : \mathbf{T} \to \mathbf{Part}$$

is a $\mathbf{T}$-clustering functor (Definition 6.1), find the best functor $F : \mathbf{D} \to \mathbf{Part}_{id}$ such that:

$$\iota \circ F : \mathbf{D} \to \mathbf{Part}$$

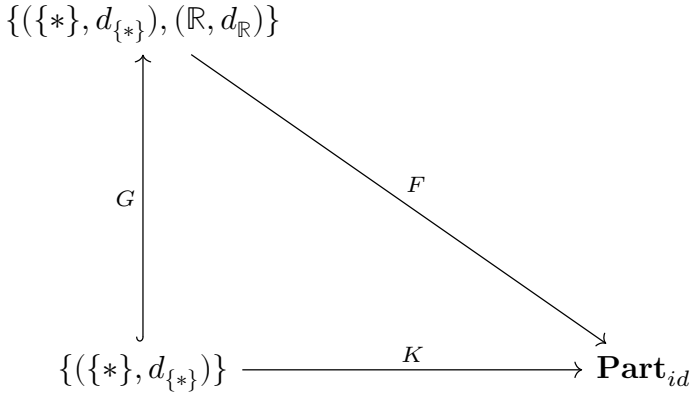
is a nonoverlapping flat $\mathbf{D}$-clustering functor and $F \circ G = K$ where $G : \mathbf{T} \hookrightarrow \mathbf{D}$ is the inclusion functor.



Intuitively, $Ob(\mathbf{T})$ is the set of unlabelled training samples, K defines the labels on these training samples, and $Ob(\mathbf{D})$ is the set of testing samples.

We would like to use the Kan extensions of K along G to find this best clustering functor. However, these Kan extensions are not guaranteed to be clustering functors. For

example, consider the case in which $\mathbf{T}$ is the discrete category that contains the single-element metric space as its only object and $\mathbf{D}$ is the discrete category that contains two objects: the single-element metric space and $\mathbb{R}$ equipped with the Euclidean distance metric [3].



Since $\mathbf{D}$ is a discrete category, the behavior of K on $(\{*\}, d_{\{*\}})$ will not affect the behavior of the left and right Kan extensions of K along G on $(\mathbb{R}, d_{\mathbb{R}})$. The left Kan extension of K along G will always map $(\mathbb{R}, d_{\mathbb{R}})$ to the initial object of $\mathbf{Part}_{id}$ (the empty set). That is, the left Kan extension does not satisfy the conditions of Definition 6.1 since it does not act as the identity on the underlying set $\mathbb{R}$. Futhermore, the right Kan extension of K along G will not exist because $\mathbf{Part}_{id}$ does not have a terminal object.

In order to solve this problem with Kan extensions we need to add a bit more structure. Suppose $Ob(\mathbf{D})$ is the discrete category with the same objects as $\mathbf{D}$ and define the following:

Definition 7.4. *The functor $K_L : Ob(\mathbf{D}) \to \mathbf{Part}_{id}$ is equal to K on $\mathbf{T}$ and maps each object (X, d_X) in $Ob(\mathbf{D}) - Ob(\mathbf{T})$ to $(X, \{\{x\} \mid x \in X\})$.*

Definition 7.5. *The functor $K_R : Ob(\mathbf{D}) \to \mathbf{Part}_{id}$ is equal to K on $\mathbf{T}$ and maps each object (X, d_X) in $Ob(\mathbf{D}) - Ob(\mathbf{T})$ to $(X, \{X\})$.*

Intuitively, K_L and K_R are extensions of K to all of the objects in $\mathbf{D}$. For any metric space $(X, d_X) \in \mathbf{Met}_{id}$ not in $Ob(\mathbf{T})$ the functor K_L maps (X, d_X) to the finest possible partition of X and K_R maps (X, d_X) to the coarsest possible partition of X.

Suppose we go back to the previous example in which $\mathbf{T}$ is the discrete category containing only the single-element metric space and $\mathbf{D}$ is the discrete category containing both the single-element metric space and $(\mathbb{R}, d_{\mathbb{R}})$. Since:

$$K_L(\mathbb{R}, d_{\mathbb{R}}) = (\mathbb{R}, \{\{x\} \mid x \in \mathbb{R}\})$$

the left Kan extension of K_L along the inclusion $G : Ob(\mathbf{D}) \hookrightarrow \mathbf{D}$ must map $(\mathbb{R}, d_{\mathbb{R}})$ to the $\leq_{\mathbf{Part}_{id}}$-smallest $(X, \mathbb{P}_X)$ such that:

$$(\mathbb{R}, \{\{x\} \mid x \in \mathbb{R}\}) \leq_{\mathbf{Part}_{id}} (X, \mathbb{P}_X)$$

which is $(X, \mathbb{P}_X) = (\mathbb{R}, \{\{x\} \mid x \in \mathbb{R}\})$. Similarly, since:

$$K_R(\mathbb{R}, d_{\mathbb{R}}) = (\mathbb{R}, \{\mathbb{R}\})$$

[3] This counterexample due to Sam Staton

the right Kan extension of K_R along the inclusion $G : Ob(\mathbf{D}) \hookrightarrow \mathbf{D}$ must map $(\mathbb{R}, d_{\mathbb{R}})$ to the $\leq_{\mathbf{Part}_{id}}$-largest $(X, \mathbb{P}_X)$ such that:

$$(X, \mathbb{P}_X) \leq_{\mathbf{Part}_{id}} (\mathbb{R}, \{\mathbb{R}\})$$

which is $(X, \mathbb{P}_X) = (\mathbb{R}, \{\mathbb{R}\})$. We can apply the same logic to the behavior of the Kan extensions on the single-element metric space as well, so the composition of:

$$\iota : \mathbf{Part}_{id} \hookrightarrow \mathbf{Part}$$

to either Kan extension yields a nonoverlapping flat $\mathbf{D}$-clustering functor.

We can now build on this perspective to construct optimal clustering functor extensions of K.

Proposition 7.6. *Consider the map $Lan_G K_L : \mathbf{D} \to \mathbf{Part}_{id}$ that acts as the identity on morphisms and sends the metric space $(X, d_X) \in \mathbf{D}$ to the partition of X defined by the transitive closure of the relation R where for $x_1, x_2 \in X$ we have $x_1 \, R \, x_2$ if and only if there exists some metric space $(X', d_{X'}) \in \mathbf{T}$ where:*

$$(X', d_{X'}) \leq_{\mathbf{D}} (X, d_X)$$

and x_1, x_2 are in the same cluster in $K(X', d_{X'})$. The map:

$$\iota \circ Lan_G K_L : \mathbf{D} \to \mathbf{Part}$$

is a nonoverlapping flat $\mathbf{D}$-clustering functor.

Proof. $Lan_G K_L$ trivially acts as the identity on morphisms and underlying sets and preserves composition and identity so we simply need to show that when:

$$(X, d_X) \leq_{\mathbf{D}} (Y, d_Y)$$

then

$$Lan_G K_L(X, d_X) \leq_{\mathbf{Part}_{id}} Lan_G K_L(Y, d_Y)$$

Suppose there exists some $x, x^* \in X$ in the same cluster in $Lan_G K_L(X, d_X)$. Then by the definition of $Lan_G K_L$ there must exist some sequence

$$(X_1, d_{X_1}), (X_2, d_{X_2}), \cdots, (X_n, d_{X_n}) \in \mathbf{T}$$

where $x \in X_1, x^* \in X_n$ and each:

$$(X_i, d_{X_i}) \leq_{\mathbf{D}} (X, d_X)$$

as well as some sequence

$$x_1, x_2, \cdots, x_{n-1}, \text{ such that } x_i \in X_i, x_i \in X_{i+1}$$

where the pair (x, x_1) is in the same cluster in $K(X_1, d_{X_1})$, the pair (x_{n-1}, x^*) is in the same cluster in $K(X_n, d_{X_n})$, and for each $1 < i < n$ the pair (x_{i-1}, x_i) is in the same cluster in $K(X_i, d_{X_i})$. Since it must be that each:

$$(X_i, d_{X_i}) \leq_{\mathbf{D}} (Y, d_Y)$$

as well then by the definition of $Lan_G K_L$ it must be that x, x^* are in the same cluster in $Lan_G K_L(Y, d_Y)$. □

Proposition 7.7. *Consider the map* $Ran_G K_R : \mathbf{D} \to \mathbf{Part}_{id}$ *that acts as the identity on morphisms and sends the metric space* $(X, d_X) \in \mathbf{D}$ *to the partition of* X *defined by the transitive closure of the relation* R *where for* $x_1, x_2 \in X$ *we have* $x_1 \, R \, x_2$ *if and only if there exists no metric space* $(X', d_{X'}) \in \mathbf{T}$ *where:*

$$(X, d_X) \leq_{\mathbf{D}} (X', d_{X'})$$

and x_1, x_2 *are in different clusters in* $K(X', d_{X'})$*. The map:*

$$\iota \circ Ran_G K_R : \mathbf{D} \to \mathbf{Part}$$

is a nonoverlapping flat $\mathbf{D}$*-clustering functor.*

Proof. $Ran_G K_R$ trivially acts as the identity on morphisms and underlying sets and preserves composition and identity so we simply need to show that when:

$$(X, d_X) \leq_{\mathbf{D}} (Y, d_Y)$$

then:

$$Ran_G K_R(X, d_X) \leq_{\mathbf{Part}_{id}} Ran_G K_R(Y, d_Y)$$

Suppose the points $x, x^* \in X$ are in the same cluster in $Ran_G K_R(X, d_X)$. Then by the definition of $Ran_G K_R$ there cannot be any $(X', d_{X'})$ in $\mathbf{T}$ such that:

$$(X, d_X) \leq_{\mathbf{D}} (X', d_{X'})$$

and x, x^* are in different clusters in $Ran_G K_R(X', d_{X'})$. By transitivity this implies that there cannot be any $(X'', d_{X''})$ in $\mathbf{T}$ such that:

$$(Y, d_Y) \leq_{\mathbf{D}} (X'', d_{X''})$$

and x, x^* are in different clusters in $Ran_G K_R(X'', d_{X''})$. By the definition of $Ran_G K_R$ the points x, x^* must therefore be in the same cluster in $Ran_G K_R(Y, d_Y)$. □

We can also make the following claim:

Proposition 7.8. *Suppose there exists some functor* $F : \mathbf{D} \to \mathbf{Part}_{id}$ *such that the composition* $\iota \circ F : \mathbf{D} \to \mathbf{Part}$ *is a nonoverlapping flat* $\mathbf{D}$*-clustering functor and* $F \circ G = K$*. Then for* $(X, d_X) \in \mathbf{T}$ *we have that:*

$$F(X, d_X) = K(X, d_X) = Lan_G K_L(X, d_X) = Ran_G K_R(X, d_X)$$

Proof. Since each of:

$$\iota \circ F : \mathbf{D} \to \mathbf{Part}$$
$$\iota \circ Ran_G K_R : \mathbf{D} \to \mathbf{Part}$$
$$\iota \circ Lan_G K_L : \mathbf{D} \to \mathbf{Part}$$

are $\mathbf{D}$-clustering functors we simply need to prove that all three functors generate the same partition of X for any input $(X, d_X) \in \mathbf{T}$.

Consider some $(X, d_X) \in \mathbf{T}$ and two points $x, x^* \in X$. Suppose x, x^* are in different clusters in

$$K(X, d_X) = F(X, d_X)$$

. Then since F is a $\mathbf{D}$-clustering functor it must be that for any sequence

$$(X_1, d_{X_1}), (X_2, d_{X_2}), \cdots, (X_n, d_{X_n}) \in \mathbf{T}$$

where $x \in X_1, x^* \in X_n$ and each:

$$(X_i, d_{X_i}) \leq_{\mathbf{D}} (X, d_X)$$

and any sequence

$$x_1, x_2, \cdots, x_{n-1}, \text{ such that } x_i \in X_i, x_i \in X_{i+1}$$

one of the following must be true:

- The pair (x, x_1) are in different clusters in $F(X_1, d_{X_1})$
- The pair (x_{n-1}, x^*) are in different clusters in $F(X_n, d_{X_n})$
- For some $1 < i < n$ the pair (x_{i-1}, x_i) are in different clusters in $F(X_i, d_{X_i})$

This implies that in $Lan_G K_L(X, d_X)$ the points x, x^* must be in different clusters. Similarly, since $(X, d_X) \leq_{\mathbf{D}} (X, d_X)$, by Proposition 7.7 it must be that x, x^* are in different clusters in $Ran_G K_R(X, d_X)$.

Now suppose x, x^* are in the same cluster in:

$$K(X, d_X) = F(X, d_X)$$

Since $(X, d_X) \leq_{\mathbf{D}} (X, d_X)$, by Proposition 7.6 it must be that x, x^* are in the same cluster in $Lan_G K_L(X, d_X)$. Similarly, since F is a $\mathbf{D}$-clustering functor there cannot exist any metric space $(X', d_{X'}) \in \mathbf{T}$ where:

$$(X, d_X) \leq_{\mathbf{D}} (X', d_{X'})$$

and x, x^* are in different clusters in:

$$K(X', d_{X'}) = F(X', d_{X'})$$

Therefore x, x^* are in the same cluster in $Ran_G K_R(X, d_X)$. □

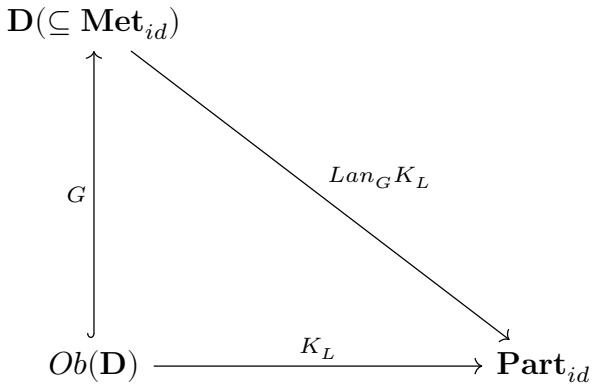
We can now put everything together and construct $Lan_G K_L, Ran_G K_R$ as Kan extensions.

Theorem 7.9. *Suppose there exists some functor $F : \mathbf{D} \to \mathbf{Part}_{id}$ such that*

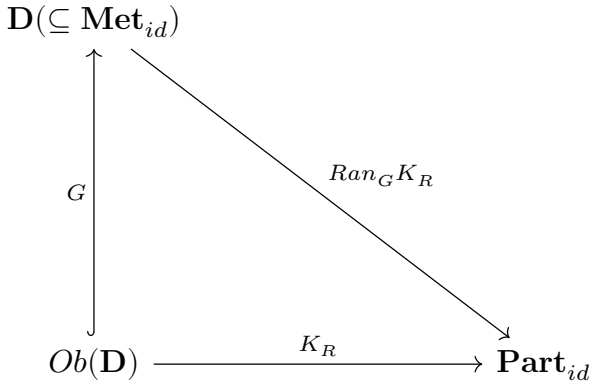
$$\iota \circ F : \mathbf{D} \to \mathbf{Part}$$

is a nonoverlapping flat $\mathbf{D}$-clustering functor and $F \circ G = K$.

Then $Lan_G K_L : \mathbf{D} \to \mathbf{Part}_{id}$ (Proposition 7.6) is the left Kan extension of $K_L : Ob(\mathbf{D}) \to \mathbf{Part}_{id}$ along the inclusion functor $G : Ob(\mathbf{D}) \hookrightarrow \mathbf{D}$.



In addition $Ran_G K_R : \mathbf{D} \to \mathbf{Part}_{id}$ *(Proposition 7.7) is the right Kan extension of* $K_R : Ob(\mathbf{D}) \to \mathbf{Part}_{id}$ *along the inclusion functor* $G : Ob(\mathbf{D}) \hookrightarrow \mathbf{D}$.



Proof. To start, note that Proposition 7.8 implies that for any $(X, d_X) \in \mathbf{T}$ we have:

$$Lan_G K_L(X, d_X) = K(X, d_X) = Ran_G K_R(X, d_X)$$

By the definition of K_L, K_R we can therefore conclude that for any $(X, d_X) \in \mathbf{D}$ we have:

$$K_L(X, d_X) \leq_{\mathbf{Part}_{id}} Lan_G K_L(X, d_X)$$
$$Ran_G K_R(X, d_X) \leq_{\mathbf{Part}_{id}} K_R(X, d_X)$$

Next, consider any functor $M_L : \mathbf{D} \to \mathbf{Part}_{id}$ such that for all $(X, d_X) \in \mathbf{D}$ we have:

$$K_L(X, d_X) \leq_{\mathbf{Part}_{id}} (M_L \circ G)(X, d_X)$$

We must show that for any $(X, d_X) \in \mathbf{D}$ we have:

$$Lan_G K_L(X, d_X) \leq_{\mathbf{Part}_{id}} M_L(X, d_X)$$

To start, note that for any $x, x^* \in X$ that are in the same cluster in $Lan_G K_L(X, d_X)$ by the definition of $Lan_G K_L$ there must exist some sequence:

$$(X_1, d_{X_1}), (X_2, d_{X_2}), \cdots, (X_n, d_{X_n}) \in \mathbf{T}$$

where $x \in X_1, x^* \in X_n$ and each:

$$(X_i, d_{X_i}) \leq_{\mathbf{D}} (X, d_X)$$

as well as some sequence

$$x_1, x_2, \cdots, x_{n-1}, \text{ such that } x_i \in X_i, x_i \in X_{i+1}$$

where the pair (x, x_1) is in the same cluster in $K_L(X_1, d_{X_1})$, the pair (x_{n-1}, x^*) is in the same cluster in $K_L(X_n, d_{X_n})$, and for each $1 < i < n$ the pair (x_{i-1}, x_i) is in the same cluster in $K_L(X_i, d_{X_i})$. Now since for each (X_i, d_{X_i}) in this sequence we have that:

$$K_L(X_i, d_{X_i}) \leq_{\mathbf{Part}_{id}} M_L(X_i, d_{X_i})$$

it must be that the pair (x, x_1) is in the same cluster in $M_L(X_1, d_{X_1})$, the pair (x_{n-1}, x^*) is in the same cluster in $M_L(X_n, d_{X_n})$, and for each $1 < i < n$ the pair (x_{i-1}, x_i) is in the same cluster in $M_L(X_i, d_{X_i})$.

Since M_L is a functor, it must therefore be that the pair x, x^* is in the same cluster in $M_L(X, d_X)$ and therefore:

$$Lan_G K_L(X, d_X) \leq_{\mathbf{Part}_{id}} M_L(X, d_X)$$

Next, consider any functor $M_R : \mathbf{D} \to \mathbf{Part}_{id}$ such that for all (X, d_X) in $\mathbf{D}$:

$$(M_R \circ G)(X, d_X) \leq_{\mathbf{Part}_{id}} K_R(X, d_X)$$

We must show that for any (X, d_X) in $\mathbf{D}$ we have:

$$M_R(X, d_X) \leq_{\mathbf{Part}_{id}} Ran_G K_R(X, d_X)$$

To start, note that for any $x, x^* \in X$ such that x, x^* are not in the same cluster in $Ran_G K_R(X, d_X)$ by the definition of $Ran_G K_R$ there must exist some:

$$(X', d_{X'}) \in \mathbf{D}, (X, d_X) \leq_{\mathbf{D}} (X', d_{X'})$$

where x, x^* are not in the same cluster in $K_R(X', d_{X'})$. Now since:

$$M_R(X', d_{X'}) \leq_{\mathbf{Part}_{id}} K_R(X', d_{X'})$$

it must be that x, x^* are not in the same cluster in $M_R(X', d_{X'})$. Since M_R is a functor, we have:

$$M_R(X, d_X) \leq_{\mathbf{Part}_{id}} M_R(X', d_{X'})$$

so x, x^* are also not in the same cluster in $M_R(X, d_X)$ and therefore:

$$M_R(X, d_X) \leq_{\mathbf{Part}_{id}} Ran_G K_R(X, d_X)$$

□

We will call $Lan_G K_L$ the left Kan supervised clustering map and $Ran_G K_R$ the right Kan supervised clustering map.

When $Ob(\mathbf{T}) = \emptyset$ we have for any $(X, d_X) \in \mathbf{D}$ that:

$$Lan_G K_L(X, d_X) = K_L(X, d_X) = (X, \{\{x\} \mid x \in X\})$$
$$Ran_G K_R(X, d_X) = K_R(X, d_X) = (X, \{\{X\}\})$$

In general for any metric space $(X, d_X) \in \mathbf{D} - Ob(\mathbf{T})$ the functors $Lan_G K_L, Ran_G K_R$ respectively map (X, d_X) to the finest (most clusters) and coarsest (fewest clusters) partitions of X such that for any metric space $(X', d_{X'}) \in \mathbf{T}$ we have:

$$K(X', d_{X'}) = Lan_G K_L(X', d_{X'}) = Ran_G K_R(X', d_{X'})$$

and $Lan_G K_L, Ran_G K_R$ are functors. For example, suppose we have a metric space (X, d_X) where $X = \{x_1, x_2, x_3\}$. We can form the subcategories $\mathbf{T} \subseteq \mathbf{D} \subseteq \mathbf{Met}_{id}$ where:

$$Ob(\mathbf{T}) = \{(\{x_1, x_2\}, d_X), (\{x_1, x_3\}, d_X), (\{x_2, x_3\}, d_X)\}$$
$$Ob(\mathbf{D}) = Ob(\mathbf{T}) \cup (\{x_1, x_2, x_3\}, d_X)$$

$\mathbf{T}$ is a discrete category and the only non-identity morphisms in $\mathbf{D}$ are the inclusions $\{x_i, x_j\} \hookrightarrow \{x_1, x_2, x_3\}$. Now define $K : \mathbf{T} \to \mathbf{Part}_{id}$ to be the following functor:

$$K(\{x_1, x_2\}, d_X) = \{\{x_1, x_2\}\}$$
$$K(\{x_1, x_3\}, d_X) = \{\{x_1\}, \{x_3\}\}$$
$$K(\{x_2, x_3\}, d_X) = \{\{x_2\}, \{x_3\}\}$$

In this case we have that:

$$K_L(\{x_1, x_2, x_3\}, d_X) = \{\{x_1\}, \{x_2\}, \{x_3\}\} \qquad K_R(\{x_1, x_2, x_3\}, d_X) = \{\{x_1, x_2, x_3\}\}$$

Since the only points that need to be put together are x_1, x_2 and there are no non-identity morphisms out of $\{x_1, x_2, x_3\}$ in $\mathbf{D}$, we have:

$$Lan_G K_L(\{x_1, x_2, x_3\}, d_X) = \{\{x_1, x_2\}, \{x_3\}\}$$
$$Ran_G K_R(\{x_1, x_2, x_3\}, d_X) = \{\{x_1, x_2, x_3\}\}$$

As another example, suppose $\mathbf{D}$ is $\mathbf{Met}_{id}$ and $\mathbf{T}$ is the discrete subcategory of $\mathbf{D}$ whose objects are all metric spaces with no more than 2 elements. Define the following nonoverlapping flat $\mathbf{T}$-clustering functor:

$$K(\{x_1, x_2\}, d) = \begin{cases} \{\{x_1, x_2\}\} & d(x_1, x_2) \leq \delta \\ \{\{x_1\}, \{x_2\}\} & \text{else} \end{cases}$$

Now for some metric space $(X, d_X) \in \mathbf{D}$ with $|X| > 2$ and points $x_1, x_2 \in X$ we have that $Lan_G K_L$ maps x_1, x_2 to the same cluster if and only if there exists some chain of points $x_1, \cdots, x_2$ in X where for each pair of adjacent points x_1', x_2' in this chain and any metric space $(\{x_1', x_2'\}, d_{X'}) \in \mathbf{D}$ equipped with a non-expansive inclusion map:

$$\iota : (\{x_1', x_2'\}, d_{X'}) \hookrightarrow (X, d_X)$$

in $\mathbf{D}$, it must be that the points x_1', x_2' are in the same cluster in $K(\{x_1', x_2'\}, d_{X'})$. This is the case if and only if:

$$d_X(x_1', x_2') \leq \delta$$

Therefore, $Lan_G K_L$ maps x_1, x_2 to the same cluster if and only if x_1, x_2 are in the same connected component of the δ-Vietoris Rips complex of (X, d_X). $Lan_G K_L$ is therefore the δ-single linkage functor (Definition 3.8).

In contrast, since $|X| > 2$ there are no morphisms in $\mathbf{D}$ from (X, d_X) to any metric spaces in $\mathbf{T}$. Therefore:

$$Ran_G K_R(X, d_X) = (X, \{X\})$$

We can use this strategy to learn a clustering algorithm from real-world data. Recall that the Fashion MNIST dataset (Xiao et al., 2017) contains images of clothing and the categories that each image falls into. Suppose that we have two subsets of this dataset: a training set X_{tr} in which images are grouped by category and a testing set X_{te} of ungrouped images. We can use UMAP (McInnes et al., 2018) to construct metric spaces $(X_{tr}, d_{X_{tr}})$ and $(X_{te}, d_{X_{te}})$ from these sets.

Now suppose we would like to group the images in X_{te} as similarly as possible to the grouping of the images in X_{tr}.

For any category $\mathbf{D}^* \subseteq \mathbf{Met}$ with:

$$Ob(\mathbf{D}^*) = \{(X_{tr}, d_{X_{tr}}), (X_{te}, d_{X_{te}})\}$$

we can define subcategories $\mathbf{T} \subseteq \mathbf{D} \subseteq \mathbf{Met}_{id}$ and functor $K : \mathbf{T} \to \mathbf{Part}_{id}$ as follows:

1. Initialize $\mathbf{T}$ to an empty category and $\mathbf{D}$ to be the discrete category with a single object $\{(X_{te}, d_{X_{te}})\}$.

2. For every morphism

   $$f : (X_{tr}, d_{X_{tr}}) \to (X_{te}, d_{X_{te}})$$

   in $\mathbf{D}^*$ and pair $(x_1, x_2) \in X_{tr}$ of samples in the same clothing category, add the object $(\{f(x_1), f(x_2)\}, d_{X_{te}})$ to $\mathbf{T}$ and $\mathbf{D}$, add the inclusion morphism:

   $$\iota : (\{f(x_1), f(x_2)\}, d_{X_{te}}) \hookrightarrow (X_{te}, d_{X_{te}})$$

   to $\mathbf{D}$, and define $K(\{f(x_1), f(x_2)\}, d_{X_{te}})$ to map $f(x_1)$ and $f(x_2)$ to the same cluster.

3. For every morphism

   $$f : (X_{te}, d_{X_{te}}) \to (X_{tr}, d_{X_{tr}})$$

   in $\mathbf{D}^*$ define a metric space $(X'_{te}, d_{X'_{te}})$ where $X_{te} = X'_{te}$ and $d_{X_{te}} = d_{X'_{te}}$. Add the object $(X'_{te}, d_{X'_{te}})$ to $\mathbf{T}$ and $\mathbf{D}$, add the inclusion map

   $$\iota : (X_{te}, d_{X_{te}}) \hookrightarrow (X'_{te}, d_{X'_{te}})$$

   to $\mathbf{D}$ and define $K(X'_{te}, d_{X'_{te}})$ to be the partition of X'_{te} defined by the preimages of the function $(h \circ f)$ where h maps each element of X_{tr} to the category of clothing it belongs to.

We can now use $Lan_G K_L$ and $Ran_G K_R$ to partition X_{te}.

In Figure 17 we compare the clusterings produced by $Lan_G K_L, Ran_G K_R$ to the ground truth clothing categories. As a baseline we compute the δ-single linkage clustering algorithm (Definition 3.8) with δ chosen via line search to maximize the adjusted Rand score (Definition 2.76) with the ground truth labels.

As expected, we see that $Lan_G K_L$ produces a finer clustering (more clusters) than does $Ran_G K_R$ and that the clusterings produced by $Lan_G K_L$ and $Ran_G K_R$ are better than the clustering produced by single linkage in the sense of adjusted Rand score with ground truth.

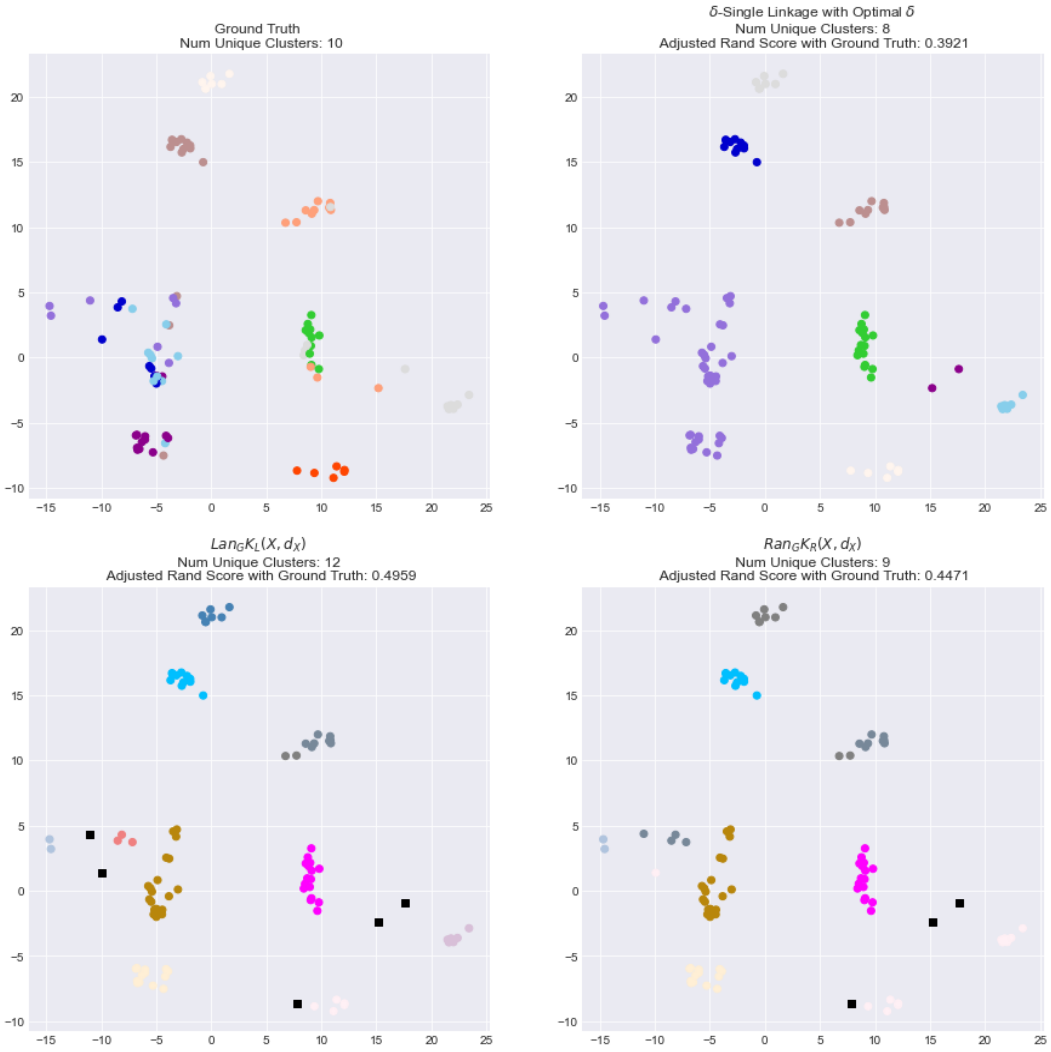


Figure 17: *Cluster assignments of a* 100 *point testing set* X_{te} *from the Fashion MNIST dataset (Xiao et al., 2017) shown in UMAP space (McInnes et al., 2018). Each color corresponds to a unique cluster, and points without clusters are shown as black squares. We show ground truth clothing categories, unsupervised* δ*-single linkage cluster assignments (*δ *chosen via line search), and the* $Lan_G K_L, Ran_G K_R$ *supervised cluster assignments. The* $Lan_G K_L, Ran_G K_R$ *algorithms are trained on a separate* 1000 *point random sample* X_{tr} *from the Fashion MNIST dataset.*

Frequency that Left Kan Clustering Beats Best δ-Single Linkage	Frequency that Right Kan Clustering Beats Best δ-Single Linkage
0.860 ($\pm$0.068)	0.680 ($\pm$0.091)

Table 7: *We compare the performance of the left and right Kan supervised clustering maps on the Fashion MNIST dataset to the performance of δ-single linkage clustering with an optimal choice of δ. We select* 100 *bootstrap training and testing samples of the Fashion MNIST dataset. We then train and evaluate each method on each such sample. To perform this evaluation we use the scikit-learn implementation of the Adjusted Rand Score (Pedregosa et al., 2011) to compare the algorithmically generated clusterings with the ground truth categorization. We then compute the frequency with which the left and right Kan supervised clustering maps perform better (have a higher Adjusted Rand Score with ground truth) than choosing the optimal value of δ for single linkage. The win rates and two standard error confidence bounds from the* 100 *experiments are shown. We see that the left and right Kan clustering maps both perform consistently better than single linkage. The code to run these experiments is on GitHub (Shiebler, 2021b).*

7.4 Meta-Supervised Learning

Suppose I is a set and O is a partial order. A supervised learning algorithm maps a labeled dataset (set of pairs of points in $I \times O$) to a function $f : I \to O$. For example, both $Lan_G K$ and $Ran_G K$ from Section 7.2 are supervised learning algorithms.

In this section we use Kan extensions to derive supervised learning algorithms from pairs of datasets and functions. Our construction combines elements of Section 7.2's point-level algorithms and Section 7.3's dataset-level functoriality constraints.

Suppose we have a finite partial order $S_f \subseteq (I \to O)$ of functions where for $f, f' \in S_f$ we have $f \leq f'$ when $\forall x \in I, f(x) \leq f'(x)$.

Proposition 7.10. *For any subset $S_f^* \subseteq S_f$ the upper antichain of S_f^* is the set:*

$$\{f \mid f \in S_f^*,\ \nexists f^* \in S_f^*, f < f^*\}$$

The upper antichain of S_f^* is an antichain in S_f^*, and for any function $f \in S_f^*$ there exists some function f^* in the upper antichain of S_f^* such that $f \leq f^*$.

Proof. Suppose f_1, f_2 are in the upper antichain of $S_f^* \subseteq S_f$ and $f_1 \leq f_2$. Then since

$$\nexists f_1^* \in S_f^*, f_1 < f_1^*$$

it must be that $f_1 = f_2$ and we can conclude that the upper antichain is an antichain.

Next, for any function $f \in S_f^*$ consider the set $\{f^* \in S_f^*, f < f^*\}$. Since S_f is finite, this set must have finite size. If this set is empty then f is in the upper antichain of S_f^*. If this set has size n then for any f^* in this set the set $\{f^{**} \in S_f^*, f^* < f^{**}\}$ must have size strictly smaller than n. We can therefore conclude by induction that the upper antichain of S_f^* contains at least one function f^* where $f \leq f^*$. □

Intuitively the upper antichain of S_f^* is the collection of all functions $f \in S_f^*$ that are not strictly upper bounded by any other function in S_f^*. The upper antichain of an empty set is of course itself an empty set.

Definition 7.11. *We can form the following categories:*

$\mathbf{D_C}$ *: The objects in* $\mathbf{D_C}$ *are* $\leq$*-antichains of functions* $X_f \subseteq S_f$*.* $\mathbf{D_C}$ *is a preorder in which* $X_f \leq X'_f$ *if for* $f \in X_f$ *there must exist some* $f' \in X'_f$ *where* $f \leq f'$*.*

$\mathbf{D_B}$ *: The objects in* $\mathbf{D_B}$ *are labeled datasets, or sets of pairs* $U = \{(x, y) \mid x \in I, y \in O\}$*.* $\mathbf{D_B}$ *is a preorder such that* $U \leq U'$ *when for all* $(x, y') \in U'$ *there exists* $(x, y) \in U$ *where* $y \leq y'$*.*

$\mathbf{D_A}$ *: A subcategory of* $\mathbf{D_B}$ *such that if* $U \leq U' \in \mathbf{D_B}$ *then* $U \leq U' \in \mathbf{D_A}$*.*

Proposition 7.12. $\mathbf{D_B}$ *and* $\mathbf{D_C}$ *are preorder categories.*

Proof.
$\mathbf{D_C}$
We trivially have $X_f \leq X_f$ in $\mathbf{D_C}$. To see that $\leq$ is transitive in $\mathbf{D_C}$ simply note that if $X_{f_1} \leq X_{f_2}$ and $X_{f_2} \leq X_{f_3}$ then for $f_1 \in X_{f_1}$ there must exist $f_2 \in X_{f_2}, f_1 \leq f_2$, which implies that there must exist $f_3 \in X_{f_3}, f_1 \leq f_2 \leq f_3$.

$\mathbf{D_B}$
We trivially have $U \leq U$ in $\mathbf{D_B}$. To see that $\leq$ is transitive in $\mathbf{D_B}$ simply note that if $U_1 \leq U_2$ and $U_2 \leq U_3$ in $\mathbf{D_B}$ then for $(x, y_3) \in U_3$ there must exist $(x, y_2) \in U_2, y_2 \leq y_3$ which implies that there must exist $(x, y_1) \in U_1, y_1 \leq y_2 \leq y_3$. □

Intuitively, $\mathbf{D_A}$ is a collection of labeled training datasets and $\mathbf{D_B}$ is a collection of labeled testing datasets. We can define a functor that maps each training dataset to all of the trained models that agree with that dataset.

Proposition 7.13. *The map* $K : \mathbf{D_A} \to \mathbf{D_C}$ *that acts as the identity on morphisms and maps the object* $U \in \mathbf{D_A}$ *to the upper antichain of the following set:*

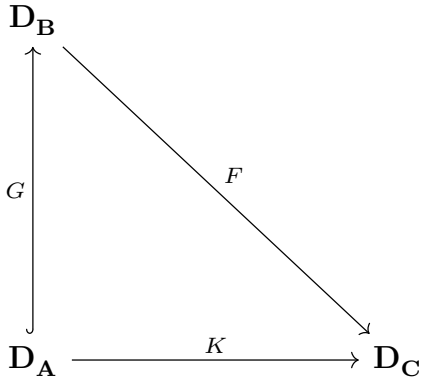
$$S_K(U) = \{f \mid f \in S_f, \forall (x, y) \in U, f(x) \leq y\}$$

is a functor.

Proof. To start, note that K maps objects in $\mathbf{D_A}$ to objects in $\mathbf{D_C}$ since the upper antichain of $S_K(U)$ must be an antichain in S_f by Proposition 7.10.

Next, we need to show that if $U \leq U'$ then $K(U) \leq K(U')$. For any $x, y' \in U'$ it must be that there exists $(x, y) \in U$ where $y \leq y'$, so if $f \in K(U)$ then by the definition of K we have $f(x) \leq y \leq y'$. Therefore $f \in S_K(U')$, so by Proposition 7.10 $K(U')$ contains f' where $f \leq f'$. Therefore $K(U) \leq K(U')$. □

Now define $G : \mathbf{D_A} \hookrightarrow \mathbf{D_B}$ to be the inclusion functor. A functor $F : \mathbf{D_B} \to \mathbf{D_C}$ such that $F \circ G$ commutes with K will then be a mapping from the testing datasets in $\mathbf{D_B}$ to collections of trained models.



We can take the left and right Kan extensions of K along the inclusion functor $G : \mathbf{D_A} \hookrightarrow \mathbf{D_B}$ to find the optimal such mapping.

Proposition 7.14. *The map $Lan_G K$ that acts as the identity on morphisms and maps the object $U \in \mathbf{D_B}$ to the upper antichain of the following set:*

$$S_L(U) = \bigcup_{\{U' \mid U' \in \mathbf{D_A}, U' \leq U\}} K(U')$$

is the left Kan extension of K along G.

Next, the map $Ran_G K$ that acts as the identity on morphisms and maps the object $U \in \mathbf{D_B}$ to the upper antichain of the following set:

$$S_R(U) = \{f \mid f \in S_f,\ \forall U' \in \{U' \mid U' \in \mathbf{D_A}, U \leq U'\},\ \exists f' \in K(U'),\ f \leq f'\}$$

is the right Kan extension of K along G.

Proof. We first need to show that $Lan_G K$ is a functor. Note that $Lan_G K$ maps objects in $\mathbf{D_B}$ to objects in $\mathbf{D_C}$ since the upper antichain of $S_L(U)$ must be an antichain in S_f.

Next, suppose $U_1 \leq U_2$ and that $f \in Lan_G K(U_1)$. Consider the set of all $U' \in \mathbf{D_A}$ where $U' \leq U_1$. Since $U_1 \leq U_2$, this is a subset of the set of all $U' \in \mathbf{D_A}$ where $U' \leq U_2$. Since $S_L(U_1)$ is defined to be a union of the elements in the set, we have that $S_L(U_1) \subseteq S_L(U_2)$. Since $f \in Lan_G K(U_1)$ implies that $f \in S_L(U_1)$, this implies that $f \in S_L(U_2)$ as well. Proposition 7.10 then implies that there must exist $f' \in Lan_G K(U_2)$ where $f \leq f'$ and therefore $Lan_G K(U_1) \leq Lan_G K(U_2)$.

Next, we will show that $Lan_G K$ is the left Kan extension of K along G.

- Consider some $U \in \mathbf{D_A}$ and $f \in K(U)$. Since $U \leq U$, we have by the definition of S_L that $f \in S_L(U)$. Proposition 7.10 then implies that $\exists f' \in Lan_G K(U)$ such that $f \leq f'$. This implies that $K \leq Lan_G K \circ G$.

- Now consider any functor $M_L : \mathbf{D_B} \to \mathbf{D_C}$ such that $K \leq (M_L \circ G)$. We must show that $Lan_G K \leq M_L$. For some $U \in \mathbf{D_B}$ suppose $f \in Lan_G K(U)$. By the definition of S_L there must exist some $U' \in \mathbf{D_A}$ where $U' \leq U$ such that $f \in K(U')$. Since $K(U') \leq M_L(U')$, there must exist some $f' \in M_L(U')$ where $f \leq f'$. Since M_L is a functor, we have $M_L(U') \leq M_L(U)$ which implies that there must exist some $f^* \in M_L(U)$ where $f \leq f' \leq f^*$. Therefore $Lan_G K \leq M_L$.

Next, we need to show that $Ran_G K$ is a functor. Note that $Ran_G K$ maps objects in $\mathbf{D_B}$ to objects in $\mathbf{D_C}$ since the upper antichain of $S_R(U)$ must be an antichain in S_f.

Next, suppose $U_1 \leq U_2$ and that $f \in Ran_G K(U_1)$. Consider the set of all $U' \in \mathbf{D_A}$ where $U_2 \leq U'$. Since $U_1 \leq U_2$, this is a subset of the set of all $U' \in \mathbf{D_A}$ where $U_1 \leq U'$. Therefore by the definition of S_R we have that $S_R(U_1) \subseteq S_R(U_2)$. Since $f \in Ran_G K(U_1)$ implies that $f \in S_R(U_1)$, this implies that $f \in S_R(U_2)$ as well. Proposition 7.10 then implies that there must exist $f' \in Ran_G K(U_2)$ where $f \leq f'$ and therefore $Ran_G K(U_1) \leq Ran_G K(U_2)$.

Next, we will show that $Ran_G K$ is the right Kan extension of K along G.

- For $U \in \mathbf{D_A}$ since $U \leq U$ when $f \in S_R(U)$ we have by the definition of S_R that $\exists f' \in K(U)$ such that $f \leq f'$. Since $Ran_G K(U)$ is a subset of $S_R(U)$, this implies that $Ran_G K \circ G \leq K$.

- Now consider any functor $M_R : \mathbf{D_B} \to \mathbf{D_C}$ such that $(M_R \circ G) \leq K$. We must show that $M_R \leq Ran_G K$. For some $U \in \mathbf{D_B}$ suppose $f \in M_R(U)$. Since M_R is a functor, it must be that for all $U' \in \mathbf{D_A}$ where $U \leq U'$ we have that $M_R(U) \leq M_R(U')$ and therefore $\exists f'_{M_R} \in M_R(U'), f \leq f'_{M_R}$. Since $(M_R \circ G) \leq K$, this implies that for all $U' \in \mathbf{D_A}$ where $U \leq U'$ we have that $\exists f'_K \in K(U'), f \leq f'_{M_R} \leq f'_K$. By the definition of S_R this implies that $f \in S_R(U)$. Proposition 7.10 therefore implies that there exists $f'_R \in Ran_G K(U)$ such that $f \leq f'_R$, and therefore $M_R(U) \leq Ran_G K(U)$.

□

Intuitively the functions in $Ran_G K(U)$ and $Lan_G K(U)$ are as large as possible subject to constraints imposed by the selection of sets in $Ob(\mathbf{D_A})$. The functions in $Lan_G K(U)$ are subject to a membership constraint and grow smaller when we remove objects from $Ob(\mathbf{D_A})$. The functions in $Ran_G K(U)$ are subject to an upper boundedness-constraint and grow larger when we remove objects from $Ob(\mathbf{D_A})$.

Consider the extreme case where $Ob(\mathbf{D_A}) = \emptyset$. For any $U \in \mathbf{D_B}$ we have that:

$$S_L(U) = \bigcup_{\{U' \mid U' \in \emptyset, \cdots\}} K(U') = \emptyset$$
$$S_R(U) = \{f \mid f \in S_f, \ \forall U' \in \emptyset, \ \cdots\} = S_f$$

so $Lan_G K(U)$ is empty and $Ran_G K(U)$ is the upper antichain of S_f.

Now consider the extreme case where $Ob(\mathbf{D_A}) = Ob(\mathbf{D_B})$. For any $U \in \mathbf{D_B}$ and $f \in K(U)$ the functoriality of K implies that:

$$\forall U' \in \{U' \mid U' \in \mathbf{D_A}, U \leq U'\}, \ \exists f' \in K(U'), \ f \leq f'$$

and therefore $f \in S_R(U)$. This implies $K(U) \leq Ran_G K(U)$. Similarly, for any $f \in Lan_G K(U)$ it must be that:

$$\exists U' \in \mathbf{D_A}, \ U' \leq U, \ f \in K(U')$$

which by the functoriality of K implies that:

$$\exists f^* \in K(U), f \leq f^*$$

and therefore $Lan_G K(U) \leq K(U)$. Therefore in this extreme case we have:

$$Ran_G K(U) = Lan_G K(U) = K(U)$$

Let's now consider a more concrete example. Suppose $I = \mathbb{R}^2_{\geq 0}, O = \{\text{false}, \text{true}\}$, and S_f is the finite set of linear classifiers $l : \mathbb{R}^2_{\geq 0} \to \{\text{false}, \text{true}\}$ that can be expressed as:

$$l_{a,b}(x_1, x_2) = \begin{cases} \text{true} & x_2 \leq a * x_1 + b \\ \text{false} & \text{else} \end{cases}$$

where a, b are integers in $(-100, 100)$. Intuitively:

- The classifiers in $Lan_G K(U)$ are selected to be the classifiers that predict true as often as possible among the set of all classifiers that have no false positives on some $U' \in \mathbf{D_A}$ where $U' \leq U$.

- The classifiers in $Ran_G K(U)$ are constructed to predict true as often as possible subject to a constraint imposed by the selection of sets in $\mathbf{D_A}$. For every set $U' \in \mathbf{D_A}$ where $U \leq U'$ it must be that each classifier in $Ran_G K(U)$ is upper bounded at each point in I by some classifier in S_f with no false positives on U'.

A concrete example will demonstrate this. Suppose that $\mathbf{D_A}$ is:

$$\begin{array}{ccc} & & \{((2,2),\text{true}),((1,3),\text{true}),((4,4),\text{false})\} \\ & & \uparrow \leq \\ \{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{true})\} & \xleftarrow[\leq]{} & \{((2,2),\text{false}),((1,3),\text{false}),((4,4),\text{false})\} \end{array}$$

and that $\mathbf{D_B}$ is:

$$\begin{array}{ccc} & & \{((2,2),\text{true}),((1,3),\text{true}),((4,4),\text{false})\} \\ & & \uparrow \leq \\ \{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{true})\} & \xleftarrow[\leq]{} & \{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{false})\} \\ & & \uparrow \leq \\ & & \{((2,2),\text{false}),((1,3),\text{false}),((4,4),\text{false})\} \end{array}$$

We can see the following:

- $l_{(1,1)} \in K(\{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{true})\})$ since:

$$l_{(1,1)}(1,3) = \left(\begin{cases} \text{true} & 3 \leq 1*1+1 \\ \text{false} & \text{else} \end{cases} \right) = \text{false}$$

  but we have that:

$$l_{(1,2)}(1,3) = \left(\begin{cases} \text{true} & 3 \leq 1*1+2 \\ \text{false} & \text{else} \end{cases} \right) = \text{true}$$

$$l_{(2,1)}(1,3) = \left(\begin{cases} \text{true} & 3 \leq 2*1+1 \\ \text{false} & \text{else} \end{cases} \right) = \text{true}$$

- $l_{(0,2)} \in K(\{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{true})\})$ since:

$$l_{(0,2)}(1,3) = \left(\begin{cases} \text{true} & 3 \leq 0*1+2 \\ \text{false} & \text{else} \end{cases} \right) = \text{false}$$

  but we have that:

$$l_{(0,3)}(1,3) = \left(\begin{cases} \text{true} & 3 \leq 0*1+3 \\ \text{false} & \text{else} \end{cases} \right) = \text{true}$$

$$l_{(1,2)}(1,3) = \left(\begin{cases} \text{true} & 3 \leq 1*1+2 \\ \text{false} & \text{else} \end{cases} \right) = \text{true}$$

- $l_{(0,3)} \in K(\{((2,2),\text{true}),((1,3),\text{true}),((4,4),\text{false})\})$ since:

$$l_{(0,3)}(4,4) = \left(\begin{cases}\text{true} & 4 \leq 0 * 4 + 3 \\ \text{false} & \text{else}\end{cases}\right) = \text{false}$$

but we have that:

$$l_{(1,3)}(4,4) = \left(\begin{cases}\text{true} & 4 \leq 1 * 4 + 3 \\ \text{false} & \text{else}\end{cases}\right) = \text{true}$$
$$l_{(0,4)}(4,4) = \left(\begin{cases}\text{true} & 4 \leq 0 * 4 + 4 \\ \text{false} & \text{else}\end{cases}\right) = \text{true}$$

- $l_{(0,1)} \in K(\{((2,2),\text{false}),((1,3),\text{false}),((4,4),\text{false})\})$ since:

$$l_{(0,1)}(2,2) = \left(\begin{cases}\text{true} & 2 \leq 0 * 2 + 1 \\ \text{false} & \text{else}\end{cases}\right) = \text{false}$$
$$l_{(0,1)}(1,3) = \left(\begin{cases}\text{true} & 3 \leq 0 * 1 + 1 \\ \text{false} & \text{else}\end{cases}\right) = \text{false}$$
$$l_{(0,1)}(4,4) = \left(\begin{cases}\text{true} & 4 \leq 0 * 4 + 1 \\ \text{false} & \text{else}\end{cases}\right) = \text{false}$$

but we have that:

$$l_{(1,1)}(4,4) = \left(\begin{cases}\text{true} & 4 \leq 1 * 4 + 1 \\ \text{false} & \text{else}\end{cases}\right) = \text{true}$$
$$l_{(0,2)}(2,2) = \left(\begin{cases}\text{true} & 2 \leq 0 * 2 + 2 \\ \text{false} & \text{else}\end{cases}\right) = \text{true}$$

By the definition of $Lan_G K$ we have that:

$$Lan_G K(\{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{false})\})$$

must contain $l_{(0,1)}$ since we have that:

$$l_{(0,1)} \in K(\{((2,2),\text{false}),((1,3),\text{false}),((4,4),\text{false})\})$$

but:

$$l_{(0,2)} \notin K(\{((2,2),\text{false}),((1,3),\text{false}),((4,4),\text{false})\})$$
$$l_{(1,1)} \notin K(\{((2,2),\text{false}),((1,3),\text{false}),((4,4),\text{false})\})$$

Similarly, by the definition of $Ran_G K$ we have that:

$$Ran_G K(\{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{false})\})$$

must contain $l_{(0,2)}$ since we have that:

$$l_{(0,2)} \leq l_{(0,3)} \qquad l_{(0,2)} \leq l_{(1,2)}$$

but that there is no $l_{(a,b)}$ such that $l_{(0,2)} < l_{(a,b)}$ that is in both:

$$K(\{((2,2),\text{true}),((1,3),\text{true}),((4,4),\text{false})\})$$

and:

$$K(\{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{true})\})$$

since:

$$l_{(1,2)} \notin K(\{((2,2),\text{true}),((1,3),\text{true}),((4,4),\text{false})\})$$
$$l_{(0,3)} \notin K(\{((2,2),\text{true}),((1,3),\text{false}),((4,4),\text{true})\})$$

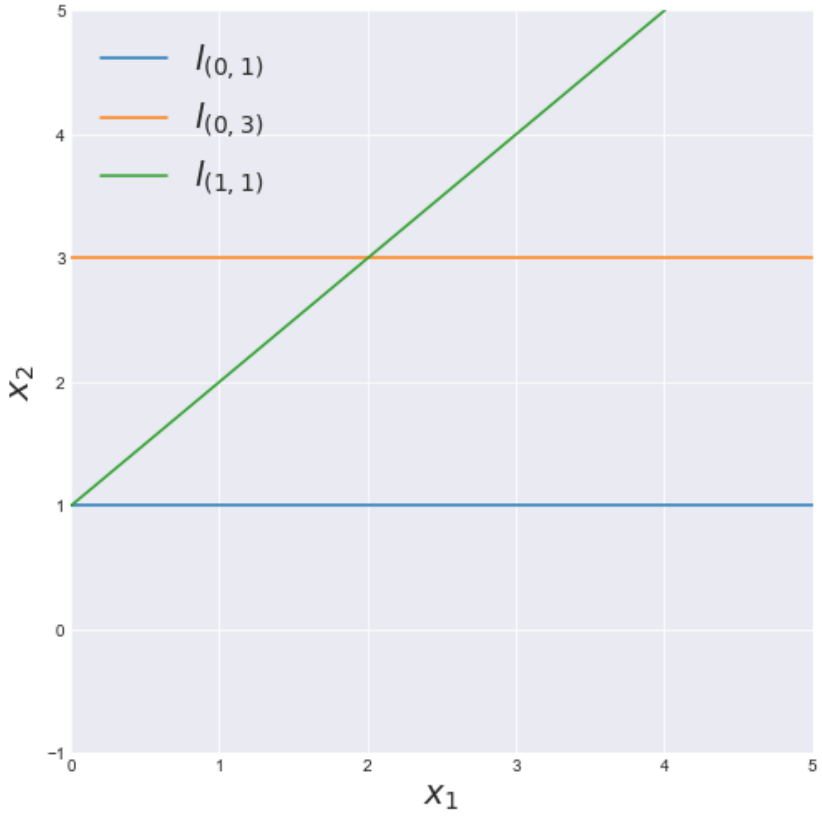


Figure 18: *The decision boundaries defined by $l_{(1,1)}$, $l_{(0,3)}$, and $l_{(0,1)}$.*

7.5 Function Approximation

In this section we will explore how we can use Kan extensions to approximate a function in one class with a function in another class. Suppose I is a set, O is a partial order, and S is a finite subset of I. We can define the following preorder:

Definition 7.15. *Define the preorder $\leq_S$ on $(I \to O)$ such that $f_1 \leq_S f_2$ if and only if $\forall x \in S, f_1(x) \leq f_2(x)$. If $f_1 \leq_S f_2, f_2 \not\leq_S f_1$ then write $f_1 <_S f_2$ and if $f_1 \leq_S f_2 \leq_S f_1$ then write $f_1 =_S f_2$.*

We can now define the following categories:

Definition 7.16. *Suppose S_f^1, S_f^2 are subsets of $(I \to O)$ such that $S_f^1 \subseteq S_f^2$ and no $=_S$-equivalence class of S_f^1 contains more than one function. We can construct the categories $\mathbf{F_A}, \mathbf{F_B}, \mathbf{F_C}$ as follows.*

- *The set of objects in the discrete category $\mathbf{F_A}$ is S_f^1.*
- *The set of objects in $\mathbf{F_B}$ is S_f^2. $\mathbf{F_B}$ is a preorder with morphisms $\leq_S$.*
- *$\mathbf{F_C}$ is the subcategory of $\mathbf{F_B}$ in which objects are functions in S_f^1 and morphisms are $\leq_S$.*

A functor $\mathbf{F_B} \to \mathbf{F_C}$ acts as a choice of a function in S_f^1 for each function in S_f^2. For example, if S_f^1 contains only linear functions and S_f^2 is the class of all polynomials then we can view a functor $\mathbf{F_B} \to \mathbf{F_C}$ as selecting a linear approximation for each polynomial in S_f^2.

Proposition 7.17. *For some function $g \in S_f^2$ define its minimal S-overapproximation to be the function $h \in S_f^1$ where $g \leq_S h$ and $\forall h' \in S_f^1$ where $g \leq_S h'$ we have $h \leq_S h'$. If this function exists it is unique.*

Proof. Suppose h_1, h_2 are both minimal S-overapproximations of g. Then $h_1 \leq_S h_2$ and $h_2 \leq_S h_1$ which by the definition of S_f^1 implies that $h_1 = h_2$. □

Proposition 7.18. *For some function $g \in S_f^2$ define its maximal S-underapproximation to be the function $h \in S_f^1$ where $h \leq_S g$ and $\forall h' \in S_f^1$ where $h' \leq_S g$ we have $h' \leq_S h$. If this function exists it is unique.*

Proof. Suppose h_1, h_2 are both maximal S-underapproximations of g. Then $h_2 \leq_S h_1$ and $h_1 \leq_S h_2$ which by the definition of S_f^1 implies that $h_1 = h_2$. □

Proposition 7.19. *Suppose that for some $g \in S_f^2$ there exists some $h \in S_f^1$ such that $h =_S g$. Then h will be both the minimal S-overapproximation and the maximal S-underapproximation of g.*

Proof. To start, note that h must satisfy $g \leq_S h$ and for any $h' \in S_f^1$ where $g \leq_S h'$ we have:

$$h =_S g \leq_S h'$$

so h is the minimal S-overapproximation of g.

Next, note that h must satisfy $h \leq_S g$ and for any $h' \in S_f^1$ where $h' \leq_S g$ we have:

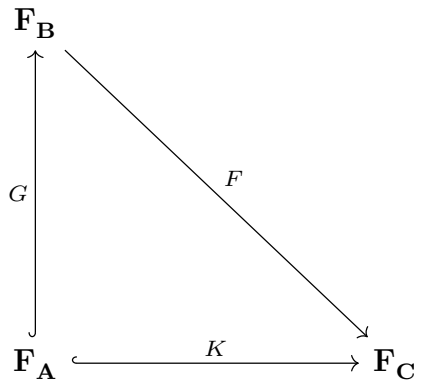
$$h' \leq_S g =_S h$$

so h is also the maximal S-underapproximation of g. □

We can now show the following:

Proposition 7.20. *Define both $K : \mathbf{F_A} \hookrightarrow \mathbf{F_C}$ and $G : \mathbf{F_A} \hookrightarrow \mathbf{F_B}$ to be inclusion functors. Then:*

- *Suppose that for any function $g \in S_f^2$ there exists a maximal S-underapproximation h of g. Then the left Kan extension of K along G is the functor $Lan_G K$ that acts as the identity on morphisms and maps g to h.*
- *Suppose that for any function $g \in S_f^2$ there exists a minimal S-overapproximation h of g. Then the right Kan extension of K along G is the functor $Ran_G K$ that acts as the identity on morphisms and maps g to h.*



Proof. We first show that $Lan_G K$ is a functor when it exists. Since $\mathbf{F_B}, \mathbf{F_C}$ are preorders, we simply need to show that when $f_1 \leq_S f_2$ then $Lan_G K(f_1) \leq_S Lan_G K(f_2)$. Since $Lan_G K(f_1) \leq_S f_1$, we have that $Lan_G K(f_1) \leq_S f_2$. Then $Lan_G K(f_1) \leq_S Lan_G K(f_2)$ by the definition of the maximal S-underapproximation of f_2.

We next show that $Ran_G K$ is a functor when it exists. Since $\mathbf{F_B}, \mathbf{F_C}$ are preorders, we simply need to show that when $f_1 \leq_S f_2$ then $Ran_G K(f_1) \leq_S Ran_G K(f_2)$. Since $f_2 \leq_S Ran_G K(f_2)$, we have that $f_1 \leq_S Ran_G K(f_2)$. Then $Ran_G K(f_1) \leq_S Ran_G K(f_2)$ by the definition of the minimal S-overapproximation of f_1.

Next, we will show that $Lan_G K$ and $Ran_G K$ are respectively the left and right Kan extensions when they exist. First, by Proposition 7.19 if $f \in S_f^1$ then f must be both the minimal S-overapproximation and maximal S-underapproximation of f. Therefore we have:

$$K(f) = Lan_G K(f) = Ran_G K(f)$$

Next, consider any functor $M_L : \mathbf{F_B} \to \mathbf{F_C}$ such that $\forall f' \in S_f^1, f' \leq_S M_L(f')$. Suppose $f \in S_f^2$. Since M_L is a functor, it must be that $\forall f' \in S_f^1$ where $f' \leq_S f$ we have $M_L(f') \leq_S M_L(f)$ and therefore $f' \leq_S M_L(f)$. Since $Lan_G K(f)$ is the maximal S-underapproximation of f, it must be that $Lan_G K(f) \in S_f^1$ and $Lan_G K(f) \leq_S f$ and therefore $Lan_G K(f) \leq_S M_L(f)$

Next, consider any functor $M_R : \mathbf{F_B} \to \mathbf{F_C}$ such that $\forall f' \in S_f^1, M_R(f') \leq_S f'$. Suppose $f \in S_f^2$. Since M_R is a functor, it must be that $\forall f' \in S_f^1$ where $f \leq_S f'$ we have $M_R(f) \leq_S M_R(f')$ and therefore $M_R(f) \leq_S f'$. Since $Ran_G K(f)$ is the minimal S-overapproximation of f, it must be that $Ran_G K(f) \in S_f^1$ and $f \leq_S Ran_G K(f)$ and therefore $M_R(f) \leq_S Ran_G K(f)$ □

Intuitively, the Kan extensions of the inclusion functor $K : \mathbf{F_A} \to \mathbf{F_C}$ along the inclusion functor $G : \mathbf{F_A} \to \mathbf{F_B}$ map a function $g \in S_f^2$ to its best S_f^1-approximations over the points in S.

For example, suppose $I = O = \mathbb{R}$, g is a polynomial, S_f^1 is a set of lines defined by all pairs of distinct points in S and $S_f^2 = S_f^1 \cup g$. $Lan_G K$ and $Ran_G K$ may or may not

exist depending on the choice of S and g. In Figure 19 we give an example S, g in which $Lan_G K$ exists and $Ran_G K$ does not (left) and an example S, g in which $Ran_G K$ exists and $Lan_G K$ does not (right).

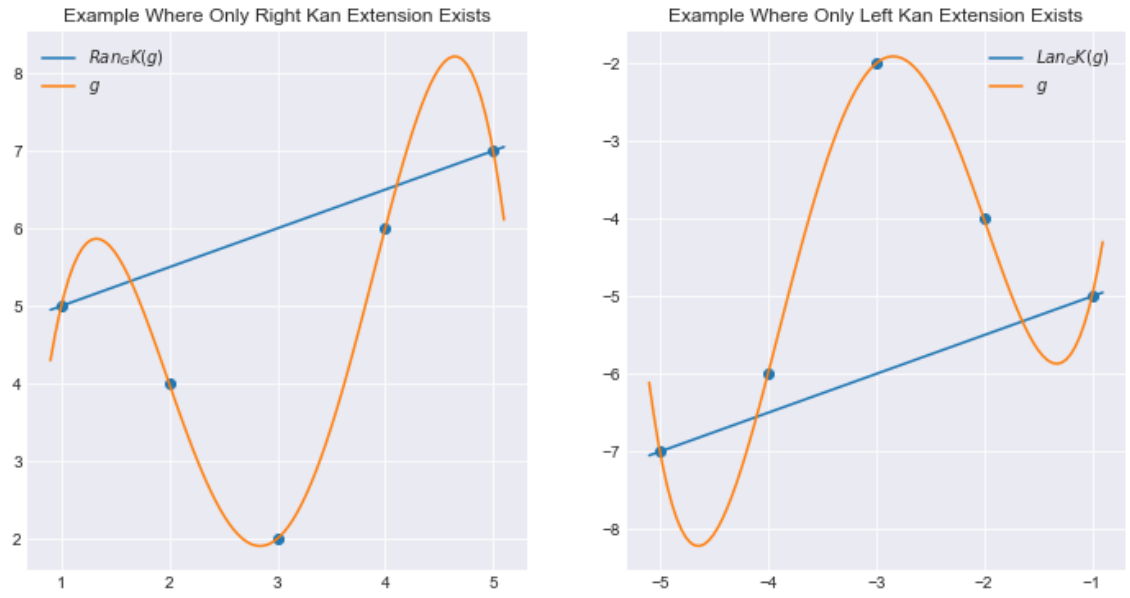


Figure 19: *Left and right Kan extensions of* $K : \mathbf{F_A} \hookrightarrow \mathbf{F_C}$ *along* $G : \mathbf{F_A} \hookrightarrow \mathbf{F_B}$ *for two example sets* S *and polynomials* g *where* S_f^1 *is a set of lines defined by pairs of distinct points in* S *and* $S_f^2 = S_f^1 \cup g$.

As another example, suppose $I = O = \mathbb{R}$, S_f^1 is a subset of all polynomials of degree $|S|-1$ and S_f^2 is a subset of all functions $\mathbb{R} \to \mathbb{R}$. Since there always exists a unique $n-1$ degree polynomial through n unique points, for any S there exists some S_f^1 so that both $Lan_G K$ and $Ran_G K$ exist and map $g \in S_f^2$ to the unique $|S|-1$ degree polynomial that passes through the points $\{(x, g(x)) \mid x \in S\}$.

7.6 Closing Thoughts on Category Theoretic Supervised Learning

The category theoretic perspective on supervised learning that we introduce in this Chapter is fundamentally different from the traditional data science perspective. Intuitively, the traditional data science perspective is mean and percentile-focused whereas the category theoretic perspective is min and max-focused. That is, traditional data science algorithms may have objectives like "minimize total errors" while the category theoretic algorithms we discuss in this Chapter have objectives like "minimize false positives subject to no false negatives on some set." As a result, algorithms built from the category theoretic perspective may behave more predictably, but can also be more sensitive to noise.

8 Conclusions

In this thesis we show that studying the underlying compositional and functorial structure in machine learning systems allows us to better understand them.

We will now revisit the three questions we pose in Section 1.1.

Question 1: Can we derive new, assumption light representations of the process of fitting a model on data?

We address this question in Chapters 4 and 5 by exploring how we can leverage simple compositional structures to represent the process of fitting a model on data. First, we demonstrate that only a light set of assumptions is required to derive iterative algorithms for optimizing a model on data. These assumptions hold over very general settings, such as polynomials over ordered rings. Traditional presentations of these algorithms assume much more structure, such as continuous differentiability; our presentation has broader applicability.

Next, we introduce a simple compositional pattern that underlies both probabilistic modeling and gradient based optimization. We use this pattern to formalize the role of maximum likelihood estimation as a bridge from statistics to model fitting.

Question 2: How does the structure of a model trained on a dataset mirror the structure of that dataset?

We address this question in Chapter 6 by exploring how unsupervised algorithms produce models that mirror the structure of the datasets they are trained on. This lets us characterize and predict the behaviors of these algorithms.

We demonstrate that the clusters or embeddings that many algorithms construct from data change in a predictable way as we transform that data. We use this insight to predict the kinds of dataset transformations that different kinds of models are resilient to.

We also build on this perspective to derive new results on the behaviors and limitations of popular models and algorithms. This enables us to derive novel algorithms like single linkage scaling and MDS-FuzzySimplex (Section 6.3.7).

Question 3: Can we identify common structures that underlie seemingly different machine learning systems?

We address this question in Chapters 6 and 7 by expressing common machine learning algorithms in a unified category theoretic language. First, we demonstrate that both clustering algorithms and manifold learning algorithms can be expressed as functors. This perspective lends us a great deal of generality. In addition to unifying clustering and manifold learning algorithms along hierarchies according to functorial behavior, we can use this perspective to construct manifold learning algorithms from clustering algorithms.

Next, we show how a wide variety of machine learning problems can be expressed as Kan extensions. Each of classification, supervised clustering, meta-learning, and function approximation fit into this framework. This perspective enables us to derive novel algorithms across a number of domains, such as the left/right Kan classifiers (Section 7.2) and the left/right Kan supervised clustering maps (Section 7.3).

8.1 Why Applied Category Theory? (Revisited)

In Section 1.3 we propose addressing the increasing complexity of machine learning systems by studying them in their simplest forms. This enables us to derive insights into these systems' fundamental structures. We further assert that the right language to explore this structure is applied category theory: in particular, compositionality and functoriality. We will now revisit these assertions.

8.1.1 Why Compositionality? (Revisited)

Although machine learning is powerful, it is also difficult to predict its behavior. This makes reasoning about the combination of machine learning systems extremely challenging. As such combinations have grown in scale and commonality, it has become increasingly important to develop techniques to understand them.

In this thesis we demonstrate multiple strategies to predict and understand the behavior of combinations of machine learning systems. By building on these strategies we derive novel insights into how to improve these machine learning systems. We also demonstrate the effectiveness of these insights on data.

In Chapter 4 we demonstrate how to use a minimal set of assumptions on compositional behavior to derive optimization algorithms. This enables us to show that some of the desirable properties of commonly used algorithms stem from compositional behaviors that exist even in very simple systems.

In Chapter 5 we show that likelihood functions compose along the same patterns as neural networks. This enables us to show how uncertainties propagate through compositional models.

In Chapter 6 we show that manifold learning algorithms decompose into a clustering component and a loss function component. This perspective makes it much easier to organize algorithms into groups or predict algorithm behavior. This perspective also enables us to recombine components to generate new algorithms.

Finally, in Chapter 7 we decompose supervised learning algorithms into primitives. This enables us to use the same primitives to express the labels on the training set, the inclusion of the training set into the testing set, and the learned model.

8.1.2 Why Functoriality? (Revisited)

Transformation invariance lurks behind most of the properties of a machine learning system. How will my model respond to changes of my data or training procedure? What are the kinds of transformations under which my model is invariant or equivariant? What kind of data or model changes will damage the well behavedness conditions of my system? What behavioral constraints must my model adhere to?

We can develop a deeper understanding of the relationships between the components of machine learning systems by studying these transformation invariances. In this thesis we demonstrate that the best tool for this task is functoriality.

In Chapter 4 we use functoriality conditions to express the advantages of Newton's method over gradient descent. The functorial perspective enables us to extend this result to novel applications, such as polynomials over ordered rings.

In Chapter 5 we explore how likelihood functions compose along with statistical models. By using the language of functoriality we can easily express this in terms of the

relationship between the compositions of parametric functions, probability distributions, and loss function gradients.

In Chapter 6 we use functoriality to predict the behavior of unsupervised learning algorithms on different kinds of data. The functorial perspective enables us to express the behavior of flat clustering, hierarchical clustering, and manifold learning with the same language.

Finally, in Chapter 7 we show that functorial representations enable us to derive new algorithms from optimality conditions, such as the left/right Kan classifiers and the left/right Kan supervised clustering maps. The language of functoriality enables us to express optimization constraints across many different supervised learning problems as a common structure.

8.2 Challenges

As we use category theory to derive insights into machine learning, there are several pain points that we repeatedly experience.

The first is the composition of approximations. It is quite common in machine learning to work with components that only approximately achieve the task they are optimized for. For example, most supervised learning algorithms do not perfectly fit the datasets we train them on. To make things even more complicated, this pattern is not a defect, but instead a core attribute of machine learning systems. Models often overfit and score worse on out of sample data if we force them to have perfect performance on their training sets.

However, this makes it challenging to reason about what happens when we combine components together. Even if we know that the model f has an error of ϵ and the model f' has an error of ϵ', it may be difficult to bound the error of model $f' \circ f$. This error may be larger than the sum or product of ϵ or ϵ', which makes it difficult to construct a category on top of this kind of composition.

We encounter this problem when we reason about the convergence of generalized optimization in Section 4.1. There is no simple way to extend this construction to capture the relationship between optimization convergence and model performance.

We also encounter this problem when we explore the relationship between the composition of likelihood functions and parametric models in Chapter 5. There is no simple way to incorporate the final solution to the maximum likelihood optimization problem into this compositional framework.

We also face this challenge in Section 6.3. Although it seems simplest to reason about manifold learning algorithms as the composition of an overlapping hierarchical clustering algorithm with a map from clusters to embeddings, the approximate nature of manifold learning optimization makes it challenging to represent this kind of map as a functor.

We can overcome this constraint by rotating our perspective and focusing on the compositions that exist around the approximation step. In Section 4.1 and Chapter 5 we choose to abstract the behavior of the trained model and focus directly on the structure of the optimization problem and the optimization process. In Section 6.3 our decision to define manifold learning algorithms in terms of the problems they solve rather than the structures they derive enables us to avoid this problem entirely.

The second challenge is the tight coupling between components. It is quite common in machine learning to apply one kind of model to transform data (such as clustering or manifold learning) before applying another model to generate predictions. For example,

in a deep neural network we can view the first layers as the data transformation and the last layer as the prediction model. However, it is often not possible to define optimization criteria for the data transformation layers that are independent of the choice of prediction layer.

We encounter this problem in Chapter 5 when we relate probability distributions to model update steps. In general we need to represent the steps from data generation to model gradient as a distinct unit. We therefore need to apply a strong set of assumptions to overcome this challenge and recombine system components with category theoretic machinery.

The third challenge is the foundational tension between optimal guarantees and resilience to noise. It is tempting to think about how we could use universal constructions to solve the optimization problems that appear in machine learning. However, the solutions that we can derive with universal constructions tend to look like "minimize false negatives given no false positives" rather than "minimize expected loss." Said another way, if the traditional machine learning solution to a problem will involve a mean or average, then the category theoretic solution to the same problem will be based on a minimum or a maximum. As a result, the category theoretic solutions to optimization problems tend to be less resilient to noisy data: a classifier that must achieve no false positives will generate useless results if a single positive sample is mislabeled in its training set.

We encounter this problem in Sections 6.1 and 6.2. Certain clustering algorithms like K-means rely heavily on averages. These algorithms are difficult to express as functors. We overcome this challenge by focusing on agglomerative clustering algorithms like robust single linkage that rely on minimum distances but use dataset transformations to achieve robustness. Although many manifold learning algorithms rely on averages (such as metric multidimensional scaling) in Section 6.3 we manage to avoid this challenge by using the optimization objective as the primitive object of study.

We also encounter this problem in Section 7.2 when we define classification algorithms from Kan extensions. In that case we handle the problem by transforming our data to fit this constraint.

8.3 Future Work

In this section we will cover some potential future directions for the intersection of category theory and machine learning.

8.3.1 Extending This Work

In Chapter 4 we explore how several popular optimization algorithms behave in a generalized setting. However, there is still much to do. For example, although we identify the properties that a generalized optimization flow must possess in order to converge, our construction does not distinguish between flows that converge to minima or to arbitrary points. Furthermore, there are many variations of gradient based optimizers that our formulation does not capture, such as stochastic optimizers like stochastic gradient descent and batch gradient descent.

Another potential future direction for this work is to explore generalizations of constrained optimization. Optimization algorithms like gradient descent and Newton's method can be adapted to solve constrained optimization problems, and we may be able to do the same for their generalized analogs. This may enable us to adapt the technique for

minimizing integer polynomials that we introduce in Section 4.1.4 to generate approximate solutions to integer programs. Solving integer programs is an NP-hard problem with an enormous number of practical applications.

In Chapter 5 we introduce a frequentist construction for composing statistical models with multiple sources of uncertainty. One potential future direction is to use this construction to model physical or biological systems. Another is to extend the category **DF** of deterministic and frequentist models to handle generative algorithms that model uncertainty in the input vector and Bayesian algorithms that model uncertainty in the parameter vector.

Another potential future direction would be to relax the restrictions on our definition of a Marginal Likelihood Factorization Category (Definition 5.20). The current definition specifies that each category is characterized by a single marginal error function er. This makes it challenging to reason about how statistical models of different types might compose.

In Chapter 6 we explored how a functorial perspective enables us to derive novel insights into unsupervised algorithms. One potential future direction would be to expand the multiparameter flattening procedure (Section 6.2) to support larger scale applications. The current algorithm reduces to a binary integer program, which can be very slow to solve.

Yet another potential future direction would be to derive more powerful theorems around the resilience of other kinds of unsupervised or supervised algorithms to noise. Our bounds in Section 6.3.6 are very general, but also very loose. We may also be able to tighten them by switching our perspective from finite metric spaces to distributions or even involving categorical probability to replace interleaving distance with a probabilistic analog.

In Chapter 7 we demonstrate that Kan extensions can be used to derive many different kinds of supervised learning algorithms. However, the category theoretic formulations of these algorithms are inherently focused on minimums and maximums rather than averages. Averages are required to build algorithms that are robust to noise, and a potential future direction for this work is to extend these algorithms to incorporate averages. For example, we may be able to combine multiple Kan classifiers together to generate a robust Kan classifier ensemble. It may even be possible to apply a boosting approach in which we minimize the ordering loss, fit Kan classifiers, and then repeat on the samples in the disagreement region.

8.3.2 Mainstream Adoption

Although there has been an enormous amount of progress on applied category theoretic approaches to data science and machine learning, still the vast majority of data science and machine learning innovations are completely unrelated to category theory. Furthermore, the machine learning community is largely unaware of these category theoretic advances or their implications. Most likely a primary reason for this gap is the substantial upfront cost and unclear payoff of understanding category theoretic terminology and technology. However, there are a few areas of machine learning where category theory has begun to make inroads.

The first is topological data analysis, or the application of topological techniques to data science. Topological data analysts study the properties of the topological spaces constructed from the data that they work with. A primary strategy in topological data

analysis is to build a simplicial complex or a family of simplicial complexes from a dataset and then to study the homological properties of these complexes. Since modern topology is closely tied to category theory, many topological data analysis applications are most easily unified and extended with category theory.

The next is geometric deep learning, or the generalization of equivariant and invariant neural network operators to non-Euclidean domains like graphs and manifolds. Geometric deep learning focuses on the application of structure preserving transformations to data. These transformations lend themselves naturally to a category theoretic perspective. As a result commutative diagrams, functoriality, and naturality have begun to work their way into the vernacular of researchers exploring geometric deep learning techniques.

The most important next step in this field is the derivation of new applications of machine learning that benefit from this perspective. All of the areas that we advance in this dissertation are now primed for such an innovation. For example, we could build on Chapter 4 and drive towards a mainstream accessible system for deriving new optimization algorithms from constraints. Furthermore, we may be able to adapt our work in Chapter 5 to develop new loss functions and inference techniques for deep probabilistic models. Also, it is possible to modify our language in Chapter 6 to support understanding by the mainstream machine learning community. In all unsupervised learning applications we aim to develop a model whose structure closely matches the structure of the data it was trained on. It is easy to map this perspective onto the language of functors and natural transformations, and our results in Chapter 6 suggest that it is quite easy to use this perspective to derive new algorithms that perform better than their commonly used counterparts. Finally, it may be possible to adapt the results in Chapter 7 to derive supervised clustering algorithms that perform near the state of the art for mainstream data science use cases.

8.3.3 Looking Forward

Looking towards future applications of category theory to machine learning, there are two areas in particular where we expect to see large strides. First, although we have begun to explore categorical extensions of classical machine learning techniques, there has been very little exploration of the generalization properties of these categorical algorithms. There is a need for a categorical perspective on learning theory to evolve along with the categorical perspective on machine learning. Second, our categorical perspectives on machine learning are not quite as unified as they cound be. For example, our synthetic perspectives on probability and gradient based learning are largely disjoint from each other and from our functorial perspective on clustering. Going forward we expect to see more innovations in unification.

We believe that these lines of research will accelerate machine learning progress by helping researchers better understand the foundational components of the algorithms that they use. This generalized perspective will also help us better understand the domains over which different algorithms will be successful.

References

Ordered ring - planetmath.org, 2022. URL https://planetmath.org/orderedring.

Martín. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. URL https://www.tensorflow.org/.

Hervé Abdi. Metric multidimensional scaling (MDS): analyzing distance matrices. *Encyclopedia of Measurement and Statistics. SAGE Publications*, 2007.

Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4), 2010.

Lloyd Allison. Types and classes of machine learning and data mining. In *Proceedings of the 26th Australasian computer science conference-Volume 16*. Australian Computer Society, Inc., 2003.

Andersen Ang. Convergence of gradient flow. *Course notes at University of Mons*, 2020. URL https://angms.science/doc/CVX/GradientFlowConvAna.pdf.

Robert B. Ash and Melvin F. Gardner. *Topics in Stochastic Processes*. Probability and Mathematical Statistics: a series of monographs and textbooks. Academic Press, 1975.

Stephen Bailey. Principal component analysis with noisy and/or missing data. *Publications of the Astronomical Society of the Pacific*, 124(919):1015–1023, 2012. ISSN 00046280, 15383873. URL http://www.jstor.org/stable/10.1086/668105.

Mukund Balasubramanian. The isomap algorithm and topological stability. *Science*, 295 (5552), 2002.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6), 2003.

Patrick Billingsley. *Probability and Measure*. John Wiley and Sons, second edition, 1986. URL https://www.colorado.edu/amath/sites/default/files/attached-files/billingsley.pdf.

Jonathan Bloom and Jeremy Orloff. Bayesian updating with continuous priors. *Notes for MIT 18.05*, 2014. URL https://ocw.mit.edu/courses/mathematics/18-05-introduction-to-probability-and-statistics-spring-2014/readings/MIT18_05S14_Reading13a.pdf.

Andrew J Blumberg and Michael Lesnick. Universality of the homotopy interleaving distance. 2017. URL https://arxiv.org/abs/1705.01690.

Richard Blute, Robin Cockett, and Robert A.G. Seely. Differential categories. *Mathematical structures in computer science*, 16(6), 2006.

Richard Blute, Robin Cockett, and Robert A.G. Seely. Cartesian differential categories. *Theory and Applications of Categories*, 22(23), 2009.

Magnus Botnan and Michael Lesnick. Algebraic stability of zigzag persistence modules. *Algebraic & Geometric Topology*, 18(6), Oct 2018. ISSN 1472-2747. doi: 10.2140/agt.2018.18.3133.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004. ISBN 0521833787. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike-20&path=ASIN/0521833787.

Stephen Boyd, Lin Xiao, and Almir Mutapcic. Subgradient methods. *Course notes at Stanford University*, 2003. URL https://web.stanford.edu/class/ee392o/subgrad_method.pdf.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. 2016. URL https://arxiv.org/abs/1611.08097.

Peter Bubenik and Jonathan A Scott. Categorification of persistent homology. *Discrete & Computational Geometry*, 51(3), 2014.

Peter Bubenik, Vin de Silva, and Vidit Nanda. Higher interpolation and extension for persistence modules. *SIAM Journal on Applied Algebra and Geometry*, 1(1), Jan 2017. ISSN 2470-6566. doi: 10.1137/16m1100472.

Ricardo Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. ISBN 978-3-642-37455-5. doi: 10.1007/978-3-642-37456-2_14.

Matteo Capucci, Bruno Gavranović, Jules Hedges, and Eigil Fjeldgren Rischel. Towards foundations of categorical cybernetics. 2021. URL https://arxiv.org/abs/2105.06332.

Gunnar Carlsson and Facundo Mémoli. Persistent clustering and a theorem of J. Kleinberg. 2008. URL https://arxiv.org/abs/0808.2241.

Gunnar Carlsson and Facundo Mémoli. Multiparameter hierarchical clustering methods. In *Classification as a Tool for Research*. Springer, 2010.

Gunnar Carlsson and Facundo Mémoli. Classifying clustering schemes. *Foundations of Computational Mathematics*, 13(2), 2013.

Benjamin P. Chamberlain, Emanuele Rossi, Dan Shiebler, Suvash Sedhain, and Michael M. Bronstein. Tuning word2vec for large scale recommendation systems. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 732–737, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375832. doi: 10.1145/3383313.3418486.

Kamalika Chaudhuri and Sanjoy Dasgupta. Rates of convergence for the cluster tree. In *Advances in Neural Information Processing Systems*, 2010.

Frédéric Chazal and Bertrand Michel. An introduction to topological data analysis: fundamental and practical aspects for data scientists. 2017. URL https://arxiv.org/abs/1710.04019.

Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J Guibas, and Steve Y Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, 2009.

Frédéric Chazal, Leonidas J Guibas, Steve Y Oudot, and Primoz Skraba. Persistence-based clustering in Riemannian manifolds. *Journal of the ACM (JACM)*, 60(6), 2013.

Frédéric Chazal, Vin De Silva, and Steve Oudot. Persistence stability for geometric complexes. *Geometriae Dedicata*, 173(1), 2014.

Kenta Cho and Bart Jacobs. Disintegration and Bayesian inversion via string diagrams. *Mathematical Structures in Computer Science*, 29(7), 2019.

Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin, and Dorette Pronk. Reverse derivative categories. 2019. URL https://arxiv.org/abs/1910.07065.

Taco Cohen, Mario Geiger, and Maurice Weiler. A general theory of equivariant CNNs on homogeneous spaces. 2020. URL https://arxiv.org/abs/1811.02017.

Taco S. Cohen and Max Welling. Group equivariant convolutional networks. 2016. URL https://arxiv.org/abs/1602.07576.

Taco S. Cohen, Mario Geiger, Jonas Koehler, and Max Welling. Spherical CNNs. 2018. URL https://arxiv.org/abs/1801.10130.

Taco S. Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge equivariant convolutional networks and the icosahedral CNN. 2019. URL https://arxiv.org/abs/1902.04615.

Geoffrey Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. 2021. URL https://arxiv.org/abs/2103.01931.

Jared Culbertson and Kirk Sturtz. Bayesian machine learning via category theory. 2013. URL https://arxiv.org/abs/1312.1445.

Jared Culbertson and Kirk Sturtz. A categorical foundation for Bayesian probability. *Applied Categorical Structures*, 22(4), 2014. doi: 10.1007/s10485-013-9324-9.

Jared Culbertson, Dan P Guralnik, Jakob Hansen, and Peter F Stiller. Consistency constraints for overlapping data clustering. 2016. URL https://arxiv.org/abs/1608.04331.

Jared Culbertson, Dan P Guralnik, and Peter F Stiller. Functorial hierarchical clustering with overlaps. *Discrete Applied Mathematics*, 236, 2018.

Justin M. Curry. Sheaves, cosheaves and applications. 2013. doi: 10.1.1.363.2881.

Pim de Haan, Taco Cohen, and Max Welling. Natural graph networks. 2020. URL https://arxiv.org/abs/2007.08349.

Joseph L. Doob. Application of the theory of martingales. In *Actes du Colloque International Le Calcul des Probabilités et ses applications*, pages 23–27, 1949.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Sven Eberhardt, Jonah Cader, and Thomas Serre. How deep is the feature analysis underlying rapid visual categorization? *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016. doi: 10.5555/3157096.3157220.

Conal Elliott. The simple essence of automatic differentiation. *Proceedings of the ACM on Programming Languages*, 2(ICFP), 2018. URL https://doi.org/10.1145/3236765.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 96, 1996.

Brendan Fong. Causal theories: A categorical perspective on Bayesian networks. 2013. URL https://arxiv.org/abs/1301.6201.

Brendan Fong and Michael Johnson. Lenses and learners. 2019. URL https://arxiv.org/abs/1903.03671.

Brendan Fong and David I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 2019. doi: 10.1017/9781108668804.

Brendan Fong, David Spivak, and Rémy Tuyéras. Backprop as functor: A compositional perspective on supervised learning. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2019. doi: https://doi.org/10.1109/LICS.2019.8785665.

Uwe Franz. What is stochastic independence? In *Non-commutativity, infinite-dimensionality and probability at the crossroads*. World Scientific, 2002. URL https://arxiv.org/abs/math/0206017.

Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370, 2020. doi: 10.1016/j.aim.2020.107239.

Tobias Fritz, Tomáš Gonda, Paolo Perrone, and Eigil Fjeldgren Rischel. Representable Markov Categories and Comparison of Statistical Experiments in Categorical Probability. 2020. URL https://arxiv.org/abs/2010.07416.

Bruno Gavranovic. Compositional deep learning. 2019. URL https://arxiv.org/abs/1907.08292.

Bruno Gavranović. Learning functors using gradient descent. *Applied Category Theory*, 2019.

Samuel Gerber, Tolga Tasdizen, and Ross Whitaker. Robust non-linear dimensionality reduction using successive 1-dimensional Laplacian eigenmaps. In *Proceedings of the 24th international conference on Machine learning*, 2007.

Malte Gerhold, Stephanie Lachs, and Michael Schürmann. Categorial Lévy processes. 2016. URL https://arxiv.org/abs/1612.05139.

Michele Giry. A categorical approach to probability theory. In *Categorical aspects of topology and analysis*. Springer, 1982. doi: https://doi.org/10.1007/BFb0092872.

Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2), 1950.

Charles R. Harris et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Kenneth D. Harris. Characterizing the invariances of learning algorithms using category theory. 2019. URL https://arxiv.org/abs/1905.02072.

Michael J Healy. Category theory applied to neural modeling and graphical representations. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3. IEEE, 2000.

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2017. doi: 10.1109/LICS.2017.8005137.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, 36(3), 2008.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2 (1), 1985.

Bart Jacobs. A channel-based perspective on conjugate priors. 2017. URL https://arxiv.org/abs/1707.00269.

Bart Jacobs. Categorical aspects of parameter learning. 2018. URL https://arxiv.org/abs/1810.05814.

David Jaz Myers. *Categorical Systems Theory*. unpublished, 2021. URL https://github.com/DavidJaz/DynamicalSystemsBook/tree/master/book.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. URL https://arxiv.org/abs/1412.6980.

Jon M Kleinberg. An impossibility theorem for clustering. In *Advances in neural information processing systems*, 2003.

Achim Klenke. *Probability Theory: A comprehensive Course*. Springer London, 2013.

Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. 2018. URL https://arxiv.org/abs/1306.2675.

Steven P Lalley. Lévy processes, stable processes, and subordinators. 2007. URL http://galton.uchicago.edu/~lalley/Courses/385/LevyProcesses.pdf.

Peter S. Landweber, Emanuel A. Lazar, and Neel Patel. On fiber diameters of continuous maps. *The American Mathematical Monthly*, 123(4), 2016.

Ken Lang. Newsweeder: Learning to filter netnews. *Machine Learning Proceedings*, 1995.

William Lawvere. The category of probabilistic mappings. *Unpublished preprint*, 1962.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553), 2015.

Tom Leinster. *Basic Category Theory*. Cambridge University Press, 2016.

Michael Lesnick and Matthew Wright. Interactive visualization of 2-d persistence modules. 2015. URL https://arxiv.org/abs/1512.00180.

Drew Linsley, Dan Shiebler, Sven Eberhardt, and Thomas Serre. Learning what and where to attend. 2018. URL https://arxiv.org/abs/1805.08819.

Saunders MacLane. *Categories for the Working Mathematician*. Springer Science+Business Media New York, 1978.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. 2019. URL https://arxiv.org/abs/1812.09902.

Alexander McCleary and Amit Patel. Edit distance and persistence diagrams over lattices. 2021. URL https://arxiv.org/abs/2010.07337.

Peter McCullagh. What is a statistical model? *Annals of statistics*, 2002.

Leland McInnes and John Healy. Accelerated hierarchical density clustering. 2017. URL https://arxiv.org/abs/1705.07321.

Leland McInnes, John Healy, and James Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. 2018. URL https://arxiv.org/abs/1802.03426.

Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Learning with invariances in random features and kernel models. 2021. URL https://arxiv.org/abs/2102.13219.

Jeffrey W Miller. A detailed treatment of Doob's theorem. 2018. URL https://arxiv.org/abs/1801.03122.

Michael C Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex systems*, 3(4), 1989.

Fabian Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

Boris Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5.

Louis B. Rall. *Automatic Differentiation: Techniques and Applications*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1981. ISBN 978-3-540-38776-3. doi: 10.1007/3-540-10861-0.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336), 1971.

Mitchell Riley. Categories of optics. 2018. URL https://arxiv.org/abs/1809.00738.

Alexander Rolle and Luis Scoccola. Stable and consistent density-based clustering. 2020. URL https://arxiv.org/abs/2005.09048.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088), 1986.

Venu Satuluri et al. SimClusters: Community-based representations for heterogeneous recommendations at Twitter. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '20, page 3183–3193, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403370. URL https://doi.org/10.1145/3394486.3403370.

Patrick Schultz and Ryan Wisnesky. Algebraic data integration. 2017. URL https://arxiv.org/abs/1503.03571.

Patrick Schultz, David I Spivak, and Ryan Wisnesky. Algebraic model management: A survey. In *International Workshop on Algebraic Development Techniques*. Springer, 2016.

Luis N Scoccola. Locally persistent categories and metric properties of interleaving distances. 2020. URL https://ir.lib.uwo.ca/cgi/viewcontent.cgi?article=9630&context=etd.

Dan Shiebler. Github Code, 2020a. URL https://github.com/dshieble/FunctorialHyperparameters.

Dan Shiebler. Github Code, 2020b. URL https://github.com/dshieble/FunctorialManifoldLearning.

Dan Shiebler. Functorial clustering via simplicial complexes. *Topological Data Analysis and Beyond at NeurIPS 2020*, 2020c. URL https://openreview.net/pdf?id=ZkDLcXCP5sV.

Dan Shiebler. Github Code, 2021a. URL https://github.com/dshieble/Generalized_Optimization.

Dan Shiebler. Github Code, 2021b. URL https://github.com/dshieble/Kan_Extensions.

Dan Shiebler. Categorical stochastic processes and likelihood. *Compositionality*, 3, April 2021c. ISSN 2631-4444. doi: 10.32408/compositionality-3-1.

Dan Shiebler. Flattening multiparameter hierarchical clustering functors. *International Conference on Geometric Science of Information*, 2021d. URL https://arxiv.org/pdf/2104.14734.pdf.

Dan Shiebler. Functorial manifold learning. *Applied Category Theory*, 2021e. URL https://arxiv.org/abs/2011.07435.

Dan Shiebler. Generalized optimization: A first step towards category theoretic learning theory. In *Intelligent Computing & Optimization*, pages 525–535. Springer International Publishing, 2022a. ISBN 978-3-030-93247-3.

Dan Shiebler. Kan extensions in data science and machine learning. 2022b. URL https://arxiv.org/abs/2203.09018.

Dan Shiebler, Alexis Toumi, and Mehrnoosh Sadrzadeh. Incremental monoidal grammars. 2020. URL https://arxiv.org/abs/2001.02296.

Dan Shiebler, Bruno Gavranovic, and Paul W. Wilson. Category theory in machine learning. *Applied Category Theory*, 2021. URL https://arxiv.org/abs/2106.07032.

Alex Simpson. Category-theoretic structure for independence and conditional independence. In Sam Staton, editor, *Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2018, Dalhousie University, Halifax, Canada, June 6-9, 2018*, volume 341 of *Electronic Notes in Theoretical Computer Science*, pages 281–297. Elsevier, 2018. doi: 10.1016/j.entcs.2018.03.028.

Samuel L. Smith, Erich Elsen, and Soham De. On the generalization benefit of noise in stochastic gradient descent. 2020. URL https://arxiv.org/abs/2006.15081.

Toby St. Clere Smithe. Bayesian updates compose optically. 2020. URL https://arxiv.org/abs/2006.01631.

David I Spivak. Metric realization of fuzzy simplicial sets. *Self published notes*, 2012.

David I. Spivak. Poly: An abundant categorical setting for mode-dependent dynamics. 2020. URL https://arxiv.org/abs/2005.01894.

David I. Spivak and Ryan Wisnesky. Relational foundations for functorial data migration. 2015. URL https://arxiv.org/abs/1212.5303.

David I Spivak and Ryan Wisnesky. Fast left-Kan extensions using the chase. *Preprint. Available at www. categoricaldata. net*, 2020.

David Sprunger and Shin-ya Katsumata. Differentiable causal computations via delayed trace. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2019.

Terence Tao. A review of probability theory, 2010. URL https://terrytao.wordpress.com/2010/01/01/254a-notes-0-a-review-of-probability-theory.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500), 2000.

Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*. Springer, 2015.

Alim Virani, Jay Baxter, Dan Shiebler, et al. Lessons learned addressing dataset bias in model-based candidate generation at Twitter. *Industrial Recommendation Systems at 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

Pauli Virtanen et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Edgar Y. Walker, R. James Cotton, Wei Ji Ma, and Andreas S. Tolias. A neural basis of probabilistic computation in visual cortex. *Nature Neuroscience*, 23, 2020. doi: 10.1038/s41593-019-0554-5.

Mitchell Wand. Continuation-based program transformation strategies. *J. ACM*, 27(1), 1980. ISSN 0004-5411. doi: 10.1145/322169.322183. URL https://doi.org/10.1145/322169.322183.

Paul Wilson and Fabio Zanasi. Reverse derivative ascent: A categorical approach to learning boolean circuits. *Electronic Proceedings in Theoretical Computer Science*, 333, Feb 2021. ISSN 2075-2180. doi: 10.4204/eptcs.333.17.

David Wishart. An algorithm for hierarchical classifications. *Biometrics*, 1969.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. doi: 10.1109/tnnls.2020.2978386.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. 2017. URL https://arxiv.org/abs/1708.07747.