Towards Data-Efficient Deep Learning with Meta-Learning and Symmetries

Jin Xu

Balliol College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy in Statistics

Trinity 2023

Acknowledgements

First and foremost, I want to express my deep gratitude to my supervisors, Prof. Yee Whye Teh and Dr. Tom Rainforth. Their unwavering support, careful guidance, and constant inspiration have been invaluable throughout my PhD journey. It has been a privilege to be mentored by them, who I regard as research role models. Their depth and breadth of knowledge have been both humbling and enlightening. Special acknowledgement goes to Yee Whye, who has always been considerate and ready to help in tough times. My heartfelt thanks go to Tom for his guidance during the challenging times brought on by the pandemic.

I would like to extend my gratitude to all my collaborators Hyunjik Kim, Jean-Francois Ton, Adam Kosiorek, Emilien Dupont, and Kaspar Märtens. Their expertise and feedback have been crucial in improving my work and I learn a great deal from them. A big thank you to Prof. Ryan Adams from Princeton University and to my internship hosts, James Hensman and Max Croci at Microsoft Research. Their mentorship outside of my PhD life has been an indispensable part of my research experience.

Moreover, I feel extremely fortunate to be surrounded by amazing and caring friends whose names are not possible to enumerate here. Among them are Emilien Dupont, Jean-Francois Ton, Charline Le Lan, Bobby He, Sheheryar Zaidi, Qinyi Zhang, Guneet Dhillon, Andrew Campbell, Chris Williams, Carlo Alfano, Faaiz Taufiq, Anna Menacher and others from our lovely office 1.17, Hanwen Xing, Yanzhao Yang, Ning Miao, Chao Zhang, Yutong lu, Yixuan He, Xi Lin, Yuan Zhou, Fan Wu, Bohao Yao from the department of statistics, Dunhong Jin, Sihan Zhou, Sijia Yao, Huining Yang, Kevin Wang, Natalia Hong, Hang Yuan, Kangning Zhang, Chengyang Wang and many others from other departments at Oxford, Deniz Oktay, Sulin Liu, Jenny Zhan and others from Princeton University, internship peers at Microsoft Research including Alexander Meulemans, Saleh Ashkboos from ETH.

A special thanks to all university and department staff, especially Chris Cullen for his kind and patient support during difficult times, and to Joanna Stoneham, Stuart

McRobert, and others who ensured a smooth PhD experience.

Finally, above all, my deepest thanks go to Yifan Yu for her love and companionship. She immensely enriched my time in Oxford, bringing colour and joy to my life. Additionally, I am eternally grateful to my parents Chengxiang Xu and Feng Chen for giving me the freedom to pursue my passions and for their unquestioning support throughout this journey.

Abstract

Recent advances in deep learning have been significantly propelled by the increasing availability of data and computational resources. While the abundance of data enables models to perform well in certain domains, there are real-world applications, such as in the medical field, where the data is scarce or difficult to collect. Furthermore, there are also scenarios where the large dataset is better viewed as lots of related small datasets, and the data becomes insufficient for the task associated with one of the small datasets. It is also noteworthy that human intelligence often requires only a handful of examples to perform well on new tasks, emphasizing the importance of designing data-efficient AI systems. This thesis delves into two strategies to address this challenge: meta-learning and symmetries. Meta-learning approaches the data-rich environment as a collection of many small, individual datasets. Each of these small datasets represents a distinct task, yet there is underlying shared knowledge between them. Harnessing this shared knowledge allows for the design of learning algorithms that can efficiently address new tasks within similar domains. In comparison, symmetry is a form of direct prior knowledge. By ensuring that models' predictions remain consistent despite any transformation to their inputs, these models enjoy better sample efficiency and generalization.

In the subsequent chapters, we present novel techniques and models which all aim at improving the data efficiency of deep learning systems. Firstly, we demonstrate the success of encoder-decoder style meta-learning methods based on Conditional Neural Processes (CNPs). Secondly, we introduce a new class of expressive meta-learned stochastic process models which are constructed by stacking sequences of neural parameterised Markov transition operators in function space. Finally, we propose group equivariant subsampling/upsampling layers which tackles the loss of equivariance in conventional subsampling/upsampling layers. These layers can be used to construct end-to-end equivariant models with improved data-efficiency.

Contents

Chapter 1

Introduction

1.1 Motivation

Recent breakthroughs in deep learning can be largely attributed to the vast amount of data available and the advancement of computational resources [Deng et al., 2009, Raina et al., 2009, Silver et al., 2016, Jumper et al., 2021, Brown et al., 2020a]. While training on large datasets enables deep learning models to excel in certain tasks, many real-world applications only provide limited data for a specific task. For instance, in medical fields, obtaining data, especially for rare diseases, is challenging and often expensive. In drug development or recommendation systems, there will always be insufficient data for new drugs/users, even though abundant data exists for other drugs or users. Therefore, to apply deep learning to these fields, it is vital to develop systems that are data-efficient. Moreover, for advanced AI systems, data-efficiency can be a crucial ingredient: Firstly, AI systems should be able to generalize beyond specific data distributions without relying on data; for instance, an image recognition system should recognize objects regardless of their position or orientation. Secondly, human intelligence can often solve new tasks with just a few examples. Thus, for AI to emulate human-like intelligence, it should also have such capability.

From a Bayesian perspective, learning involves updating our beliefs about a model (represented by θ) given the data, i.e. $p(\theta|\mathcal{D}_{\text{data}})$. For a model to learn efficiently from a small amount of data, it's important to start with a good initial guess or "prior" $p(\theta)$. In this paper, we look at two directions to obtain such prior for data-efficient learning: The first is meta-learning, which learns the prior (or the shared knowledge) from

similar tasks. It can be understood as "learning to learn more efficiently". The second is symmetries in deep learning, which serves as a known prior for certain problems. Symmetry, a fundamental concept in physics, represents a form of prior knowledge that is ubiquitously observed throughout our physical world.

Meta-learning tackles a specific scenario in which the vast pool of data can be viewed as many small datasets, each representing a distinct task. Yet, these tasks contain underlying shared knowledge that can be harnessed to address new tasks within the same category. This scenario is prevalent in many applications. Take, for instance, an online retail company with data from customers worldwide. The data associated with each user is typically sparse. In this context, predicting behaviours for each user constitutes an individual task, but patterns among different users often exhibit similarities. Meta-learning algorithms are designed to handle such circumstances. The goal of meta-learning is to learn data-efficient learning algorithms that can later be applied to a particular task. The training data for meta-learning comprises numerous related tasks, each with a limited set of data points. After the meta-learning phase, the learned learning algorithms can solve a new task in a data-efficient manner. In contrast, the aim of conventional supervised learning is just to learn a predictive model.

Meta-learning problems can be tackled from various perspectives, and these approaches can be understood through different viewpoints such as optimization-based approaches[Ravi and Larochelle, 2016, Finn et al., 2017a], metric-based approaches[Koch, 2015, Vinyals et al., 2016, Sung et al., 2018, Snell et al., 2017], and model-based approaches[Santoro et al., 2016, Mishra et al., 2018, Garnelo et al., 2018a], among others. Note that these views are not exclusive. For example, methods such as prototypical Networks [Snell et al., 2017], MAML [Finn et al., 2017a], ML-PIP[Gordon et al., 2018] etc. can be reformulated under a model-based framework that uses an encoder-decoder setup. In this setup, the encoder produces a task representation using training data, and the decoder then makes predictions based on the task representation. These approaches transform the meta-learning challenge to resemble a regular learning problem involving sequences, and it is also more computationally efficient if no gradient computation is involved in both the encoder and the decoder like CNP-type models [Garnelo et al., 2018a]. Our study in Chapter 3 explicitly adopts this encoder-decoder framework for meta-learning. By using a functional task representation, and iteratively updating the representation directly in function space,

we demonstrate that encoder-decoder approaches without gradient information can also be competitive with other approaches, which has not been shown before.

Furthermore, because training data for each task in meta-learning is often limited, uncertainty estimation becomes crucial. Stochastic Processes (SPs) (e.g. Gaussian Processes (GPs)) can be used to make predictions with uncertainty estimation. Thus, learning these processes can be seen as a way to approach meta-learning with uncertainty in mind. In Chapter 4, we propose a new framework to construct expressive neural parameterised SPs by parameterising Markov transitions in function space.

Unlike meta-learning above, which discovers shared knowledge from related tasks, symmetry serves as a direct form of prior or inductive bias, integrated into deep learning models without the need for pre-training. Symmetries refer to transformations that maintain certain properties of an object of interest unchanged. These include transformations such as image translation, rotation, or permutation of set elements. By incorporating these symmetries into deep learning models, ensuring that the outputs remain consistent (the same or undergo the corresponding transformation) despite input transformations, the model inherently generalizes to transformed inputs. Consequently, deep learning models equipped with these symmetries not only become more data-efficient but also generalize better. A simple example of this is Convolutinal Neural Networks (CNNs), which are invariant to input translations for classification tasks, and perform significantly better compared to plain feed-forward networks. Earlier research has introduced many methods to build convolutional [Cohen and Welling, 2016, 2017, Cohen et al., 2019] and attention blocks [Hutchinson et al., 2021, Fuchs et al., 2020] that are equivariant w.r.t. to various symmetries. However, the pooling layers or subsampling/upsampling layers commonly used in various deep learning architectures break these symmetries [Zhang, 2019]. In Chapter 5, we present group equivariant subsampling/upsampling layers that have exact equivariance.

1.2 Thesis outline

In Chapter 2, we provide a short introduction to meta-learning, neural processes and symmetries in deep learning, to set the stage for later chapters.

In Chapter 3, we introduce an iterative functional encoder-decoder method for supervised meta-learning, which is based on Neural Processes (NPs) [Garnelo et al.,

2018a,b]. On standard few-shot classification benchmarks like miniImageNet and tieredImageNet, it is demonstrated that meta-learning methods based on the neural process family can be competitive or even outperform gradient-based methods such as MAML [Finn et al., 2017a] and LEO [Rusu et al., 2019].

In Chapter 4, we introduce Markov Neural Processes (MNPs), a new class of Stochastic Processes (SPs) which are constructed by stacking sequences of neural parameterised Markov transition operators in function space. Therefore, the proposed iterative construction adds substantial flexibility and expressivity to the original framework of Neural Processes (NPs) without compromising consistency or adding restrictions. Our experiments demonstrate clear advantages of MNPs over baseline models on a variety of tasks. It's noteworthy that SP models can be viewed through a meta-learning lens. So the proposed method can also be seen as a meta-learning approach with principled uncertainty estimation.

Chapter 5, we first introduce translation equivariant subsampling/upsampling layers that can be used to construct exact translation equivariant CNNs. We then generalise these layers beyond translations to general groups, thus proposing group equivariant subsampling/upsampling. We use these layers to construct group equivariant autoencoders (GAEs) that allow us to learn low-dimensional equivariant representations. We empirically verify on images that the representations are indeed equivariant to input translations and rotations, and thus generalise well to unseen positions and orientations. We further use GAEs in models that learn object-centric representations on multi-object datasets, and show improved data efficiency and decomposition compared to non-equivariant baselines.

In Chapter 6, we summarize our findings and explore potential avenues for future research to further advance the field.

1.3 Papers

This is an integrated thesis and includes the following published papers:

Chapter 3 contains:

Xu, J., Ton, J. F., Kim, H., Kosiorek, A., & Teh, Y. W. Metafun: Meta-

learning with iterative functional updates. International Conference on Machine Learning (ICML), 2020 [Xu et al., 2020]

Chapter 4 contains:

Xu, J., Kim, H., Rainforth, T., & Teh, Y. (2021). Group equivariant subsampling. Advances in Neural Information Processing Systems (NeurIPS), 2021 [Xu et al., 2021]

Chapter 5 contains

Xu, J., Dupont, E., Märtens, K., Rainforth, T., & Teh, Y. W. (2023). Deep Stochastic Processes via Functional Markov Transition Operators. Advances in Neural Information Processing Systems (NeurIPS), 2023 [Xu et al., 2023]

Chapter 2

Background

2.1 Meta-learning

2.1.1 Conventional supervised learning and meta-learning

In conventional supervised learning, the objective is to learn a function f that maps an input feature vector $\mathbf{x} \in \mathcal{X}$ to an output label $\mathbf{y} \in \mathcal{Y}$. Learning is based on example input-output pairs in a training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$. Common types of supervised learning tasks include regression where output labels are real-valued, and classification where the output labels represent different classes. The function f, often referred to as the predictive model, is a member of a hypothesis class,

$$\mathcal{H} := \{f | f(\mathbf{x}; \phi), \phi \in \mathbb{R}^{d_\phi}\}.$$

For each task, there is a risk function $\ell(\mathbf{y}, f(\mathbf{x}))$ which measures prediction error. As an example, in the context of a regression task, ℓ often takes the form of a squared error, $\ell(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$. The training process of the model f translates to solving an optimization problem defined as follows:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \hat{R}(f; \mathcal{D}_{\text{train}}) = \arg\min_{f \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} \ell(\mathbf{y}_i, f(\mathbf{x}_i)). \tag{2.1}$$

It is called *empirical risk minimization* because this objective is an estimation of the population risk $\mathbb{E}_{(\mathbf{x}_i, \mathbf{y}_i) \sim p(\mathbf{x}, \mathbf{y})}[\ell(y_i, f(\mathbf{x}_i))]$ based on the empirical distribution of training data.

After training, the model should generalize effectively when presented with a test set, denoted as $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=m+1}^{n}$. The model's performance can be assessed using the test risk $\hat{R}(f; \mathcal{D}_{\text{test}})$ which serves as an estimate of the overall population risk using unseen data.

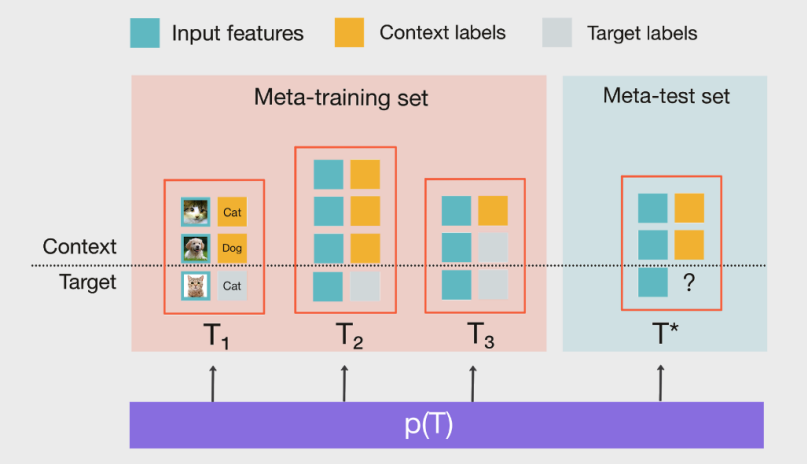


Figure 2.1: Data for a meta-classification problem. Both the meta-training and meta-test sets consist of tasks (red rectangles) and are presumed to come from the same task distribution $p(\mathcal{T})$. Each of these tasks encompasses its own task-specific training and test sets, which are commonly referred to as the context (yellow labels) and the target (grey labels) respectively.

In practice, it is common to have scenarios where lots of supervised learning tasks are related to each other, yet the number of data points for each individual task is limited. Meta-learning emerges as a new learning paradigm to address such challenges. Specifically, we have a meta-training set defined as $\mathcal{M}_{\text{train}} = \{(\mathcal{D}_{\text{train}}^{(j)}, \mathcal{D}_{\text{test}}^{(j)}, \ell^{(j)})\}_{j=1}^{M}$ and a meta-test set given by $\mathcal{M}_{\text{test}} = \{(\mathcal{D}_{\text{train}}^{(j)}, \mathcal{D}_{\text{test}}^{(j)}, \ell^{(j)})\}_{j=M+1}^{N}$. Each element in these meta-datasets is a tuple consisting of a training set (called the context), a test set (called the target) and a risk function (typically the same within a meta-dataset). This 3-tuple characterizes a task $\mathcal{T}_j$ (see Figure 2.1 illustration). In supervised learning, we use training data to train a predictive model, hoping it can generalize across the entire data distribution. In meta-learning, the assumption is that there is a common task distribution, denoted as $p(\mathcal{T})$, from which both the meta-training set and the meta-test set are drawn. Meta-learning algorithms aim to use meta-training data to discover learning algorithms that can generalize across the entire task distribution.

More specifically, a learning algorithm for a supervised learning task takes in a training

set $\mathcal{D}_{\text{train}}$, a risk function ℓ and outputs a predictive model, written as:

$$\hat{f} = \Phi_{\text{ALGO}}(\mathcal{D}_{\text{train}}, \ell). \tag{2.2}$$

Since ℓ is usually fixed, we will omit the dependency on it in subsequent discussions. For a particular task, the learning algorithm Φ_{ALGO} can be evaluated by the test risk of the learned predictive model, denoted as:

$$\hat{R}(\hat{f}; \mathcal{D}_{\text{test}}). \tag{2.3}$$

Meta-learning finds a learning algorithm based on tasks from the meta-training set $\mathcal{M}_{\text{train}}$, so that this learning algorithm can be more efficiently applied to new tasks, and generalizes across the task distribution $p(\mathcal{T})$. The meta-learning algorithm can be represented as:

$$\Phi_{\text{ALGO}} = \text{MetaAlgo}(\mathcal{M}_{\text{train}}). \tag{2.4}$$

To evaluate the meta-learning algorithm, we can compute:

$$\mathcal{L}_{\text{meta}}(\Phi_{\text{ALGO}}; \mathcal{M}_{\text{test}}) = \frac{1}{N-M} \sum_{j=M+1}^{N} \hat{R}(\Phi_{\text{ALGO}}(\mathcal{D}_{\text{train}}^{(j)}, \ell^{(j)}); \mathcal{D}_{\text{test}}^{(j)}). \tag{2.5}$$

While it resembles the test loss in supervised learning, the aggregated test risk for a task replaces the traditional risk function for a data point.

It is worth noting that while we focus on supervised learning tasks here, meta-learning can be extended to unsupervised learning [Edwards and Storkey, 2016, Reed et al., 2018, Hsu et al., 2018] or reinforcement learning [Wang et al., 2016, Finn et al., 2017a,b].

2.1.2 Different views of meta-learning

Bi-level optimization view Let us assume both the predictive model f and the learning algorithm Φ_{ALGO} can be parameterised, and the parameters are denoted as ϕ and θ accordingly. That is to say, the learning algorithm can be written as:

$$\phi = \Phi_{\text{ALGO}}(\mathcal{D}_{\text{train}}; \theta). \tag{2.6}$$

Meta-learning can be formulated as the following bi-level optimization problem:

$$\min_{\theta} \ \mathcal{L}_{\text{meta}}(\theta; \mathcal{M}_{\text{train}}) = \min_{\theta} \ \frac{1}{M} \sum_{j=1}^{M} \hat{R}(\phi_j(\theta); \mathcal{D}_{\text{test}}^{(j)}) \tag{2.7}$$

where task-specific parameter ϕ_j depends on θ through the inner-loop optimization:

$$\phi_j(\theta) = \Phi_{\text{ALGO}}(\mathcal{D}_{\text{train}}^{(j)}; \theta) \tag{2.8}$$

Many meta-learning algorithms are developed based on this bi-level optimization view, such as Finn et al. [2017a], Nichol et al. [2018], Ravi and Larochelle [2016].

Hierarchical model view From a probabilistic perspective, the generative process for each task $\mathcal{T}_j$ can be expressed as:

$$\theta \sim p(\theta), \ \phi_j \sim p(\phi_j|\theta), \ \mathbf{y}_i^{(j)} \sim p(\mathbf{y}_i^{(j)}|\mathbf{x}_i^{(j)}\phi_j, \theta) \tag{2.9}$$

Both the training set $\mathcal{D}_{\text{train}}^{(j)}$ and the test set $\mathcal{D}_{\text{test}}^{(j)}$ follow the same distribution (as illustrated in Figure 2.2). This can be seen as a probabilistic hierarchical model where θ indicates the high-level global parameters for all tasks and ϕ_j denotes the low-level local parameters for each task. In this context, meta-learning is about inferring θ from lots of tasks in the meta-training set, that is $p(\theta|\mathcal{M}_{\text{train}})$. Learning, on the other hand, infers ϕ_j given the training set $\mathcal{D}_{\text{train}}^{(j)}$ for task $\mathcal{T}_j$, that is $p(\phi_j|\theta, \mathcal{D}_{\text{train}}^{(j)})$.

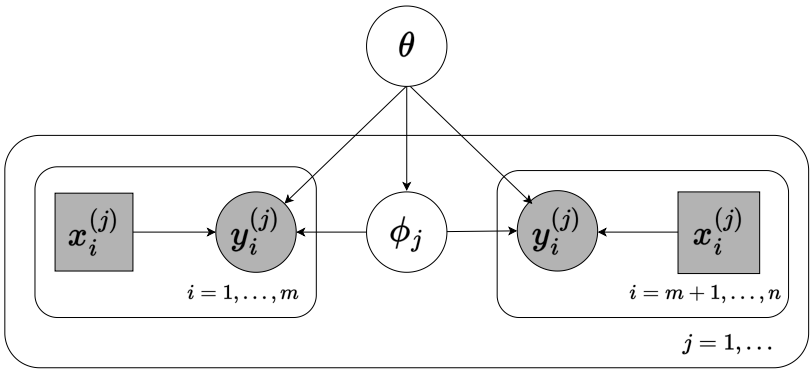


Figure 2.2: Meta-learning as hierarchical models (A remake of Figure 1 in Gordon et al. [2018]). Task-specific parameter ϕ_j depends on the global parameter θ. Data points in both the context and the target have the same generative process, which depend on both θ and ϕ_j.

Note that $p(\phi_j|\theta)$ can be seen as a prior for task $\mathcal{T}_j$ conditioned on θ. Therefore, meta-learning can be seen as learning an empirical prior from the meta-training set. Finn et al. [2018], Requeima et al. [2019] adopts this view.

Model-based view A learning algorithm $f = \Phi_{\text{ALGO}}(\mathcal{D}_{\text{train}})$ can be seen as a function that takes in the entire training set and outputs a predictive model. The model is then used to make predictions on test data in $\mathcal{D}_{\text{test}}$. The learning and prediction processes can thus be conceptualized as sequence-to-sequence mappings. For the sake of brevity, let's use a concise notation for data sequences, such as $\mathbf{x}_{1:n} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$. For a specific task $\mathcal{T}_j$, making predictions for test set data points based on those from the training set can be described as the following inference task

$$p(\mathbf{y}_{m+1:n}|\mathbf{x}_{m+1:n}, \mathbf{x}_{1:m}, \mathbf{y}_{1:m}). \tag{2.10}$$

From this perspective, meta-learning is about creating this conditional model. Meta-learning only differs from conventional supervised learning in that both the inputs and output labels are data sequences. In order to formulate this sequence-to-sequence model, one can harness attention mechanisms, as in Mishra et al. [2018]. When the predictive model is parametric, Φ_{ALGO} outputs the parameters ϕ. Then conditioned on ϕ, the predictive model generates predictions. In this case, Φ_{ALGO} and f can be seen as the encoder and the decoder. And ϕ serves as the latent variables. This viewpoint is embraced by NPs [Garnelo et al., 2018a,b].

2.1.3 Common approaches to meta-learning

From various perspectives on meta-learning, diverse approaches have been developed to tackle these problems. To provide an overview, we roughly categorize previous work into optimization-based, metric-based, and model-based approaches. However, it is worth noting that these categories are neither exclusive nor exhaustive. Some methods may not fit neatly into these categories, while others can be understood through multiple lenses.

Optimization-based approaches Deep learning models are trained with backpropagation, and first-order optimizers such as Adam optimizers [Kingma and Ba, 2014] are employed to optimize the training objectives. However, such optimization algorithms are designed to optimize over a large number of training examples, and they also cannot converge within a few optimization steps. Optimization-based meta-learning methods learn optimization procedures over lots of related tasks such that the learned optimizers can be applied to a new task to handle a small training and converge within a few steps. In particular, one idea is to learn the updating rule of optimization with neural networks. Ravi and Larochelle [2016] uses Long short-term memory (LSTM)

to model the updating rules where the updates of the cell states are analogous to the gradient-based updating. Instead of learning the entire optimization procedures, Finn et al. [2017a] finds that learning a good initialization can already solve lots of meta-learning tasks and this approach is both simple and efficient.

Metric-based approaches Metric-based meta-learning is based on comparisons. This is similar to some nonparametric supervised learning methods such as K-nearest neighbours. During the meta-learning phase, metric-based methods learn how to compare. During test time, the learned comparison metrics are used to make predictions. The comparison is often via a pairwise comparison function k_θ which measures the similarity between input features of two data points $k_\theta(\mathbf{x}, \mathbf{x}')$. Notable examples of metric-based approaches include siamese neural network[Koch, 2015], matching networks [Vinyals et al., 2016], relation networks[Sung et al., 2018], prototypical networks [Snell et al., 2017], etc.

Model-based approaches Recall that under the model-based view in Section 2.1.2, the task of making predictions for tests inputs $\mathbf{x}_{m+1:n}$ based on training data $\mathbf{x}_{1:m}$, $\mathbf{y}_{1:m}$ can be described as modelling/inferring $p(\mathbf{y}_{m+1:n}|\mathbf{x}_{m+1:n}, \mathbf{x}_{1:m}, \mathbf{y}_{1:m})$. Therefore, from this perspective, meta-learning transforms into a traditional supervised learning task where both inputs and outputs are sequences. Santoro et al. [2016] delves into this by treating learning and prediction as akin to storing and querying within a neural memory. Meanwhile, Mishra et al. [2018] employs attention networks to model these sequence-to-sequence mappings. Recently, the neural process family [Garnelo et al., 2018a,b, Gordon et al., 2019, Bruinsma et al., 2021] adopts an encoder-decoder framework to infer such conditional distributions on sequences (see next section for details). In particular, when permutation invariance/equivariance of training/test sets are considered, these sequences can be treated as sets, and neural networks e.g. deep sets [Zaheer et al., 2017], set transformers [Lee et al., 2019a] that maps from sets to sets can be employed in these methods.

2.2 Neural processes

The NP family [Garnelo et al., 2018a,b, Kim et al., 2018, Gordon et al., 2019] provide more adaptable and scalable alternatives to the conventional nonparametric SPs such as GPs. A SP is a (typically infinite) collection of random variables, whose distribution can be indirectly defined via marginal distributions over arbitrary finite subsets of

these random variables. NPs use neural networks to parameterise these marginal distributions while guaranteeing the SP is properly defined. Enhancing NPs remains an active area of research.

2.2.1 Stochastic processes

A SP can be considered as a random function $F : \mathcal{X} \to \mathcal{Y}$ where inputs can be regarded as indexing the output random variables. This random function f maps inputs $\mathbf{x} \in \mathcal{X}$ to outputs $\mathbf{y} \in \mathcal{Y}$. It is difficult to define a proper density directly in function space, but a SP can be *indirectly* defined via a collection of marginal distributions, denoted as $\{p_{\mathbf{x}_{1:n}}(\mathbf{y}_{1:n})\}_{\mathbf{x}_{1:n} \in \mathcal{X}^n}$. Kolmogorov's Consistency Theorem states that the SP is properly defined if these marginals satisfy the *exchangeability* condition:

$$p_{\mathbf{x}_{1:n}}(\mathbf{y}_{1:n}) = p_{\pi(\mathbf{x}_{1:n})}(\pi(\mathbf{y}_{1:n})) := \; p_{\mathbf{x}_{\pi(1)},\ldots,\mathbf{x}_{\pi(n)}}(\mathbf{y}_{\pi(1)}, \ldots, \mathbf{y}_{\pi(n)}), \tag{2.11}$$

and the *consistency* condition:

$$p_{\mathbf{x}_{1:m}}(\mathbf{y}_{1:m}) = \int p_{\mathbf{x}_{1:n}}(\mathbf{y}_{1:n}) \, \mathrm{d}\mathbf{y}_{m+1:n} \quad \forall 1 \leq m < n. \tag{2.12}$$

2.2.2 Neural processes as stochastic processes

A NP [Garnelo et al., 2018c] characterizes a SP by providing a collection of exchangeable and consistent marginal densities, which are parameterized by neural networks. These marginal densities take the form below:

$$p_{\mathbf{x}_{1:n}}(\mathbf{y}_{1:n}; \theta) = \int p_\theta(\mathbf{z}) \prod_{i=1}^{n} \mathcal{N}(\mathbf{y}_i | \mu_\theta(\mathbf{x}_i, \mathbf{z}), \sigma^2) \mathrm{d}\mathbf{z}. \tag{2.13}$$

In this equation, $\mathbf{z}$ is a latent variable capturing dependencies across data points. The term $\mathcal{N}(\mathbf{y}_i | \mu_\theta(\mathbf{x}_i, \mathbf{z}), \sigma^2)$ models the conditional distribution of each output $\mathbf{y}_i$, given $\mathbf{z}$ and the corresponding input $\mathbf{x}_i$. Both $p_\theta(\mathbf{z})$ and $\mu_\theta(\mathbf{x}_i, \mathbf{z})$ can be parameterized by deep neural networks. This factorized expression of the marginal density corresponds to a conditional version of de Finetti's Theorem. In fact, it can be proved that the marginal density of a SP can always be expressed with this factorisation.

For two arbitrary subsets of data points: a context denoted as $\mathcal{D}_\mathcal{C} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$, and a target denoted as $\mathcal{D}_\mathcal{T} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=m+1}^n$, the conditional density $p(\mathcal{D}_\mathcal{T} | \mathcal{D}_\mathcal{C}) := p_{\mathbf{x}_{m+1:n} | \mathbf{x}_{1:m}}(\mathbf{y}_{m+1:n} | \mathbf{y}_{1:m})$ can be computed. Notably, the collection of conditional

distributions, $\{p_{\mathbf{x}_{m+1:n}|\mathbf{x}_{1:m}}(\mathbf{y}_{m+1:n}|\mathbf{y}_{1:m})\}_{\mathbf{x}_{m+1:n}}$, satisfies both exchangeability and consistency, and thus defines a valid *posterior* SP. Many NP applications essentially revolve around computing these conditional distributions. Under the formulation in Equation (2.13), the conditional density can be calculated with:

$$p_{\mathbf{x}_{m+1:n}|\mathbf{x}_{1:m}}(\mathbf{y}_{m+1:n}|\mathbf{y}_{1:m};\theta) = \int p(\mathbf{z}|\mathbf{x}_{1:m},\mathbf{y}_{1:m};\theta)\Big(\prod_{i=m+1}^{n} p(\mathbf{y}_i|\mathbf{z},\mathbf{x}_i;\theta)\Big)\mathrm{d}\mathbf{z}. \tag{2.14}$$

As computing $p(\mathbf{z}|\mathbf{x}_{1:m},\mathbf{y}_{1:m};\theta)$ is intractable, NP introduces an inference model (or an encoder) denoted as $q(\mathbf{z}|\mathbf{x}_{1:m},\mathbf{y}_{1:m};\phi)$. The generative model and the inference model are trained together with training objectives discussed in Section 2.2.3.

The original NP tends to underfit in practice. Hence, subsequent research, such as Attentive Neural Processes (ANPs) [Kim et al., 2018] and Convolutional Neural Processes (CONVNPs) [Gordon et al., 2019, Foong et al., 2020], has enhanced expressivity by incorporating attention modules and convolutional layers into the model.

2.2.3 Neural process training objectives

To learn a distribution over functions with NPs, we sample random functions and evaluate them at a finite set of input locations $\mathbf{x}_{1:n}$ to get the corresponding outputs $\mathbf{y}_{1:n}$. These data points are then split into the context $\mathcal{D}_\mathcal{C} = \{(\mathbf{x}_i,\mathbf{y}_i)\}_{i=1}^m$ and the target $\mathcal{D}_\mathcal{T} = \{(\mathbf{x}_i,\mathbf{y}_i)\}_{i=m+1}^n$. To better reflect the desired test-time behaviour, we maximize the log-marginal predictive likelihood $\log p_{\mathbf{x}_{m+1:n}|\mathbf{x}_{1:m}}(\mathbf{y}_{m+1:n}|\mathbf{y}_{1:m})$.

Given the generative model parameterised by θ and the inference model parameterised by ϕ, the original NPs optimize a variational lower bound to the log-marginal predictive likelihood, denoted as $-\mathcal{L}_{\mathrm{VI}}(\theta,\phi)$:

$$\begin{aligned}
&\log p_{\mathbf{x}_{m+1:n}|\mathbf{x}_{1:m}}(\mathbf{y}_{m+1:n}|\mathbf{y}_{1:m}) \\
&\geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x}_{1:n},\mathbf{y}_{1:n};\phi)}\Big[\sum_{i=m+1}^{n}\log p(\mathbf{y}_i|\mathbf{z},\mathbf{x}_i;\theta) + \log\frac{q(\mathbf{z}|\mathbf{x}_{1:m},\mathbf{y}_{1:m};\phi)}{q(\mathbf{z}|\mathbf{x}_{1:n},\mathbf{y}_{1:n};\phi)}\Big] \\
&= -\mathcal{L}_{\mathrm{VI}}(\theta,\phi).
\end{aligned} \tag{2.15}$$

NPs train the model by minimising $\mathcal{L}_{\mathrm{VI}}(\theta,\phi)$ over randomly sampled pairs of context and target, which indirectly pushes up the log-marginal predictive likelihood.

However, Foong et al. [2020] proposes an alternative maximum likelihood objective to train NPs which usually leads to better predictive performance. This objective directly

estimates Equation (2.14) with Monte Carlo estimation, i.e.

$$\begin{aligned}
&\log p_{\mathbf{x}_{m+1:n}|\mathbf{x}_{1:m}}(\mathbf{y}_{m+1:n}|\mathbf{y}_{1:m}) \\
&= \log \mathbb{E}_{q(\mathbf{z}|\mathbf{x}_{1:m},\mathbf{y}_{1:m})}\Big[\sum_{i=m+1}^{n} \log p(\mathbf{y}_i|\mathbf{z},\mathbf{x}_i;\theta)\Big] \\
&\approx \log \frac{1}{L}\sum_{l=1}^{L}\sum_{i=m+1}^{n} \log p(\mathbf{y}_i|\mathbf{z},\mathbf{x}_i;\theta) \\
&= -\mathcal{L}_{\mathrm{ML}}(\theta,\phi)
\end{aligned} \tag{2.16}$$

This estimator has high variance so it often needs a large L to work sufficiently well.

2.2.4 A meta-learning perspective

NPs can be interpreted from a meta-learning perspective. Specifically, the context can be equated to the training set for a specific task, while the target corresponds to the test set, i.e. $\mathcal{D}_{\mathcal{C}}^{(j)} = \mathcal{D}_{\text{train}}^{(j)}$, $\mathcal{D}_{\mathcal{T}}^{(j)} = \mathcal{D}_{\text{test}}^{(j)}$. The conditional mean $\mu_\theta(\cdot, z)$ serves as the predictive model, while the variance σ^2 represents observational noise. Note that each latent variable $\mathbf{z}$ yields a unique predictive model. The posterior distribution over these predictive models, conditioned on the context (training set) $\mathcal{D}_{\mathcal{C}}^{(j)}$ is inferred with the approximate inference model $q(\mathbf{z}|\mathbf{x}_{1:m}, \mathbf{y}_{1:m}; \phi)$.

NPs can be used to solve meta-learning tasks. In the meta-learning phase, both the generative model (referred to as the decoder) and the inference model (the encoder) are trained by optimizing either $\mathcal{L}_{\mathrm{VI}}$ or $\mathcal{L}_{\mathrm{ML}}$ with respect to parameters θ and ϕ. The learning phase for a new task only involves a forward pass through the encoder while making predictions is achieved via the decoder's forward pass. Notably, the forward passes for both the encoder and decoder in NPs don't require gradient information. For the original NPs, the computational cost for both learning and prediction linearly scales with the number of training data points. This is particularly attractive when compared with other meta-learning algorithms. Furthermore, NPs provides uncertainty estimation along with predictions, which is a useful feature in a low-data regime.

2.3 Symmetries in deep learning

2.3.1 Group, coset and quotient space

A *group* G is a set of elements equipped with a binary operation (denoted as $\cdot$) that satisfies the following group axioms:

1. (Closure) For all $a, b \in G$, $a \cdot b \in G$.
2. (Associative) For all $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
3. (Identity element) There exists an identity element e in G such that, for any $a \in G$ we have $e \cdot a = a \cdot e = a$.
4. (Inverse element) For each $a \in G$, there exists an element $b \in G$ such that $a \cdot b = b \cdot a = e$ where e is the identity element.

The centered dot $\cdot$ can sometimes be omitted if there is no ambiguity.

In this work, we are mainly interested in symmetry groups where each group element is associated with a symmetry of a pattern, which is a transformation that leaves the pattern invariant. In symmetry groups, the binary operation corresponds to composition of transformations.

A subset H contained within G is a *subgroup* of G if it forms a group on its own under the same binary operation. Given a subgroup H and an arbitrary group element $g \in G$, one can define *left cosets* of H as follows:

$$gH = \{g \cdot h \mid h \in H\}$$

The left cosets of H form a partition of G for any choice of H, i.e. the union of all cosets is G and all cosets defined above are either identical or have empty interception. The set of all left cosets is called the *quotient space* and is denoted as $G/H = \{gH \mid g \in G\}$.

As an example, all integers $\mathbb{Z}$ under addition forms a group and all multiples of n, denoted as $n\mathbb{Z}$ is a subgroup of $\mathbb{Z}$. For any integer $k \in \mathbb{Z}$, the set $n\mathbb{Z} + k$ containing all integers that has the remainder as k divided by n, is a coset of $n\mathbb{Z}$. There are n distinct cosets like this, and they form the quotient space $\mathbb{Z}/n\mathbb{Z}$.

2.3.2 Group homomorphism, group actions and group equivariance

Given two groups $(G, \cdot_G)$ and $(H, \cdot_H)$, a *group homomorphism* from G to H is function $f : G \to H$ such that for any $u, v \in G$

$$f(u \cdot_G v) = f(u) \cdot_H f(v).$$

It is a special mapping between two groups that is compatible with group structures. If f is an one-to-one mapping, we call it a *group isomorphism*. Two groups G_1 and G_2 are isomporphic if there is an isomorphism between them, and this is written as $G_1 \cong G_2$.

A *group action* is a group homomorphism from a given group G to the group of transformations on a space $\mathcal{X}$. We say the group G acts on the space $\mathcal{X}$ and the transformation corresponding to $g \in G$ is a bijection on $\mathcal{X}$ that maps $\mathbf{x}$ to $g \cdot \mathbf{x}$.

If the group actions of G on spaces $\mathcal{X}$ and $\mathcal{Y}$ are both defined, a function $f : \mathcal{X} \to \mathcal{Y}$ is said to be *group equivariant* if

$$g \cdot f(\mathbf{x}) = f(g \cdot \mathbf{x})$$

The function f is said to be *group invariant* if

$$f(\mathbf{x}) = f(g \cdot \mathbf{x})$$

It is a special case of group equivariance where the group action on $\mathcal{Y}$ is trivial.

2.3.3 Homogeneous spaces and lifting feature maps

If the action of a group G on the space $\mathcal{X}$ is defined, and the action is transitive (i.e. $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \exists g \in G$, s.t. $\mathbf{x}' = g \cdot \mathbf{x}$), we refer to $\mathcal{X}$ as being a homogeneous space for G. There is a natural one-to-one correspondence between the homogeneous space $\mathcal{X}$ and disjoint subsets of the group G. Given an arbitrary origin $\mathbf{x}_0 \in \mathcal{X}$, $H = \{g \in G | g \cdot \mathbf{x}_0 = \mathbf{x}_0\}$ is a subgroup of G, where H is called the stabiliser of the origin. Because the group action on $\mathcal{X}$ is transitive, every element $x \in \mathcal{X}$ corresponds to a left coset in $s(\mathbf{x}; \mathbf{x}_0)H \in G/H$, where $s(\mathbf{x}; \mathbf{x}_0)$ is (any) group element that transforms $\mathbf{x}_0$ to $\mathbf{x}$. It can be shown that for $\mathbf{x}, \mathbf{x}' \in \mathcal{X}, \mathbf{x} \neq \mathbf{x}'$, $s(\mathbf{x}; \mathbf{x}_0)$ and $s(\mathbf{x}'; \mathbf{x}_0)$ are disjoint.

Because spatial data is often represented as functions on the homogeneous space $f_{\mathcal{X}} : x_i \mapsto f_i$, while lifting-based group equivariant neural networks operate on feature maps defined on the group, there is usually an operation called *lifting*, that maps the data to the feature space of functions on the group, before applying equivariant modules. Using the correspondence between $\mathcal{X}$ and the quotient space G/H, we can map each pair (x_i, f_i) to the set $\{(g, f_i)|g \in s(x_i; x_0)H\}$. It can be seen as lifting the input feature map $f_{\mathcal{X}} : x_i \mapsto f_i$ to the feature map $\text{LIFT}(f_{\mathcal{X}}) : g \mapsto f_i$ for $g \in s(x_i; x_0)H$. In this work, we assume all input feature maps have been lifted to feature maps on the group.

2.3.4 Feature maps in G-CNNs

A general mathematical framework is introduced in Cohen et al. [2019] to specify convolutional feature spaces used in G-CNNs, and feature maps are treated as fields over a homogeneous space. It covers most previous works on equivariant neural networks including Cohen and Welling [2016, 2017], Cohen et al. [2018b], Weiler and Cesa [2019b]. Under this framework, one way to represent fields is through constrained functions defined on the whole symmetry group, also known as Mackey functions [Cohen et al., 2019].

Formally, let G be a symmetry group, and $H \leq G$ together with G determines the homogeneous space G/H. For a group representation (ρ, V) of H, the action of the whole group G on fields can be described by an induced representation $\pi = \text{Ind}_H^G \rho$, whose realisation depends on how we represent these fields. Below we specify the feature space $\mathcal{I}_M$ [1] for the Mackey function field representation discussed in [Cohen et al., 2018c, 2019]:

$$\mathcal{I}_M = \{f : G \to V | f(gh) = \rho(h^{-1}) f(g), \forall g \in G, h \in H\} \tag{2.17}$$

which forms a vector space. Moreover, when ρ is a *regular representation*, which is the implicit choice of Gens and Domingos [2014], Kanazawa et al. [2014], Dieleman et al. [2015, 2016], Cohen and Welling [2016], Marcos et al. [2016], fields can also be represented as unconstrained functions on G and the feature space can be written as

$$\mathcal{I}_G = \{f : G \to V'\}$$

[1] $\mathcal{I}_M$ corresponds to $\mathcal{I}_G$ in Cohen et al. [2019]

with V' being a different vector space from V. If ρ is a regular representation.

Feature maps are represented as functions on G in both $\mathcal{I}_M$ and $\mathcal{I}_G$, even though $\mathcal{I}_M$ have additional conditions given in Equation (2.17). Moreover, the induced representation $\pi = \text{Ind}_H^G \rho$ for them have the same form:

$$[\pi(u)f](g) = f(u^{-1}g)$$

2.3.5 Group equivariant neural networks

Group equivariant neural networks parameterise a family of functions that have guaranteed equivariance to specific symmetry groups. A well-known example of such networks is CNNs [Lecun et al., 1998]. CNNs consist of alternating convolutional layers and nonlinear layers such as pooling layers or activation functions, where each convolutional layer is equivariant to translations. This means if the inputs undergo a certain shift, the output feature maps will also shift correspondingly. In CNNs, feature maps can be represented as functions evaluated on a grid, written as $\mathbf{x}(u)$, where u denotes the integer grid coordinates. The convolutional layers can be represented by:

$$[\boldsymbol{\phi} * \mathbf{x}]_k(u) = \sum_{v \in \mathbb{Z}^2} \boldsymbol{\phi}_k(v) \cdot \mathbf{x}(u - v). \tag{2.18}$$

where k indexes the output feature map channel and $\boldsymbol{\phi}_k : \mathbb{Z}^2 \to \mathcal{Y}$ is the k-th convolutional fitler. The convolutional layers have guaranteed translational equivariance. Let us denote the translation operator as T_v, we have:

$$\begin{aligned} [T_v(\mathbf{x})](u) &:= \mathbf{x}(u - v) \\ \boldsymbol{\phi} * (T_v(\mathbf{x})) &= T_v(\boldsymbol{\phi} * \mathbf{x}),\ \forall v \in \mathbb{Z}^2 \end{aligned} \tag{2.19}$$

Compared to plain feed-forward networks, CNNs are more sample-efficient and generalize better.

Group Equivariant Convolutinal Networks (G-CNNs) [Cohen and Welling, 2016] extend this idea to symmetry groups beyond translations, e.g. rotations and reflections. In G-CNNs, feature maps can be seen as functions on groups $\mathbf{x} : G \to \mathbb{R}^{d_\mathbf{x}}$. We use group action to describe the transformations of feature maps, $g \cdot \mathbf{x} := T_g(\mathbf{x})$. In this work, we consider group action with the following form: $[g \cdot \mathbf{x}](u) = \mathbf{x}(g^{-1} \cdot u)$ (It is based on regular representations of groups). The group convolution is then defined as:

$$[\boldsymbol{\phi} * \mathbf{x}](u) = \sum_{g \in G} \boldsymbol{\phi}(g)\mathbf{x}(g^{-1} \cdot u) \tag{2.20}$$

One can prove that the convolution operation is equivariant to any group action, i.e.

$$\boldsymbol{\phi} * (g \cdot \mathbf{x}) = g \cdot (\boldsymbol{\phi} * \mathbf{x}). \tag{2.21}$$

These group convolutional layers can then be combined with activation functions or pooling layers to construct G-CNNs, which enjoys even better sample-efficiency and generalization compared to plain CNNs.

Permutation symmetry A particular symmetry, not discussed in G-CNNs but important to this thesis, is the permutation symmetry of a sequence. As an example, to predict the attributes of a set, the predictive model we use should be invariant to the order of set elements. For a sequence-to-sequence mapping $\Phi : \mathcal{X}^n \to \mathcal{Y}^n$, permutation equivariance means Φ satisfies the condition

$$\pi \cdot \Phi([\mathbf{x}_1, \ldots, \mathbf{x}_n]) = \Phi(\pi \cdot [\mathbf{x}_1, \ldots, \mathbf{x}_n]) := \Phi(\mathbf{x}_{\pi(1)}, \ldots, \mathbf{x}_{\pi(n)}) \tag{2.22}$$

for any permutation π.

Deep sets [Zaheer et al., 2017] and set transformers [Lee et al., 2019a] are two popular deep learning architectures that incorporate permutation symmetries. Deep sets parameterise the sequence mapping by stacking equivariant layers as follows:

$$\Phi_{\text{DS}}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \boldsymbol{\sigma}(\lambda \boldsymbol{I}[\mathbf{x}_1, \ldots, \mathbf{x}_n] + \gamma \text{pool}([\mathbf{x}_1, \ldots, \mathbf{x}_n])\mathbf{1}). \tag{2.23}$$

The computational cost of deep sets scales linearly with the size of the set, marking them as highly efficient. However, their ability to capture interactions between set elements is somewhat limited due to reliance solely on the pooling operation, which can make them less expressive in real-world scenarios. Set transformers address this issue by using self-attention blocks without positional encoding to model the interaction. The attention module is provably permutation equivariant and is written as:

$$\Phi_{\text{ATTN}}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \text{MultiHead}(K = \boldsymbol{X}, Q = \boldsymbol{X}, V = \boldsymbol{X}) \tag{2.24}$$

where $\boldsymbol{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and $\boldsymbol{X}$ are used as keys, values and queries. These attention modules can be combined with feed-forward networks and layer normalization [Ba et al., 2016] to construct set transformers.

Chapter 3

MetaFun: Meta-Learning with Iterative Functional Updates

3.1 Introduction

The goal of meta-learning is to be able to generalise to new tasks from the same task distribution as the training tasks. In supervised meta-learning, a task can be described as making predictions on a set of unlabelled data points (*target*) by effectively learning from a set of data points with labels (*context*). Various ideas have been proposed to tackle supervised meta-learning from different perspectives [Andrychowicz et al., 2016, Ravi and Larochelle, 2016, Finn et al., 2017a, Koch, 2015, Snell et al., 2017, Vinyals et al., 2016, Santoro et al., 2016, Rusu et al., 2019]. In this work, we are particularly interested in a family of meta-learning models that use an encoder-decoder pipeline, such as Neural Processes [Garnelo et al., 2018a,b]. The encoder is a permutation-invariant function on the context that summarises the context into a task representation, while the decoder produces a predictive model for the targets, conditioned on the task representation. The objective of meta-learning is then to learn the encoder and the decoder such that the produced predictive model generalises well to the targets of new tasks.

Previous works in this category such as the Conditional Neural Process (CNP) and the Neural Process (NP) [Garnelo et al., 2018a,b] use sum-pooling operations to produce finite-dimensional, vectorial, task representations. In this work, we investigate the idea of summarising tasks with infinite-dimensional functional representations. Although there is a theoretical guarantee that sum-pooling of instance-wise representations can

express any set function (*universality*) [Zaheer et al., 2017, Bloem-Reddy and Teh, 2020], in practice CNP and NP tend to underfit the context [Kim et al., 2019]. This observation is in line with the theoretical finding that for universality, the dimension of the task representation should be at least as large as the cardinality of the context set, if the encoder is a smooth function Wagstaff et al. [2019]. We develop a method that explicitly uses functional representations. Here the effective dimensionality of the task representation grows with the number of context points, which addresses the aforementioned issues of fixed-dimensional representations. Moreover, in practice it is difficult to model interactions between data points with only sum-pooling operations. This issue can be partially addressed by inserting modules such as relation networks [Sung et al., 2018, Rusu et al., 2019] or set transformers [Lee et al., 2019a] before sum-pooling. However, only within-context but not context-target interactions can be modelled by these modules. The construction of our functional representation involves measuring similarities between all data points, which naturally contains information regarding interactions between elements in either the context or the target.

Furthermore, rather than producing the functional representation in a single pass, we develop an approach that *learns* iterative updates to encode the context into the task representation. In general, learning via iterative updates is often easier than directly learning the final representation, because of the error-correcting opportunity at each iteration. For example, an iterative parameterisation of the encoder in Variational Autoencoders (VAEs) has been demonstrated to be effective in reducing the amortisation gap [Marino et al., 2018], while in meta-learning, both learning to learn methods [Andrychowicz et al., 2016, Ravi and Larochelle, 2016] and Model Agnostic Meta Learning (MAML) [Finn et al., 2017a] use iterative updating procedures to adapt to new tasks, although these update rules operate in parameter space rather than function space. Therefore, it is reasonable to conjecture that iterative structures are favourable inductive biases for the task encoding process.

In summary, the primary contribution of this work is a meta-learning approach that learns to summarise a task using a functional representation constructed via iterative updates. We apply our approach to solve meta-learning problems on both regression and classification tasks, and demonstrate competitive results on heavily benchmarked datasets such as miniImageNet [Vinyals et al., 2016] and tieredImageNet [Ren et al., 2018], which has not been shown with meta-learning methods based on CNP [Garnelo et al., 2018a] and NP [Garnelo et al., 2018b]. We also conducted an ablation study to understand the effects of the different model components.

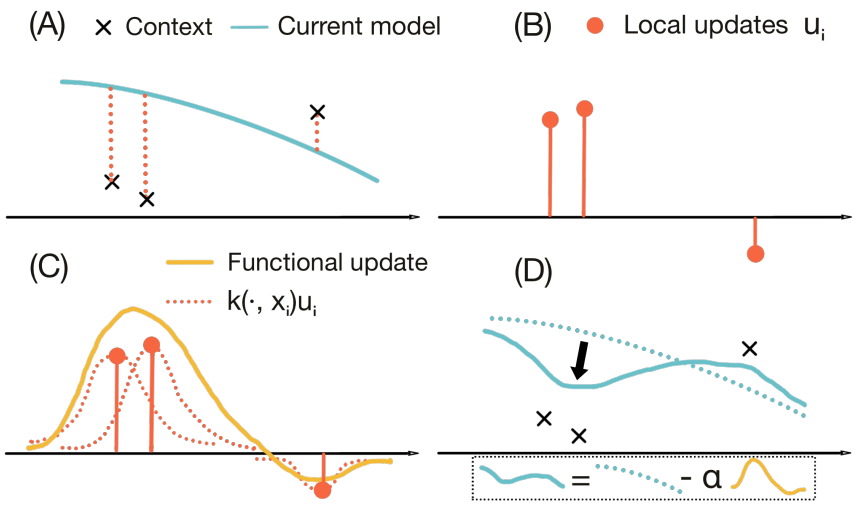


Figure 3.1: To illustrate the iterative procedure in MetaFun, we consider a simpler case where our functional representation is just a predictor for the task. (A) The figure depicts a 1D regression task with the current predictor. (B) Local updates are computed by evaluating the functional representation (the current predictor) on the context inputs, and comparing it to the corresponding context outputs. Here we simply measure differences between evaluations (predictions) and outputs. (C) We apply functional pooling to aggregate local updates into a global functional update, which generalises the local updates to the whole input domain. (D) The functional update is applied to the current functional representation with a learning rate α.

3.2 MetaFun

Meta-learning, or learning to learn, leverages past experiences to quickly adapt to tasks $\mathcal{T} \sim p(\mathcal{T})$ drawn iid from some task distribution. In supervised meta-learning, a task $\mathcal{T}$ takes the form of $\mathcal{T} = \{\ell, \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}, \{(\mathbf{x}'_j, \mathbf{y}'_j)\}_{j\in\mathbf{T}}\}$, where $\mathbf{x}_i, \mathbf{x}'_j \in \mathcal{X}$ are inputs, $\mathbf{y}_i, \mathbf{y}'_j \in \mathcal{Y}$ outputs, ℓ is the loss function to be minimised, $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}$ is the context, and $\{(\mathbf{x}'_j, \mathbf{y}'_j)\}_{j\in\mathbf{T}}$ is the target. We consider the process of learning as constructing a predictive model using the task context and refer to the mapping from context $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}$ to a predictive model $f = \Phi(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}; \phi)$ as the *learning model* parameterised by ϕ. In our formulation, the objective of meta-learning is to optimise the learning model such that the expected loss on the target under f is minimised, formally written as:

$$
\begin{aligned}
f &= \Phi(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}; \phi) \\
\phi^* &= \arg\min_{\phi} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\frac{1}{|\mathbf{T}|} \sum_{j\in\mathbf{T}} \ell(f(\mathbf{x}'_j), \mathbf{y}'_j) \right],
\end{aligned} \tag{3.1}
$$

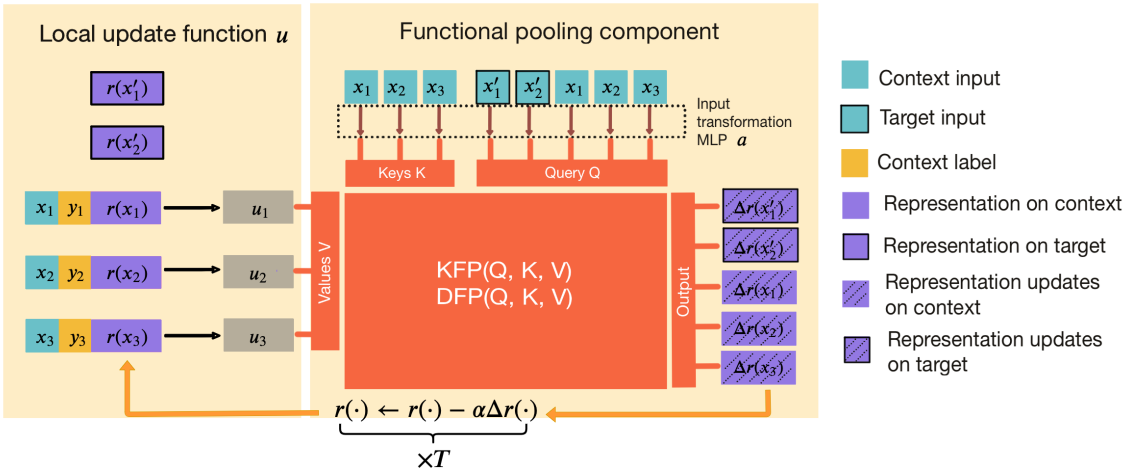


Figure 3.2: This figure illustrates the iterative computation of functional representation in MetaFun. At each iteration, we first evaluate the current functional representation at both context and target points. Then the shared local update function u takes in each context point and the corresponding evaluation as inputs, and produces local update u_i. Next, we apply (kernel-based or attention-based) functional pooling to aggregate local updates u_i into a functional update $\Delta r(\cdot)$, which for each query is a linear combination of local updates u_i weighted by similarities between this query and all keys. Finally, the functional updates are evaluated for both the context and the target, and are applied to the corresponding evaluations of functional representation with a learning rate α.

where both $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in \mathbf{C}}$, $\{(\mathbf{x}'_j, \mathbf{y}'_j)\}_{i \in \mathbf{T}}$ come from task $\mathcal{T}$.

3.2.1 Learning functional task representation

Like previous works such as CNP and NP, we construct the learning model using an encoder-decoder pipeline, where the encoder $\Phi_e(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in \mathbf{C}}; \phi_e)$ is a permutation-invariant function of the context producing a task representation. In past works, pooling operations are usually used to enforce permutation-invariance. CNP and NP use sum-pooling: $\mathbf{r} = \sum_{i \in \mathbf{C}} \mathbf{r}_i$, where $\mathbf{r}_i = h(\mathbf{x}_i, \mathbf{y}_i; \phi_e)$ is a representation for context pair $\mathbf{x}_i, \mathbf{y}_i$, and $\mathbf{r}$ is a fixed-dimensional task representation. Instead, we introduce functional-pooling operations, which also enforce permutation-invariance but output a function that can loosely be interpreted as an infinite-dimensional representation.

Definition 3.2.1 (Functional pooling)**.** Let $k(\cdot, \cdot)$ be a real-valued similarity measure, and $\{(\mathbf{x}_i, \mathbf{r}_i)\}_{i \in \mathbf{C}}$ be a set of key-value pairs with $\mathbf{x}_i \in \mathcal{X}$, $\mathbf{r}_i \in \mathcal{R}$. *Functional pooling*

is a mapping $\textsc{FunPooling} : (\mathcal{X} \times \mathcal{R})^{|\mathbf{C}|} \to \mathcal{H}$ defined as

$$r(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{r}_i)\}_{i \in \mathbf{C}}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{r}_i \,, \tag{3.2}$$

where the output is a function $r : \mathcal{X} \to \mathcal{R}$ and $\mathcal{H}$ is a space of such functions.

In practice, we only need to evaluate this function on a finite query set $\{(\mathbf{x}'_j, \mathbf{y}'_j)\}_{j \in \mathcal{Q}}$ (consisting of both contexts and targets; see below). That is, we only need to compute $R = [r(\mathbf{x}'_1), \ldots, r(\mathbf{x}'_{|\mathcal{Q}|})]^\top$, which can be easily implemented using matrix operations. We consider two types of $\textsc{FunPooling}$ here, though others are possible. The kernel-based $\textsc{FunPooling}$ reads as,

$$R = \textsc{kfp}\,(Q, K, V) := k_{\text{rbf}}(Q, K)V \,, \tag{3.3}$$

where k_{rbf} is the RBF kernel, $Q = [a(\mathbf{x}'_1), \ldots, a(\mathbf{x}'_{|\mathcal{Q}|})]^\top$ is a matrix whose rows are queries, $a(\cdot)$ is a transformation mapping inputs into features, $K = [a(\mathbf{x}_1), \ldots, a(\mathbf{x}_{|C|})]^\top$ a matrix whose rows are keys, and $V = [\mathbf{r}_1, \ldots, \mathbf{r}_{|C|}]^\top$ a matrix whose rows are values (using terminology from the attention literature). Parameterising input transformation a with a deep neural network can be seen as using deep kernels [Wilson et al., 2016] as the similarity measure. The second type of $\textsc{FunPooling}$ is given by dot-product attention,

$$R = \textsc{dfp}\,(Q, K, V) := \text{softmax}(QK^\top/\sqrt{d_k})V \,, \tag{3.4}$$

where d_k is the dimension of the query/key vectors.

Our second core idea is that rather than producing the task representation in a single pass (like previous encoder-decoder meta-learning approaches), we start from an initial representation $r^{(0)}(\cdot)$, and iteratively produce improved representations $r^{(1)}(\cdot), \ldots, r^{(T)}(\cdot)$. At each step, a parameterised local update function u produces local updates $\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$ for each $i \in \mathbf{C}$. These can then be aggregated into a global update to the task representation using functional pooling. Finally, the global functional update $\Delta r^{(t)}(\cdot)$ is applied to $r^{(t)}(\cdot)$ with a step size α:

$$\begin{aligned} \mathbf{u}_i &= u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i)) \,, \\ \Delta r^{(t)}(\cdot) &= \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in \mathbf{C}}) \,, \\ r^{(t+1)}(\cdot) &= r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot) \,, \end{aligned} \tag{3.5}$$

Typically, the local update function u and the functional pooling operation can be parameterised by multi-layer perceptron (MLP) and deep kernels/attentions respectively (as we will discuss next). Equation (3.5) then defines a neural update rule operating directly in function space. The functional update $\Delta r^{(t)}(\cdot)$ depends on the current functional representation $r^{(t)}(\cdot)$ and the context $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}$. Figure 3.1 illustrates our iterative procedure in a simplified setting.

The final task representation can then be decoded into a predictor $f(\cdot) = \Phi_d(r^{(T)}(\cdot); \phi_d)$. The specific parametric forms of the decoder take different forms for regression and classification, and are described in Section 3.2.2. The decoder requires the evaluation of functional representation $r^{(T)}(\mathbf{x})$ at $\mathbf{x}$ only for predicting $f(\mathbf{x})$. Therefore, it is unnecessary to compute the functional representations $r(\cdot)$ (including their functional updates) on all input points. Instead, we compute them only on the context $\{\mathbf{x}_i\}_{i\in\mathbf{C}}$ and target inputs $\{\mathbf{x}'_j\}_{j\in\mathbf{T}}$. We use $\mathbf{r}^{(t)} = [r^{(t)}(\mathbf{x}_1) \dots r^{(t)}(\mathbf{x}_{|\mathbf{C}|}), r^{(t)}(\mathbf{x}'_1) \dots r^{(t)}(\mathbf{x}'_{|\mathbf{T}|})]^\top$ to denote a matrix where each row is $r^{(t)}(\mathbf{x})$ evaluated on either context or target inputs, and let $Q = [a(\mathbf{x}_1) \dots a(\mathbf{x}_{|\mathbf{C}|}), a(\mathbf{x}'_1) \dots a(\mathbf{x}'_{|\mathbf{T}|})]^\top$. Equation (3.5) can be implemented using matrix computations as follows,

$$\mathbf{u}_i^{(t)} = u(\mathbf{x}_i, \mathbf{y}_i, \mathbf{r}_i^{(t)}), \tag{3.6}$$

$$U^{(t)} = [\mathbf{u}_1^{(t)}, \dots, \mathbf{u}_{|\mathbf{C}|}^{(t)}]^\top, \tag{3.7}$$

$$\Delta\mathbf{r}^{(t)} = \text{KFP or DFP}\left(Q, K, U^{(t)}\right), \tag{3.8}$$

$$\mathbf{r}^{(t+1)} = \mathbf{r}^{(t)} - \alpha\Delta\mathbf{r}^{(t)} \tag{3.9}$$

where $\mathbf{r}_i^{(t)}$ denotes the i-th row of $\mathbf{r}^{(t)}$.

To obtain a prediction $\mathbf{f}_j$ for the target $(\mathbf{x}'_j, \mathbf{y}'_j)$, we decode the final representation for this target point: $\mathbf{f}_j = \Phi_d(\mathbf{r}^{(T)}_{|\mathbf{C}|+j}; \phi_d)$, and the overall training loss can be written as:

$$L(u, a, \phi_d) = \frac{1}{|\mathbf{T}|}\sum_{j\in\mathbf{T}} \ell(\mathbf{f}_j, \mathbf{y}'_j), \tag{3.10}$$

where the predictions $\mathbf{f}_j$ depend on u, a and ϕ_d.

Assuming the width and depth of all our neural network components are bounded by W and D respectively, and the output dimension of u is also less than W, the time complexity of our approach is $\mathcal{O}\big(W|\mathbf{C}|(|\mathbf{C}| + |\mathbf{T}|) + W^2D(|\mathbf{C}|T + |\mathbf{T}|)\big)$, and the space complexity is $\mathcal{O}\big((|\mathbf{C}| + WT)(|\mathbf{C}| + |\mathbf{T}|) + W^2D\big)$. For few-shot problems, $|\mathbf{C}|$ and $|\mathbf{T}|$ are typically small, and $T \leq 6$ in all our experiments.

3.2.2 MetaFun for regression and classification

While the proposed framework can be applied to any supervised learning task, the specific parameterisation of learnable components can affect the model performance. In this section, we specify the parametric forms of our model that work well on regression and classification tasks.

Regression For regression tasks, we parameterise the local update function $u(\cdot)$ using a multi-layer perceptron as $u([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = \text{MLP}([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)])$, $i \in C$, where $[\cdot]$ is concatenation. We also use an MLP to parametrise the input transformation $a(\cdot)$ in the functional pooling. The decoder in this case is given by $\mathbf{w} = \text{MLP}(r(\mathbf{x}))$, another MLP[1] that outputs $\mathbf{w}$, which then parameterises the predictive model $f = \text{MLP}(\mathbf{x}; \mathbf{w})$.

Note that our model can easily be modified to incorporate Gaussian uncertainty by adding an extra output vector for the predictive standard deviation: $P(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mu_{\mathbf{w}}(\mathbf{x}), \sigma_{\mathbf{w}}(\mathbf{x})), \mathbf{w} = \text{MLP}(r(\mathbf{x}))$. For further architecture details, see Appendix.

Classification For K-way classification, we divide the latent functional representation $r(\mathbf{x})$ into K parts $[r^1(\mathbf{x}), \ldots, r^K(\mathbf{x})]$, where $r^k(\mathbf{x})$ corresponds to the class k. Consequently, the local update function $u(\cdot)$ also has K parts, that is, $u([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = [u^1(\cdot), \ldots, u^K(\cdot)]$. In this case, $\mathbf{y}_i = [y_i^1, \ldots, y_i^K]$ is the class label expressed as a one-hot vector; the u^k is defined as follows,

$$\begin{aligned} u^k([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = {} & y_i^k u_+(m(r^k(\mathbf{x}_i)), \boldsymbol{m}_i) \\ & + (1 - y_i^k) u_-(m(r^k(\mathbf{x}_i)), \boldsymbol{m}_i)\,, \end{aligned} \tag{3.11}$$

where $\boldsymbol{m}_i = \sum_{k=1}^{K} m(r^k(\mathbf{x}_i))$ summarises representations of all classes, and m, u_+, u_- are parameterised by separate MLPs. With this formulation, we update the class representations using either u_+ (when the label matches k) or u_- (when the label is different to k), so that labels are not concatenated to the inputs, but directly used to activate different model components, which is crucial for model performance. Furthermore, interactions between data points in classification problems include both within-class and between-class interactions. Our approach is able to integrate two types of interactions by having separate functional representation for each class and

[1] It might be desirable to use other parameterisations of the input transformation $a(\cdot)$, and the decoder $f(\cdot)$, e.g., $f(\mathbf{x}) = \text{MLP}([\mathbf{x}, r(\mathbf{x})])$, or feeding $r(\mathbf{x})$ to each layer of the MLP.

computing local updates for each class differently based on class membership of each data point. In fact, this formulation resembles the structure of the local update rule in functional gradient descent for classification tasks, which is a special case of our approach (see Section 5.4). Same as in regression tasks, the input transformation $a(\cdot)$ in the functional pooling is still an MLP. The parametric form of the decoder is the same as in Latent Embedding Optimisation (LEO) [Rusu et al., 2019]. The class representation $r^k(\mathbf{x})$ generates weights $\mathbf{w^k} \sim \mathcal{N}(\mu(r^k(\mathbf{x})), \sigma(r^k(\mathbf{x})))$ where μ and σ are MLPs or just linear functions, and the final prediction is given by

$$P(\mathbf{y} = k|\mathbf{x}) = \text{softmax}(\mathbf{x}^T\mathbf{w})_k \,, \tag{3.12}$$

where $\mathbf{w} = [\mathbf{w^1}, \ldots, \mathbf{w^K}]$, $k = 1, \ldots, K$. Hyperparameters of all components are described in Appendix.

3.3 Related work

Functional Gradient Descent Functional gradient descent [Mason et al., 1999, Guo et al., 2001] is an optimisation algorithm used to minimise the objective function by moving in the direction of the negative gradient in function space. To ensure smoothness, we may work with functions in a Reproducing kernel Hilbert space (RKHS) [Aronszajn, 1950, Berlinet and Thomas-Agnan, 2011] defined by a kernel $k(\mathbf{x}, \mathbf{x}')$. Given a function f in the RKHS, we are interested in minimising the supervised loss $L(f) = \sum_{i \in C} \ell(f(\mathbf{x}_i), \mathbf{y}_i)$ with respect to f. We can do so by computing the functional derivative and use it to iteratively update f (see Appendix for more details),

$$f^{(t+1)}(\mathbf{x}) = f^{(t)}(\mathbf{x}) - \alpha \sum_{i \in C} k(\mathbf{x}, \mathbf{x}_i) \nabla \ell(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i) \tag{3.13}$$

with step size α, and $\nabla \ell(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i)$ denotes gradient w.r.t. to predictions in the loss function ℓ.

The update rule in Equation (3.5) becomes that of functional gradient descent in Equation (3.13) when

(i) A trivial decoder $f(\mathbf{x}) = \Phi_d(r(\mathbf{x}); \phi_d)(\mathbf{x}) = r(\mathbf{x})$ is used, so the functional representation $r(\mathbf{x})$ is the same as the predictive model $f(\mathbf{x})$.

(ii) Kernel functional pooling KFP is used and the kernel function is fixed.

(iii) Using gradient-based local update function $u(\mathbf{x}, \mathbf{y}, f(\mathbf{x})) = \nabla \ell(f(\mathbf{x}), \mathbf{y})$.

Furthermore, for a K-way classification problem, we predict K-dimensional logits $f(\mathbf{x}) = [f^1(\mathbf{x}), \ldots, f^K(\mathbf{x})]^\top$, and use cross entropy loss as follows:

$$\ell(f(\mathbf{x}), \mathbf{y}) = -\sum_{k=1}^{K} y^k \log \frac{e^{f^k(\mathbf{x})}}{\sum_{k'=1}^{K} e^{f^{k'}(\mathbf{x})}}\,, \tag{3.14}$$

where $\mathbf{y} = [y^1, \ldots, y^K]^\top$ is the one-hot label for $\mathbf{x}$.

The gradient-based local update function is now $\nabla \ell(f(\mathbf{x}), \mathbf{y}) = [\partial_1 \ell(f(\mathbf{x}), \mathbf{y}), \ldots, \partial_K \ell(f(\mathbf{x}), \mathbf{y})]^\top$ where $\partial_k \ell(f(\mathbf{x}), \mathbf{y})$ is partial derivative w.r.t. each predictive logit:

$$\partial_k \ell(f(\mathbf{x}), \mathbf{y}) = \frac{e^{f^k(\mathbf{x})}}{\sum_{k'=1}^{K} e^{f^{k'}(\mathbf{x})}} - y^k\,. \tag{3.15}$$

Here $\partial_k \ell(f(\mathbf{x}), \mathbf{y})$ is analogous to $u^k(\cdot)$ in Equation (3.11), which is the local update function for class k.

If m, u_+, u_- in Equation (3.11) are specified rather than being learned, more specifically:

$$\begin{aligned} m(f^{k'}(\mathbf{x})) &= e^{f^{k'}(\mathbf{x})} \\ u_+(m(f^k(\mathbf{x})), \boldsymbol{m}) &= \frac{m(f^k(\mathbf{x}))}{\boldsymbol{m}} - 1 \\ u_-(m(f^k(\mathbf{x})), \boldsymbol{m}) &= \frac{m(f^k(\mathbf{x}))}{\boldsymbol{m}} \\ \boldsymbol{m} &= \sum_{k=1}^{K} m(f^{k'}(\mathbf{x}))\,, \end{aligned} \tag{3.16}$$

Equation (3.15) can be rewritten as:

$$\begin{aligned} \partial_k \ell(f(\mathbf{x}), \mathbf{y}) = y^k u_+(m(f^k(\mathbf{x})), \boldsymbol{m}) \\ + (1 - y^k) u_-(m(f^k(\mathbf{x})), \boldsymbol{m})\,, \end{aligned} \tag{3.17}$$

which has a similar form as Equation (3.11).

Therefore, our approach can be seen as an extension of functional gradient descent, with an additional learning capacity due learnable neural modules which afford more flexibility. From this perspective, our approach tackles supervised meta-learning problems by learning an optimiser in function space.

Supervised Meta-Learning Various ideas have been proposed to solve the problem of supervised meta-learning. Andrychowicz et al. [2016], Ravi and Larochelle [2016] learn the neural optimisers from previous tasks which can be used to optimise models for new tasks. However, these learned optimisers operate in parameter space rather than function space as we do. MAML [Finn et al., 2017a] learns the initialisation from which models are further adapted for a new task by a few gradient descent steps. Koch [2015], Snell et al. [2017], Vinyals et al. [2016] explore the idea of learning a metric space from previous tasks in which data points are compared to each other to make predictions at test time. Santoro et al. [2016] demonstrate that Memory-Augmented Neural Networks (MANN) can rapidly integrate the data for a new task into memory, and utilise this stored information to make predictions. Recently, Dhillon et al. [2019] show that well-designed fine-tuning can provide strong baselines to meta-learning methods, highlighting the limitations of current benchmarks.

Our approach, in line with previous works such as CNP and NP, adopt an encoder-decoder pipeline to tackle supervised meta-learning. The encoder in CNP corresponds to a summation of instance-level representations produced by a shared instance encoder. NPs, on the other hand, use a probabilistic encoder with the same parametric form as CNP, but producing a distribution of stochastic representation. The Attentive Neural Process (ANP) [Kim et al., 2019] adds a deterministic path in addition to the stochastic path in NP. The deterministic path produces a target-specific representation, which can be interpreted as applying functional pooling (implemented with multihead attention [Vaswani et al., 2017]) to instance-wise representation. However, the representation is directly produced in a single pass rather than iteratively improved as we do, and only regression applications are explored as opposed to few-shot image classification. In fact, to achieve high performance for classification tasks, it is crucial for CNP to only apply sum-pooling within each class [Garnelo et al., 2018a], and it is unclear how to follow similar practices in ANP with both within-class and between-class interactions still being modelled. Recently, Gordon et al. [2019] have also extended CNP to use functional representations, but for the purpose of incorporating translation equivariance in the inputs as an inductive bias rather than increasing representational capacity as we do. Their approach uses convnets to impose translation equivariance and does not learn a flexible iterative encoder.

Pooling operations are usually used in encoder-decoder meta-learning to enforce permutation invariance in the encoder. As an example, encoders in both CNP and NP use simple *sum-pooling* operations. More expressive pooling operations have been

proposed to model interactions between data points. Murphy et al. [2019] introduces *Janossy pooling* which applies permutation-sensitive functions to all reorderings and averages the outputs, while Lee et al. [2019a] use *pooling by multihead attention* (PMA), which uses a finite query set to attend to the processed key-value pairs. Loosely speaking, attention-based functional pooling can be seen as having the whole input domain $\mathcal{X}$ as the query set in PMA.

Gradient-Based Meta-Learning Interestingly, many gradient-based meta-learning methods such as MAML can also be cast into an encoder-decoder formulation, because a gradient descent step is a valid permutation-invariant function. For a model $f(\cdot, \theta)$ parameterised by θ, one gradient descent step on the context loss has the following form,

$$\theta_{t+1} = \theta_t - \alpha \sum_{i \in C} \nabla_\theta \ell(f(\mathbf{x}_i; \theta_t), \mathbf{y}_i)\,, \tag{3.18}$$

where ℓ is the loss function, α is the learning rate, and θ_t are the model parameters after t gradient steps. This corresponds to a special case of permutation-invariant functions where we take the instance-wise encoder to be $h_t(\mathbf{x}_i, \mathbf{y}_i; \theta_t) = \theta_t/|\mathbf{C}| - \alpha \nabla_\theta \ell(f(\mathbf{x}_i; \theta_t), \mathbf{y}_i)$ and apply sum-pooling $\theta_{t+1} = \sum_i h_t(\mathbf{x}_i, \mathbf{y}_i; \theta_t)$. Multiple gradient-descent steps also result in a permutation-invariant function, which can be proved by induction. We refer to this as a gradient-based encoder. What follows is that popular meta-learning methods such as MAML can be seen as part of the encoder-decoder formulation. More specifically, in MAML, we learn an initialisation of the model parameters θ_0 from training tasks, and adapt to new tasks by running T gradient steps from the learned initialisation. Therefore, θ_T can be seen as the task representation (albeit very high-dimensional) produced by a gradient-based encoder. The success of MAML on a variety of tasks can be partially explained by the high-dimensional representation and the iterative adaptation by gradient descent, supporting our usage of a functional ('infinite-dimensional') representation and iterative updating procedure. Note, however, that the update rule in MAML operates in parameter space rather than function space as in our case.

Under the same encoder-decoder formulation, a comparison regarding MAML and MetaFun can be made, which partially explains why MetaFun can be desirable: Firstly, the updates in MAML must lie in its parametric space, while there is no parametric constraint in MetaFun, which is better illustrated in Figure 3.3. Secondly, MAML uses gradient-based updates, while MetaFun uses learned local updates, which

potentially contains more information than gradient. Finally, MAML does not explicitly consider interactions between data points, while both within-context and context-target interactions are modelled in MetaFun.

3.4 Experiments

We evaluate our proposed model on both few-shot regression and classification tasks. In all experiments that follow, we partition the data into training, validation and test meta-sets, each containing data from disjoint tasks. For quantitative results, we train each model with 5 different random seeds and report the mean and the standard deviation of the test accuracy. For further details on hyperparameter tuning, see the Appendix. All experiments are performed using TensorFlow [Abadi et al., 2015], and the code is available online [2].

3.4.1 1-D function regression

We first explore a 1D sinusoid regression task where we visualise the updating procedure in function space, providing intuition for the learned functional updates. Then we incorporate Gaussian uncertainty into the model, and compare our predictive uncertainty against that of a GP which generates the data.

Table 3.1: Few-shot regression on sinusoid. MAML can beneift from more parameters, but MetaFun still outperforms all MAMLs despite less parameters being used compared to large MAML. We report mean and standard deviation of 5 independent runs.

Model	5-shot MSE	10-shot MSE
Original MAML	0.390 ± 0.156	0.114 ± 0.010
Large MAML	0.208 ± 0.009	0.061 ± 0.004
Very Wide MAML	0.205 ± 0.013	0.059 ± 0.010
MetaFun	$\mathbf{0.040 \pm 0.008}$	$\mathbf{0.017 \pm 0.005}$

Visualisation of functional updates We train a T-step MetaFun with dot-product functional pooling, on a simple sinusoid regression task from Finn et al. [2017a], where

[2] A tensorflow implementation of our model is available at github.com/jinxu06/metafun-tensorflow

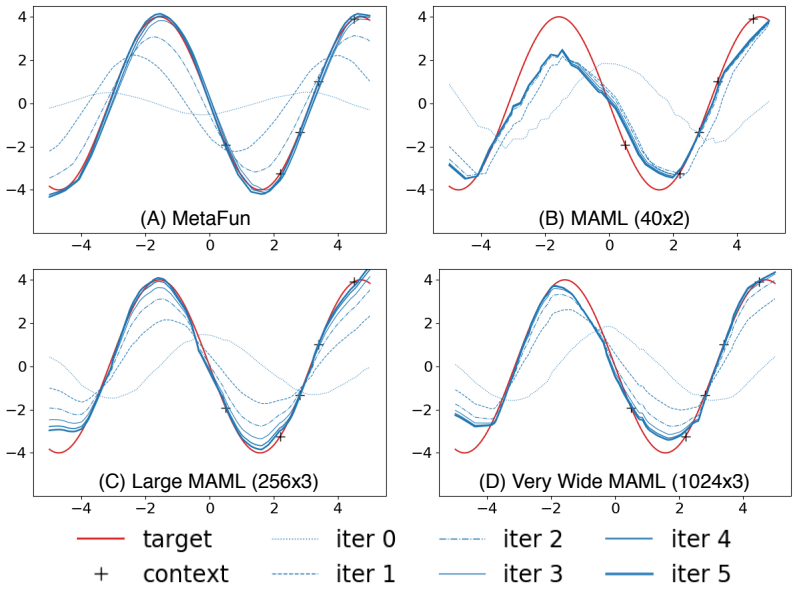


Figure 3.3: MetaFun is able to learn smooth updates, and recover the ground truth function almost perfectly. While the updates given by MAMLs are relatively not smooth, especially for MAML with less parameters.

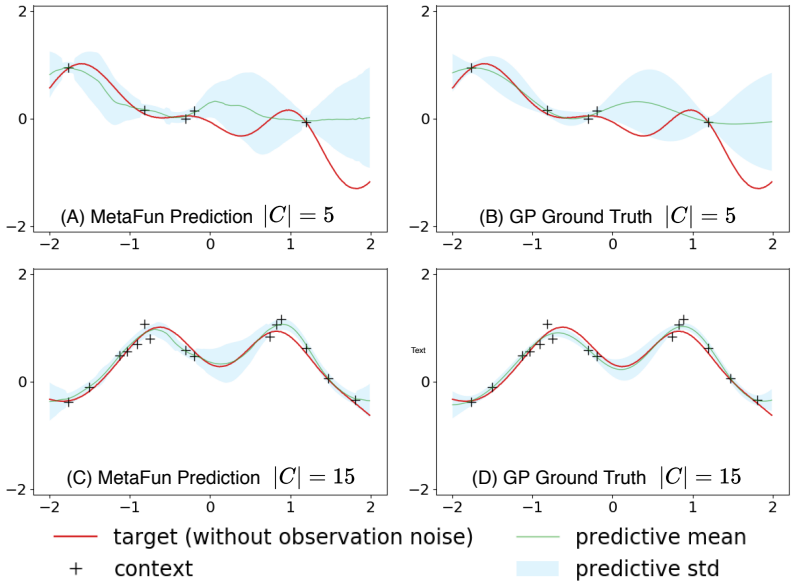


Figure 3.4: Predictive uncertainties for MetaFun matches those for the oracle Gaussian Process (GP) very closely in both 5-shot and 15-shot cases. The model is trained on random context size ranging from 1 to 20.

each task uses data points of a sine wave. The amplitude A and phase b of the sinusoid varies across tasks and are randomly sampled during training and test time, with $A \in \mathcal{U}(0.1, 5.0)$ and $b \in \mathcal{U}(0, \pi)$. The x-coordinates are uniformly sampled from $\mathcal{U}(-5.0, 5.0)$. Figure 3.3 shows that our proposed algorithm learns a smooth transition from the initial state to the final prediction at $t = T = 5$. Note that although only 5 context points on a single phase of the sinusoid are given at test time, the final iteration makes predictions close to the ground truth across the whole period. As a comparison, we use MAML as an example of updating in parameter space. The original MAML (40 units $\times$ 2 hidden layers) can fit the sinusoid quite well after several iterations from the learned initialisation. However the prediction is not as good, particularly on the left side where there are no context points (see Figure 3.3 B). As we increase the model size to large MAML (256 units $\times$ 3 hidden layers), updates become much smoother (Figure 3.3 C) and the predictions are closer to the ground truth. We further conduct experiments with a very wide MAML (1024 units $\times$ 3 hidden layers), but the performance cannot be further improved (Figure 3.3 D). In Table 3.1, we compare the mean squared error averaged across tasks. MetaFun performs much better than all MAMLs, even though less parameters (116611 parameters) are used compared to large MAML (132353 parameters).

Predictive uncertainties As another simple regression example, we demonstrate that MetaFun, like CNP, can produce good predictive uncertainties. We use synthetic data generated using a GP with an RBF kernel and Gaussian observation noise ($\mu = 0, \sigma = 0.1$), and our decoder produces both predictive means and variances. As in Kim et al. [2019], we found that MetaFun-DFP can produce somewhat piece-wise constant mean predictions which is less appealing in this situation. On the other hand, MetaFun-KFP (with deep kernels) performed much better, as can be seen in Figure 3.4. We consider the cases of 5 or 15 context points, and compare our predictions to those for the oracle GP. In both cases, our model gave very good predictions.

3.4.2 Classification: miniImageNet and tieredImageNet

The *miniImageNet* dataset [Vinyals et al., 2016] consists of 100 classes selected randomly from the ILSVRC-12 dataset [Russakovsky et al., 2015], and each class contains 600 randomly sampled images. We follow the split in Ravi and Larochelle [2016], where the dataset is divided into training (64 classes), validation (16 classes),

Table 3.2: Few-shot Classification Test Accuracy

Models	**miniImageNet 5-way 1-shot**	**miniImageNet 5-way 5-shot**
(Without deep residual networks feature extraction):		
Matching networks [Vinyals et al., 2016]	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
Meta-learner LSTM [Ravi and Larochelle, 2016]	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
MAML [Finn et al., 2017a]	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$
LLAMA [Grant et al., 2018]	$49.40 \pm 1.83\%$	-
REPTILE [Nichol et al., 2018]	$49.97 \pm 0.32\%$	$65.99 \pm 0.58\%$
PLATIPUS [Finn et al., 2018]	$50.13 \pm 1.86\%$	-
(Without data augmentation):		
Meta-SGD [Li et al., 2017]	$54.24 \pm 0.03\%$	$70.86 \pm 0.04\%$
SNAIL [Mishra et al., 2018]	$55.71 \pm 0.99\%$	$68.88 \pm 0.92\%$
Bauer et al. [2017]	$56.30 \pm 0.40\%$	$73.90 \pm 0.30\%$
Munkhdalai et al. [2018]	$57.10 \pm 0.70\%$	$70.04 \pm 0.63\%$
TADAM [Oreshkin et al., 2018]	$58.50 \pm 0.30\%$	$76.70 \pm 0.30\%$
Qiao et al. [2018]	$59.60 \pm 0.41\%$	$73.74 \pm 0.19\%$
LEO	$\mathbf{61.76 \pm 0.08}\%$	$77.59 \pm 0.12\%$
MetaFun-DFP	$\mathbf{62.12 \pm 0.30}\%$	$77.78 \pm 0.12\%$
MetaFun-KFP	$61.16 \pm 0.15\%$	$\mathbf{78.20 \pm 0.16}\%$
(With data augmentation):		
Qiao et al. [2018]	$\mathbf{63.62 \pm 0.58}\%$	$78.83 \pm 0.36\%$
LEO	$63.97 \pm 0.20\%$	$79.49 \pm 0.70\%$
MetaOptNet-SVM [Lee et al., 2019b][1]	$\mathbf{64.09 \pm 0.62}\%$	$80.00 \pm 0.45\%$
MetaFun-DFP	$\mathbf{64.13 \pm 0.13}\%$	$\mathbf{80.82 \pm 0.17}\%$
MetaFun-KFP	$63.39 \pm 0.15\%$	$\mathbf{80.81 \pm 0.10}\%$

Models	**tieredImageNet 5-way 1-shot**	**tieredImageNet 5-way 5-shot**
(Without deep residual networks feature extraction):		
MAML [Finn et al., 2017a]	$51.67 \pm 1.81\%$	$70.30 \pm 0.08\%$
Prototypical Nets [Snell et al., 2017]	$53.31 \pm 0.89\%$	$72.69 \pm 0.74\%$
Relation Net [in Liu et al. [2019]]	$54.48 \pm 0.93\%$	$71.32 \pm 0.78\%$
Transductive Prop. Nets [Liu et al., 2019]	$57.41 \pm 0.94\%$	$71.55 \pm 0.74\%$
(With deep residual networks feature extraction):		
Meta-SGD	$62.95 \pm 0.03\%$	$79.34 \pm 0.06\%$
LEO	$66.33 \pm 0.05\%$	$81.44 \pm 0.09\%$
MetaOptNet-SVM	$65.81 \pm 0.74\%$	$81.75 \pm 0.58\%$
MetaFun-DFP	$\mathbf{67.72 \pm 0.14}\%$	$82.81 \pm 0.15\%$
MetaFun-KFP	$67.27 \pm 0.20\%$	$\mathbf{83.28 \pm 0.12}\%$

and test (20 classes) meta-sets. The *tieredImageNet* dataset [Ren et al., 2018] contains a larger subset of the ILSVRC-12 dataset. These classes are further grouped into 34 higher-level nodes. These nodes are then divided into training (20 nodes), validation (6 nodes), and test (8 nodes) meta-sets. This dataset is considered more challenging because the split is near the root of the ImageNet hierarchy [Ren et al., 2018]. For both datasets, we use the pre-trained features provided by Rusu et al. [2019], where they train

a 28-layer Wide Residual Network [Zagoruyko and Komodakis, 2016] to classify images in the training meta-set, and extract the intermediate feature representation in layer 21. We use these image embeddings directly as input features in all our experiments. For more details, please see https://github.com/google-deepmind/leo.

Following the commonly used experimental setting, each few-shot classification task consists of 5 randomly sampled classes from a meta-set. Within each class, we have either 1 example (1-shot) or 5 examples (5-shot) as context, and 15 examples as target. For all experiments, hyperparameters are chosen by training on the training meta-set, and comparing target accuracy on the validation meta-set. We conduct randomised hyperparameters search [Bergstra and Bengio, 2012], and the search space is given in Appendix. Then with the model configured by the chosen hyperparameters, we train on the union of the training and validation meta-sets, and report final target accuracy on the test meta-set.

In Table 3.2 we compare our approach to other meta-learning methods. The numbers presented are the mean and standard deviation of 5 independent runs. The table demonstrates that our model outperforms previous state-of-the-art on 1-shot and 5-shot classification tasks for the more challenging tieredImageNet. As for miniImageNet, we note that previous work, such as MetaOptNet-SVM [Lee et al., 2019b], used significant data augmentation to regularise their model and hence achieved superior results. For a fair comparison, we also equipped each model with data augmentation and reported accuracy with/without data augmentation. However, MetaOptNet-SVM [Lee et al., 2019b] uses a different data augmentation scheme involving horizontal flip, random crop, and color (brightness, contrast, and saturation) jitter. On the other hand, MetaFun, Qiao et al. [2018] and LEO [Rusu et al., 2019], only use image features averaging representation of different crops and their horizontal mirrored versions. In 1-shot cases, MetaFun matches previous state-of-the-art performance, while in 5-shot cases, we get significantly better results. In Table 3.2, results for both MetaFun-DFP (using dot-product attention) and MetaFun-KFP (using deep kernels) are reported. Although both of them demonstrate state-of-the-art performance, MetaFun-KFP generally outperforms MetaFun-DFP for 5-shot problems, but performs slightly worse for 1-shot problems.

Table 3.3: Ablation Study. We conduct independent randomised hyperparameter search for each number presented, and reported means and standard deviations over 5 independent runs for each.

Functional pooling	Local update function	Decoder	**MiniImageNet** 5-way 1-shot	5-way 5-shot
Attention	NN	✓	**62.12 ± 0.30**%	77.78 ± 0.12%
Deep Kernel	NN	✓	61.16 ± 0.15%	**78.20 ± 0.16**%
Attention	Gradient	✓	59.63 ± 0.19%	75.84 ± 0.04%
Deep Kernel	Gradient	✓	59.73 ± 0.21%	76.41 ± 0.14%
SE Kernel	NN	✓	60.04 ± 0.19%	75.25 ± 0.12%
Deep Kernel	Gradient	✗	57.67 ± 0.16%	73.55 ± 0.04%
Functional pooling	Local update function	Decoder	**tieredImageNet** 5-way 1-shot	5-way 5-shot
Attention	NN	✓	**67.72 ± 0.14**%	82.81 ± 0.15%
Deep Kernel	NN	✓	67.27 ± 0.20%	**83.28 ± 0.12**%
Attention	Gradient	✓	62.55 ± 0.10%	78.18 ± 0.09%
Deep Kernel	Gradient	✓	65.24 ± 0.11%	80.31 ± 0.16%
SE Kernel	NN	✓	60.81 ± 0.30%	79.70 ± 0.20%
Deep Kernel	Gradient	✗	62.53 ± 0.17%	76.86 ± 0.07%

3.4.3 Ablation study

As stated in Section 3.2.2, our model has three learnable components: the local update function, the functional pooling, and the decoder. In this section we explore the effects of using different versions of these components. We also investigate how the model performance would change with different numbers of iterations.

Table 3.3 demonstrates that neural network parameterised local update functions, described in Section 3.2.1, consistently outperforms gradient-based local update function, despite the latter having build-in inductive biases. Interestingly, the choice between dot-product attention and deep kernel in functional pooling is problem dependent. We found that MetaFun with deep kernels usually perform better than MetaFun with dot product attention on 5-shot classification tasks, but worse on 1-shot tasks. We conjecture that the deep kernel is better able to fuse the information across the 5 images per class compared to attention. In the comparative experiments in Section 3.4.2 we reported results on both.

In addition, we investigate how a simple Squared Exponential (SE) kernel would perform on these few-shot classification tasks. This corresponds to using an identity input transformation function a in deep kernels. Table 3.3 shows that using SE kernel

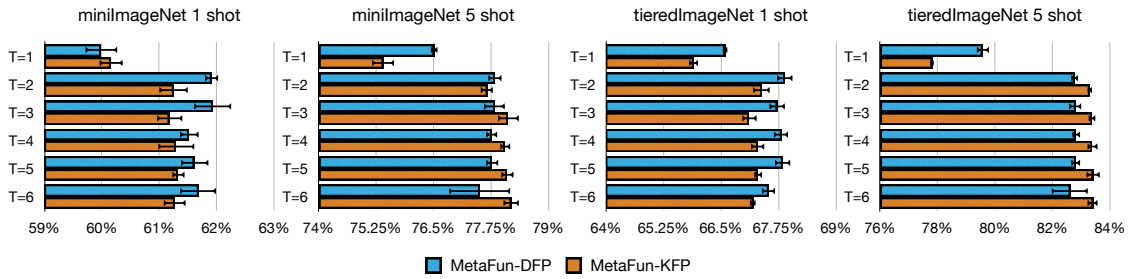


Figure 3.5: This figure illustrates the accuracy of our approach for varying number of iterations $T = 1, \ldots, 6$, over different few-shot learning problems. For each problem, we use the same configuration of hyperparameters except for the number of iterations and the choice between attention and deep kernels. Error bars (standard deviations) are given by training the same model 5 times with different random seeds.

is consistently worse than using deep kernels, showing that the heavily parameterised deep kernel is necessary for these problems.

Next, we looked into directly applying functional gradient descent with parameterised deep kernel to these tasks. This corresponds to removing the decoder and using deep kernels and gradient-based local update function (see Section 5.4). Unsurprisingly, this did not fare as well, given as it only has one trainable component (the deep kernel) and the updates are directly applied to the predictions rather than a latent functional representation.

Finally, Figure 3.5 illustrates the effects of using different numbers of iterations T. On all few-shot classification tasks, we can see that using multiple iterations (two is often good enough) always significantly outperform one iteration. We also note that this performance gain diminishes as we add more iterations. In Section 3.4.2 we treated the number of iterations as one of the hyperparameters.

3.5 Conclusions and future work

In this paper, we propose a novel functional approach for meta-learning called MetaFun. The proposed approach learns to generate a functional task representation and an associated functional update rule, which allows to iteratively update the task representation directly in the function space. We evaluate MetaFun on both few-shot regression and classification tasks, and demonstrate that it matches or exceeds previous

gradient-based methods on miniImageNet and tieredImageNet few-shot classification tasks.

Interesting future research directions include a) exploring a stochastic encoder and hence working with stochastic functional representations, akin to the Neural Process (NP), and b) using local update functions and the functional pooling components whose parameters change with iterations instead of sharing them across iterations, where the added flexibility could lead to further performance gains.

3.6 Supplementary materials

3.6.1 Functional gradient descent

Functional gradient descent [Mason et al., 1999, Guo et al., 2001] is an iterative optimisation algorithm for finding the minimum of a function. However, the function to be minimised is now a function on functions (*functional*). Formally, a functional $L : \mathcal{H} \to \mathbf{R}$ is a mapping from a function space $\mathcal{H}$ to a $1D$ Euclidean space $\mathbf{R}$. Just like gradient descent in parameter space which takes steps proportional to the negative of the gradient, functional gradient descent updates f following the gradient in function space. In this work, we only consider a special function space called RKHS (Section 3.6.1.1), and calculate functional gradients in RKHS (Section 3.6.1.2). The algorithm is further detailed in Section 3.6.1.3.

3.6.1.1 Reproducing kernel Hilbert space

A Hilbert space $\mathcal{H}$ extends the notion of Euclidean space by introducing inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which describes the concept of distance or similarity in this space. A RKHS $\mathcal{H}_k$ is a Hilbert space of real-valued functions on $\mathcal{X}$ with the reproducing property that for all $\mathbf{x} \in \mathcal{X}$ there exists a unique $k_{\mathbf{x}} \in \mathcal{H}_k$ such that the *evaluation functional* $E_{\mathbf{x}}(f) = f(\mathbf{x})$ can be represented by taking the inner product of this element $k_{\mathbf{x}}$ and f, formally as:

$$E_{\mathbf{x}}(f) = \langle k_{\mathbf{x}}, f \rangle_{\mathcal{H}_k}. \tag{3.19}$$

Since $k_{\mathbf{x}'} \in \mathcal{H}_k$ for any $\mathbf{x}' \in \mathcal{X}$, we can define a kernel function $k(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbf{R}$

by letting

$$k(\mathbf{x}, \mathbf{x}') = k_{\mathbf{x}'}(\mathbf{x}) = \langle k_{\mathbf{x}}, k_{\mathbf{x}'} \rangle_{\mathcal{H}_k}. \tag{3.20}$$

Using properties of inner product, it is easy to show that the kernel function $k(\mathbf{x}, \mathbf{x}')$ is symmetric and positive definite, and we call it the *reproducing kernel* of the Hilbert space $\mathcal{H}_k$.

3.6.1.2 Functional gradients

Functional derivative can be thought of as describing the rate of change of the output with respect to the input in a functional. Formally, functional derivative at point f in the direction of g is defined as:

$$\frac{\partial L}{\partial f}(g) = \lim_{\epsilon \to 0} \frac{L(f + \epsilon g) - L(f)}{\epsilon}, \tag{3.21}$$

which is a function of g. This is known as *Fréchet derivative* in a Banach space, of which the Hilbert space is a special case.

Functional gradient, denoted as $\nabla_f L$, is related to functional derivative by the following equation:

$$\frac{\partial L}{\partial f}(g) = \langle \nabla_f L, g \rangle_{\mathcal{H}_k}. \tag{3.22}$$

Thanks to the reproducing property, it is straightforward to calculate functional derivative of an evaluation functional in RKHS:

$$\begin{aligned} E_{\mathbf{x}}(f + \epsilon g) &= \langle f + \epsilon g, k_{\mathbf{x}} \rangle_{\mathcal{H}_k} \\ &= \langle f, k_{\mathbf{x}} \rangle_{\mathcal{H}_k} + \epsilon \langle g, k_{\mathbf{x}} \rangle_{\mathcal{H}_k} \qquad (3.23) \\ \frac{\partial E_{\mathbf{x}}}{\partial f}(g) &= \langle k_{\mathbf{x}}, g \rangle_{\mathcal{H}_k} \qquad (3.24) \end{aligned}$$

Therefore, the functional gradient of an evaluation functional is:

$$\nabla_f E_{\mathbf{x}} = k_{\mathbf{x}}. \tag{3.25}$$

For a learning task with loss function ℓ and a context set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in \mathbf{C}}$, the overall supervised loss on the context can be written as:

$$L(f) = \sum_{i \in \mathbf{C}} \ell(f(\mathbf{x}_i), \mathbf{y}_i). \tag{3.26}$$

In this case, the functional gradient of L can be easily calculated by applying the chain rule:

$$\nabla_f L = \sum_{i \in \mathbf{C}} \ell'(f(\mathbf{x}_i), \mathbf{y}_i) k_{\mathbf{x}_i} \tag{3.27}$$

$$= \sum_{i \in \mathbf{C}} k(\cdot, \mathbf{x}_i) \ell'(f(\mathbf{x}_i), \mathbf{y}_i). \tag{3.28}$$

3.6.1.3 Functional gradient descent

To optimise the overall loss on the entire context in Equation (3.26), we choose a suitable learning rate α, and iteratively update f with:

$$f^{(t+1)}(\mathbf{x}) = f^{(t)}(\mathbf{x}) - \alpha \nabla_f L(f^{(t)})(\mathbf{x}) \tag{3.29}$$

$$= f^{(t)}(\mathbf{x}) - \alpha \sum_{i \in \mathbf{C}} k(\mathbf{x}, \mathbf{x}_i) \ell'(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i) \tag{3.30}$$

In order to evaluate the final model $f^T(\mathbf{x})$ at iteration T, we only need to compute

$$f^{(T)}(\mathbf{x}) = f^{(0)}(\mathbf{x}) - \sum_{t=0}^{T-1} \alpha \sum_{i \in \mathbf{C}} k(\mathbf{x}, \mathbf{x}_i) \ell'(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i), \tag{3.31}$$

which does not depend on function values outside the context from previous iterations $t < T$.

3.6.2 Experimental details

We run experiments on Nvidia's GeForce GTX 1080 Ti, and it typically takes about 20–40 minutes to train a few-shot model on a single GPU card until early-stopping is triggered (after seeing $10k$–$100k$ tasks). For miniImageNet and tieredImageNet, we conduct randomised hyperparameters search [Bergstra and Bengio, 2012] for hyperparameters tunning. Typically, 64 configurations of hyperparameters are sampled for each problem, and the best configuration is chosen by comparing accuracy on the validation set. The considered range of hyperparameters is given in Table 3.4, and the chosen hyperparameters are shown in Table 3.5. For regression tasks, we simply use hyperparameters listed in Table 3.6 for both MetaFun-DFP and MetaFun-KFP.

Table 3.4: Considered Range of Hyperparameters. The random generators such as `randint` or `uniform` use `numpy.random` syntax, so the first argument is inclusive while the second argument is exclusive. Whenever a list is given, it means uniformly sampling from the list. u_+ and u_- will be followed by a linear transformation with an output dimension of *dim-reprs*.

Components	Architecture
Shared MLP m	*nn-sizes* $\times$ *nn-layers*
MLP for positive labels u_+	*nn-sizes* $\times$ *nn-layers*
MLP for negative labels u_-	*nn-sizes* $\times$ *nn-layers*
Key/query transformation MLP a	*dim*($\mathbf{x}$) $\times$ *embedding-layers*
Decoder	linear with output dimension *dim*($\mathbf{x}$)

Hyperparameters	Considered Range
num-iters	`randint(2, 7)`
nn-layers	`randint(2, 4)`
embedding-layers	`randint(1, 3)`
nn-sizes	$[64, 128]$
dim-reprs	$=$*nn-sizes*
Initial representation $\mathbf{r}^0$	[zero, constant, parametric]
Outer learning rate	$10^{-5} \times$ `uniform(-5, -4)`
Initial inner learning rate	$[0.1, 1.0, 10.0]$
Dropout rate	`uniform(0.0, 0.5)`
Orthogonality penalty weight	$10^{\texttt{uniform(-4, -2)}}$
L2 penalty weight	$10^{\texttt{uniform(-10, -8)}}$
Label smoothing	$[0.0, 0.1, 0.2]$

Table 3.5: Results of randomised hyperparameters search. Hyperparameters shown in this table are not guaranteed to be optimal within the considered range, because we conduct randomised hyperparameters search. However, models configured with these hyperparameters perform reasonably well, and we used them to report final results comparing to other methods. Furthermore, dropout is only applied to the inputs. Orthogonality penalty weight and L2 penalty weight are used in exactly the same way as in Rusu et al. [2019]. Inner learning rate α is trainable so only an initial inner learning rate is given in the table.

	miniImageNet		**tieredImageNet**	
Hyperparameters (for MetaFun-DFP)	1-shot	5-shot	1-shot	5-shot
num-iters	2	5	3	5
nn-layers	3	2	2	3
embedding-layers	2	2	1	1
nn-sizes	64	128	128	128
Initial state	zero	constant	constant	constant
Outer learning rate	8.56×10^{-5}	3.71×10^{-5}	5.55×10^{-5}	5.78×10^{-5}
Initial inner learning rate	0.1	10.0	1.0	1.0
Dropout rate	0.397	0.075	0.123	0.223
Orthogonality penalty weight	3.28×10^{-3}	1.56×10^{-3}	1.37×10^{-3}	2.58×10^{-3}
L2 penalty weight	1.32×10^{-10}	2.60×10^{-10}	1.92×10^{-9}	1.63×10^{-9}
Label smoothing	0.2	0.2	0.1	0.0

	miniImageNet		**tieredImageNet**	
Hyperparameters (for MetaFun-KFP)	1-shot	5-shot	1-shot	5-shot
num-iters	3	6	4	4
nn-layers	3	2	2	3
embedding-layers	2	2	1	1
nn-sizes	64	64	64	128
Initial state	zero	parametric	parametric	zero
Outer learning rate	4.21×10^{-5}	8.60×10^{-5}	8.01×10^{-5}	4.50×10^{-5}
Initial inner learning rate	0.1	0.1	0.1	0.1
Dropout rate	0.424	0.359	0.115	0.148
Orthogonality penalty weight	2.69×10^{-3}	2.73×10^{-4}	1.06×10^{-4}	7.33×10^{-3}
L2 penalty weight	1.19×10^{-9}	1.68×10^{-9}	4.90×10^{-9}	6.22×10^{-9}
Label smoothing	0.2	0.2	0.1	0.1

Table 3.6: Hyperparameters for regression tasks. Local update function and the predictive model will be followed by linear transformations with output dimension of *dim-reprs* and $dim(\mathbf{y})$ accordingly.

Components	Architecture
Local update function	*nn-sizes* × *nn-layers*
Key/query transformation MLP a	*nn-sizes* × *embedding-layers*
Decoder	*nn-sizes* × *nn-layers*
Predictive model	*nn-sizes* × (*nn-layers*-1)

Hyperparameters	Considered Range
num-iters	5
nn-layers	3
embedding-layers	3
nn-sizes	128
dim-reprs	=*nn-sizes*
Initial representation $\mathbf{r}^0$	zero
Outer learning rate	10^{-4}
Initial inner learning rate	0.1
Dropout rate	0.0
Orthogonality penalty weight	0.0
L2 penalty weight	0.0

Chapter 4

Deep Stochastic Processes via Functional Markov Transition Operators

4.1 Introduction

SPs are widely used in many scientific disciplines, including biology [Bressloff, 2021], chemistry [van Kampen and Reinhardt, 1981], neuroscience [Laing and Lord, 2009], physics [Paul and Baschnagel, 2000] and control theory [Bertsekas and Shreve, 2007]. They are formed by a (typically infinite) collection of random variables and can be used to model data by considering the conditional distribution of target variables given observed context variables. In machine learning, SPs in the form of Bayesian nonparametric models—such as GPs [Rasmussen and Williams, 2009] and Dirichlet processes [Teh et al., 2004]—are used in tasks such as regression, classification, and clustering. SPs parameterised by neural networks have also been used for meta-learning [Garnelo et al., 2018a, Xu et al., 2020] and generative modelling [Mathieu et al., 2021, Dupont et al., 2022].

With the increasing amount of data available, and the complex patterns arising in many applications, more flexible and scalable SP models with greater learning capacity are required. The NP family [Garnelo et al., 2018a,c, Kim et al., 2018, Gordon et al., 2019, Foong et al., 2020] meets this demand by parameterising SPs with neural networks, and enjoys greater flexibility and computational efficiency compared to traditional nonparametric models.

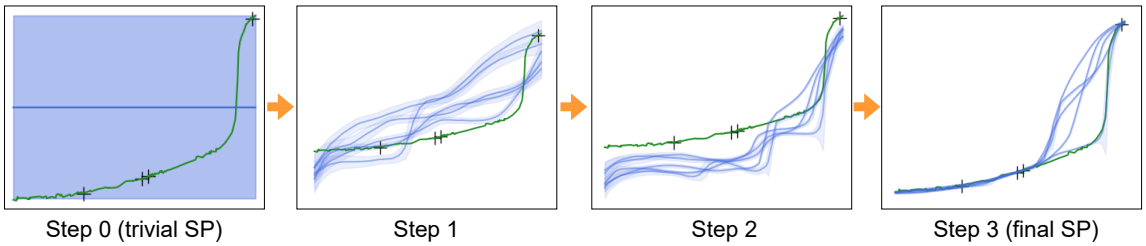


Figure 4.1: **MNPs construct expressive SPs via iterative transformations.** Here we use MNPs to model distributions over monotonic functions. We start with trivial initial SPs and gradually transform them into more complex SPs conditioned on observed context points marked as black crosses. The blue lines represent sampled mean functions and the shaded region indicates one standard deviation.

Unfortunately, the original version of NPs [Garnelo et al., 2018c] lacks expressivity and often underfits the data in practice [Kim et al., 2018]. Various extensions have therefore been proposed—such as ANPs [Kim et al., 2018], CONVNPs [Gordon et al., 2019, Foong et al., 2020] and Gaussian Neural Processes (GNPs) [Bruinsma et al., 2021]—to improve expressivity. These models—which we refer to as *predictive* SPs—are based around directly constructing mappings from contexts to predictive distributions on targets, and enhance expressivity by modelling context-target interactions with attentions or convolutions. Predictive SP models map from a context $\mathcal{C}$ to a predictive SP $p(f;\mathcal{C})$ and make a clear distinction between the context and the target in model specification. Consequently, for these models $p(f;\mathcal{C})$ no longer corresponds to a posterior derived from a certain prior $p(f)$, though they are typically constructed to ensure $p(f;\mathcal{C})$ is itself a valid SP for new evaluations of f with $\mathcal{C}$ held fixed.

We propose an alternative mechanism to extend the NP family and provide increased expressivity by generalising the marginal density functions of NPs. The generalised form can be viewed as Markov transition operators with functions as states. This lays the foundation for stacking these operators to construct more powerful SP models that we call Markov Neural Processes (MNPs). MNPs can be seen as Markov chains in *function* space, parameterised by neural networks: they make iterative transformations from simple initial SPs into more flexible, yet properly defined, SPs (as illustrated in Figure 4.1), without compromising consistency or introducing additional assumptions.

To empirically demonstrate the value of our proposed approach, we first benchmark MNPs against baselines on 1D function regression tasks, and conduct ablation studies in this controlled setting. We then show that they can be used as a high-performing surrogate for contextual bandit problems. Finally, we apply them to geological data,

for which they demonstrate encouraging performance.

4.2 Background

A SP is a (typically infinite) collection of random variables defined on a common probability space. We can consider a SP as a random function $F : \mathcal{X} \to \mathcal{Y}$ where inputs can be regarded as indexing the output random variables. With a relaxed use of notation, we employ $p(f)$ in denoting a SP, where f maps inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. Kolmogorov's Consistency Theorem shows that a SP can be *indirectly* defined via a collection of marginal distributions, $\{p_{x_{1:n}}(y_{1:n})\}_{x_{1:n} \in \mathcal{X}^n}$ (we drop $\mathcal{X}^n$ from here on for conciseness) if they, for any permutation π and all possible sets of inputs $x_{1:n} \in \mathcal{X}^n$, satisfy the *exchangeability* condition:

$$p_{x_{1:n}}(y_{1:n}) = p_{\pi(x_{1:n})}(\pi(y_{1:n})) := \; p_{x_{\pi(1)},\ldots,x_{\pi(n)}}(y_{\pi(1)}, \ldots, y_{\pi(n)}), \tag{4.1}$$

and the *(marginal) consistency* condition:

$$p_{x_{1:m}}(y_{1:m}) = \int p_{x_{1:n}}(y_{1:n}) \, \mathrm{d}y_{m+1:n} \qquad \forall 1 \leq m < n. \tag{4.2}$$

If none of the random variables in a SP are observed, we call it a *prior* SP. For any two distinct subsets of datapoints $\mathcal{C} = \{(x_i, y_i)\}_{i=1}^m$ (the context) and $\mathcal{T} = \{(x_i, y_i)\}_{i=m+1}^n$ (the target), one can use this prior SP to compute the conditional density $p_{x_{m+1:n}|x_{1:m}}(y_{m+1:n}|y_{1:m})$ via Bayesian inference. If inference is exact, it can be proved that the collection of conditional distributions $\{p_{x_{m+1:n}|x_{1:m}}(y_{m+1:n}|y_{1:m})\}_{x_{m+1:n}}$ also satisfy exchangeability and consistency, and hence define a valid *posterior* SP, $p(f|\mathcal{C})$. We discuss the impact of approximate inference in Section 4.3.3.

Based on a conditional version of de Finetti's Theorem, a NP [Garnelo et al., 2018c] defines a SP by providing a collection of exchangeable and consistent marginal densities parameterized by neural networks. Specifically, they set up their marginal densities to have the form:

$$p_{x_{1:n}}(y_{1:n}; \theta) = \int p_\theta(z) \prod_{i=1}^n \mathcal{N}(y_i | \mu_\theta(x_i, z), \sigma_\theta^2(x_i, z)) \mathrm{d}z, \tag{4.3}$$

where z is a latent variable which captures dependencies across different input locations, $y_i := f(x_i)$, μ_θ and σ_θ are deep neural networks, and $p_\theta(z)$ is a, typically Gaussian, prior distribution on the latents. Note that this form is more general than the one in [Garnelo et al., 2018c] where both p_θ and σ_θ are not learnt.

4.3 Markov neural processes

To correct the shortfalls of NPs while maintaining their conditional consistency, we now introduce a more expressive family of generative SP models termed Markov Neural Processes (MNPs). Our starting point is to extend NP density functions into a generalised form that can be viewed as a transition operator which transforms a trivial SP to a more flexible one. MNPs are then formed by stacking sequences of these transition operators to form a highly expressive and flexible model class. The training and inference procedure for MNP mirrors that of NP, but we also introduce a novel inference model that allows for efficient learning in our scenario.

4.3.1 A more general form of Neural Process density functions

Recall the form of the NP marginal density functions from Equation (4.3). One can draw joint samples from $p_{x_{1:n}}(y_{1:n}; \theta)$ via reparameterization using:

$$y_{1:n}^{(0)} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}), \quad z \sim p_\theta(z), \quad y_i = \sigma_\theta(z, x_i) \cdot y_i^{(0)} + \mu_\theta(z, x_i). \tag{4.4}$$

The key starting point for MNPs is to show that this can be generalised to the case where $y_{1:n}^{(0)}$ is drawn from any given SP of its own, $p(f^{(0)})$, and each y_i is any invertible transformation, F_θ, of $y_i^{(0)}$, parameterized by x_i and z. Specifically, we introduce the following result (see Section 4.7.1 for proof).

Proposition 4.3.1. *Let $F_\theta(\cdot; x, z) : \mathcal{Y} \mapsto \mathcal{Y}$ denote an invertible transformation between outputs, parameterized by the input and latent. If $\{p_{x_{1:n}}(y_{1:n}^{(0)})\}_{x_{1:n}}$ is a valid* SP *(i.e. it satisfies Equation (4.1) and Equation (4.2)) and*

$$y_{1:n}^{(0)} \sim \{p_{x_{1:n}}(y_{1:n}^{(0)})\}_{x_{1:n}}, \quad z \sim p_\theta(z), \quad y_i = F_\theta(y_i^{(0)}; x_i, z). \tag{4.5}$$

then $\{p_{x_{1:n}}(y_{1:n})\}_{x_{1:n}}$ is also a valid SP.

Our next step is to realise that Equation (4.5) can be viewed as a Markov transition in function space, denoted as $p(f|f^{(0)})$, which transforms a simpler SP $p(f^{(0)})$ to a more expressive one $p(f)$. We can thus generalise things further by stacking sequences of Markov transition operators in function spaces (fMTOs), denoted by $p(f^{(1)}|f^{(0)})$, $\ldots$, $p(f^{(T)}|f^{(T-1)})$, together to form a Markov chain $f^{(0)} \to \cdots \to f^{(T)}$ in function space. This will be the basis of MNPs.

4.3.2 Markov chains in function space

Analogously to defining a SPs through its marginals, we indirectly specify FMTOs using a collection of marginal Markov transition operators (MMTOs), denoted by $\{p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\}_{x_{1:n}}$, where each MMTO is just Markov transition operator over a finite sequence of function outputs.

To adapt the definitions of consistency and exchangeability for SPs to the transition operator setting, we say that the MMTOs are consistent if and only if, for any $1 < m < n$ and sequence $x_{1:n} \in \mathcal{X}^n$,

$$\int p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\,\mathrm{d}y_{m+1:n} = p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:n}) = p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m}). \tag{4.6}$$

Similarly, MMTOs are exchangeable if, and only if, for all possible permutations π,

$$p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) = p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y^{(0)}_{1:n})). \tag{4.7}$$

Note that, if we consider $(x_i, y^{(0)}_i)$ as inputs, and y_i as function outputs, the transition operator $p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})$ can be seen as the finite marginals of a random function $F' : \mathcal{X} \times \mathcal{Y} \to \mathcal{Y}$ whose distribution is a SP, and the conditions (4.6) and (4.7) correspond to (4.2) and (4.1).

Provided that these MMTOs are consistent and exchangeable, the FMTOs in function space will also be well-defined indirectly—i.e. the transition produces a well-defined SP given input random functions from a SP—as per the following result (see Section 4.7.1 for proof):

Proposition 4.3.2. *If the collection of marginals before transition $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ is consistent and exchangeable (as per Equations (4.1) and (4.2)) and the collection of MMTOs $\{p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\}_{x_{1:n}}$ is also consistent and exchangeable (as per Equations (4.6) and (4.7)), then the collection of marginals after transition $\{p_{x_{1:n}}(y_{1:n})\}$ is also consistent and exchangeable, hence defining a valid SP.*

Furthermore, a Markov chain in function space can be constructed by a sequence of FMTOs $p(f^{(1)}|f^{(0)}), \ldots, p(f^{(T)}|f^{(T-1)})$ where $p(f^{(t)}|f^{(t-1)}) \coloneqq \{p_{x_{1:n}}(y^{(t)}_{1:n}|y^{(t-1)}_{1:n})\}_{x_{1:n}}$. With repeated applications of Proposition 4.3.2, if the initial state $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ is exchangeable and consistent, $\{p_{x_{1:n}}(y^{(T)}_{1:n})\}_{x_{1:n}}$ at time T is also exchangeable and consistent, hence defining a SP $p(f^{(T)})$.

Equation (4.5) then provides a valid construction of consistent and exchangeable MMTOs by introducing an auxiliary latent variable z. The transition operator is written as

$$p_{x_{1:n}}(y_{1:n}|y_{1:n}^{(0)};\theta) = \int p_\theta(z) \prod_{i=1}^{n} \delta(y_i - F_\theta(y_i^{(0)}; x_i, z))\,\mathrm{d}z, \tag{4.8}$$

where both p_θ and F_θ in Equation (4.8) can be parameterised by neural networks. Critically, we can extend Proposition 4.3.2 to cover these auxiliary settings in Equation (4.8) as well, as per the following result (see Section 4.7.1 for proof):

Proposition 4.3.3. MMTOs *in the form of Equation* (4.8) *are consistent and exchangeable.*

4.3.3 Parameterisation, inference and training

We can now define a MNP as a sequence of neural FMTOs, with each FMTO indirectly specified via a collection of MMTOs in the form of Equation (4.8). If we specify a distribution over the sequence of auxiliary latent variables $p_\theta(z^{(1:T)})$ along with an initial SP with marginals $p_{x_{1:n}}(y_{1:n}^{(0)})$, we can write down the marginal distribution over function outputs $y_{1:n} := y_{1:n}^{(T)}$ for a sequence of inputs $x_{1:n}$ under the MNP model as (see Figure 4.2a for illustration and Section 4.7.1 for derivation):

$$\begin{aligned} p_{x_{1:n}}(y_{1:n};\theta) &= \int p_\theta(z^{(1:T)}) p_{x_{1:n}}(y_{1:n}^{(0)}) \prod_{t=1}^{T}\prod_{i=1}^{n} p_\theta^{(t)}(y_i^{(t)} \mid y_i^{(t-1)}, x_i, z^{(t)}) \mathrm{d}y_{1:n}^{(0:T-1)} \mathrm{d}z^{(1:T)} \\ &= \int p_\theta(z^{(1:T)}) p_{x_{1:n}}(y_{1:n}^{(0)}) \prod_{t=1}^{T}\prod_{i=1}^{n} \left|\det \frac{\partial F_\theta^{(t)}(y_i^{(t-1)}; x_i, z^{(t)})}{\partial y_i^{(t-1)}}\right| \mathrm{d}z^{(1:T)} \end{aligned} \tag{4.9}$$

According to Propositions 4.3.2 and 4.3.3, $\{p_{x_{1:n}}(y_{1:n};\theta)\}_{x_{1:n}}$ defines a valid SP parameterised by θ. The initial SP can be arbitrarily chosen, as long as we can evaluate its marginals $p_{x_{1:n}}(y_{1:n}^{(0)})$. In our experiments, we use a trivial SP where all the outputs are i.i.d. standard normal distributed. We adopt normalising flows [Rezende and Mohamed, 2015, Papamakarios et al., 2019, Durkan et al., 2019] to parameterise the invertible transformations $F_\theta^{(t)}$.

To integrate over latent variables $z^{(1:T)}$ in Equation (4.9), we introduce a posterior approximation $q_{x_{1:n}}(z^{(1:T)}|y_{1:n};\phi)$. For many applications, we need to learn and

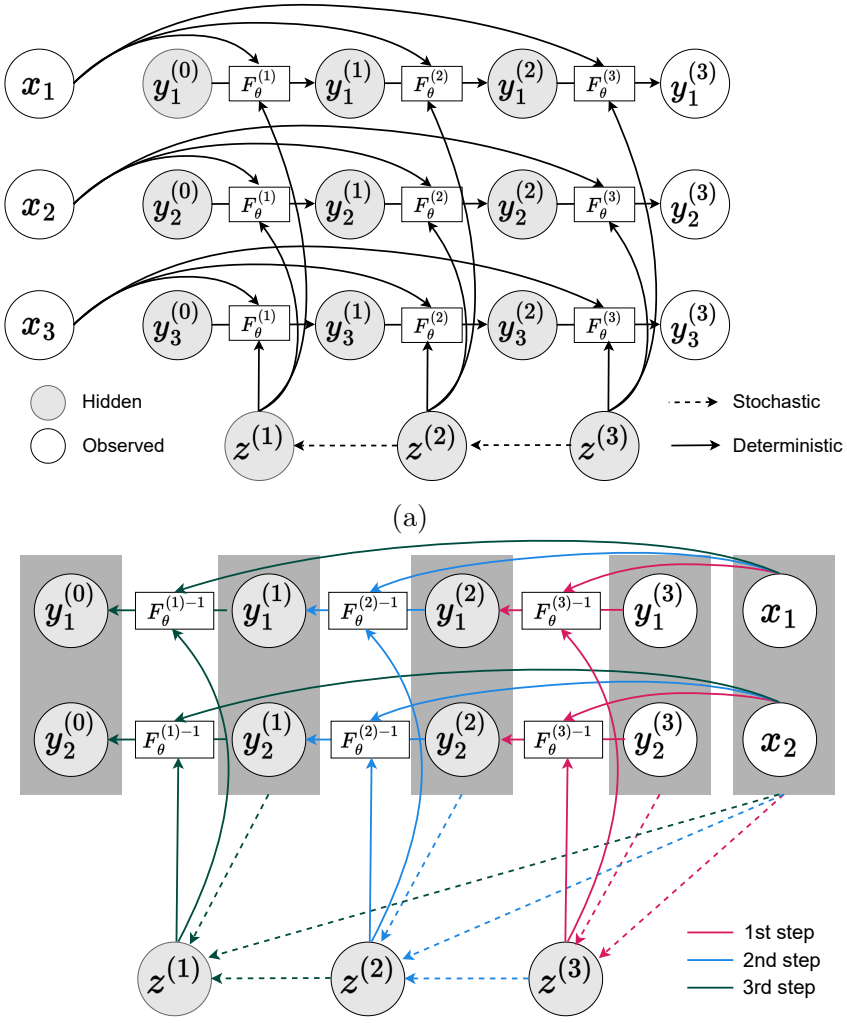


Figure 4.2: (a) **Consistent generative models on finite sets**. Observed variables are shown in white, and hidden variables are shaded. Stochastic paths are indicated with dashed lines while deterministic paths are solid lines. Conditioned on the function inputs $x_{1:n}$ and the auxiliary latent variables $z^{(1:T)}$, the function outputs are transformed independently using instance-wise conditional normalising flows (CNFs). If the initial states $\{(x_i, y_i)\}_{i=1:n}$ come from a SP, the construction will ensure that the collection of marginals $p(y^{(t)}_{1:n}|x_{1:n})$ are consistent under marginalisation for any t. (b) **Permutation-invariant inference models**. Auxiliary latent variables $z^{(1:T)}$ are inferred in reverse order in our inference model. We reuse the parameters of CNFs in the generative model to compute function values $y^t_{1:n}$ at all time steps, where each step has a different colour. We parameterise the conditional distribution $q^{(t)}_{x_{1:n}}(z^{(t)}|z^{(t+1)}, y^{(t)}_{1:n}; \phi)$ with permutation-invariant neural networks.

query the conditional distributions of a target $\mathcal{T} = \{(x_i, y_i)\}_{i=m+1}^n$ given context $\mathcal{C} = \{(x_i, y_i)\}_{i=1}^m$. To better reflect the desired model behaviour at test time, similar to NPs [Garnelo et al., 2018c], we train the model by maximising the following approximation of the conditional log-likelihood w.r.t. both model θ and variational ϕ parameters:

$$\mathcal{L}_{\theta,\phi}(y_{1:n}; x_{1:n}) := \mathbb{E}_{q_{x_{1:n}}(z^{(1:T)} \mid y_{1:n}; \phi)} \left[\log p_{x_{m+1:n}}(y_{m+1:n} \mid z^{(1:T)}, \theta) + \log \frac{q_{x_{1:m}}(z^{(1:T)} \mid y_{1:m}; \phi)}{q_{x_{1:n}}(z^{(1:T)} \mid y_{1:n}; \phi)} \right] \tag{4.10}$$

where $\log p_{x_{m+1:n}|x_{1:m}}(y_{m+1:n} \mid y_{1:m}; \theta) \gtrapprox \mathcal{L}_{\theta,\phi}(y_{1:n}; x_{1:n})$. Here $q_{x_{1:m}}(z^{(1:T)} \mid y_{1:m}; \phi)$ can be seen as an approximate prior conditioned on the context $\{(x_i, y_i)\}_{i=1}^m$, while $q_{x_{1:n}}(z^{(1:T)} \mid y_{1:n}; \phi)$ is the approximate posterior after the target $\{(x_i, y_i)\}_{i=m+1}^n$ is observed. Thus the prior and the posterior share the same inference model with parameters ϕ. The training objective is optimised by stochastic gradient descent, and the gradients $\nabla_{\theta,\phi}\mathcal{L}_{\theta,\phi}(y_{1:n}; x_{1:n})$ can be efficiently estimated with low variance using the reparameterisation trick [Mohamed et al., 2020].

However, different from NPs, the latent variables $z^{(1:T)}$ for MNPs are a sequence. Therefore, we propose a different inference model which shares parameters with the generative model above. Firstly, we write $q_{x_{1:m}}(z^{(1:T)} \mid y_{1:m}; \phi)$ in a factorised form:

$$q_{x_{1:n}}(z^{(1:T)} \mid y_{1:n}; \phi) = \prod_{t=1}^T q_{x_{1:n}}^{(t)}(z^{(t)} | z^{(t+1)}, y_{1:n}^{(t)}; \phi) \tag{4.11}$$

where ϕ are variational parameters and we set $z^{(T+1)} = \mathbf{0}$, $y_{1:n} = y_{1:n}^{(T)}$. In our implementation, we use a Gaussian distribution for each factor $q_{x_{1:n}}^{(t)}$, with mean $\mu_\phi^{(t)}(z^{(t+1)}, y_{1:n}^{(t)}, x_{1:n})$ and variance $\Sigma_\phi^{(t)}(z^{(t+1)}, y_{1:n}^{(t)}, x_{1:n}))$, Here $\mu_\phi^{(t)}$ and $\Sigma_\phi^{(t)}$ are parameterised functions invariant to the permutation of data points $\{(x_i, y_i)\}_{i=1}^n$, i.e.

$$\mu_\phi^{(t)}(z^{(t+1)}, y_{1:n}^{(t)}, x_{1:n}) = \mu_\phi^{(t)}(z^{(t+1)}, y_{\pi(1:n)}^{(t)}, x_{\pi(1:n)})$$
$$\Sigma_\phi^{(t)}(z^{(t+1)}, y_{1:n}^{(t)}, x_{1:n}) = \Sigma_\phi^{(t)}(z^{(t+1)}, y_{\pi(1:n)}^{(t)}, x_{\pi(1:n)}).$$

We can parameterise them by first having

$$r^{(t)} = \textsc{SetEncoder}(y_{1:n}^{(t)}, x_{1:n})) \tag{4.12}$$

where SetEncoder could be a Set Transformer [Lee et al., 2019a], Deep Sets [Zaheer et al., 2017], then taking

$$\mu_\phi^{(t)}(z^{(t+1)}, y_{1:n}^{(t)}, x_{1:n}) = \text{MLP}_\mu(z^{(t+1)}, r^{(t)}) \tag{4.13}$$
$$\sigma_\phi^{(t)}(z^{(t+1)}, y_{1:n}^{(t)}, x_{1:n}) = \text{MLP}_\sigma(z^{(t+1)}, r^{(t)}). \tag{4.14}$$

In Equation (4.11), the intermediate function outputs $y_{1:n}^{(1:T-1)}$ are not observed. However, we can sample them autoregressively by iterating between sampling z^{t+1} and calculating $y_i^{(t)} = (F_\theta^{(t+1)})^{-1}(y_i^{(t+1)}; x_i, z^{(t+1)})$ for each i (see Figure 4.2b). Therefore, we share the parameters of the normalising flows $F_\theta^{(1:T)}$ between the generative and inference models, leading to better performance. This idea is originally was proposed by Cornish et al. [2020], except that our normalising flows are applied to a sequence of function outputs $y_{1:n}$ given the inputs $x_{1:n}$.

In practice, the inference model $q_{x_{1:n}}(z^{(1:T)} \mid y_{1:n}; \phi)$ provides an approximate prior/posterior. Ideally, if it gives the exact posterior, conditional consistency would hold perfectly. However, when the inference model is approximate, the degree of conditional consistency depends on the discrepancy between the inference model and the true posterior. Predictive SP models also have a stochastic mapping conditioned on the context, represented as $q_{x_{1:m}}(z^{(1:T)}|y_{1:m}; \phi)$. However, it is important not to confuse this with the inference model, as they do not serve to approximate the posterior.

4.4 Related work

Bayesian nonparametric models Bayesian nonparametric models such as GPs [Rasmussen and Williams, 2006, Ghahramani, 1999] and Student-t processes [Shah et al., 2014, Bui et al., 2016, Chung and Turner, 2019] provide common classical approaches for modelling distributions over functions. Under these models, any conditional distribution of a target given a context can be directly evaluated, and both marginal/conditional consistency is guaranteed by construction (if all computation is exact). However, they can be too restrictive for some applications, e.g. any conditional or marginal distribution of a GP is also Gaussian. Further, the evaluation of conditional densities is typically computationally intensive (cubic w.r.t. the context due to matrix inversion). Deep GPs [Damianou and Lawrence, 2013, Wilson et al., 2016] use GPs as building blocks to construct deep architectures, designed to be flexible enough to model a wide range of complex systems. However, only the hyperparameters for each GP layer are tunable, which means they can still be restrictive when modelling highly non-Gaussian data.

Copula-based processes In multivariate statistics, a copula function refers to a multivariate function that describes the dependence between random variables and links the marginal distributions of each variable to the joint distribution of

all the variables [Nelsen, 2007, Joe, 1997]. Similarly, a copula process [Wilson and Ghahramani, 2010] describes the dependency between infinitely many random variables independent of their marginal distributions. Korshunova et al. [2018, 2020], Maroñas et al. [2021] exploited this independence to specify the more flexible Copula-Based Processes (CBPs) by combing the copula processes of existing SPs with flexible models of marginal distributions based on normalising flows [Rezende and Mohamed, 2015, Papamakarios et al., 2019, Durkan et al., 2019]; the consistency of CBPs is guaranteed as long as the base SPs are consistent. However, CBPs are still restrictive in terms of modelling relationship between variables because the underlying copula processes still come from known SPs. For example, BRUNOs [Korshunova et al., 2018, 2020] have the same underlying copula processes as GPs, so they cannot be used to model data with non-Gaussian dependencies.

Neural process family NPs [Garnelo et al., 2018c] are generative SP models which specify a prior SP and rely on Bayesian inference to compute conditional densities. To improve expressivity of the original NPs, Kim et al. [2018], Foong et al. [2020], Bruinsma et al. [2021] explore predictive SP models that directly learn mappings from context to predictive SPs. More specifically, ANPs [Kim et al., 2018] incorporate cross-attention modules to model the interaction between context and target. CONVNPs [Foong et al., 2020] produce a functional representation with a convolutional architecture which parameterizes a stationary predictive SP over the target, given the context. In GNPs [Bruinsma et al., 2021], predictive SPs are modelled by GPs where the mean/kernel functions are directly produced by neural networks conditioned on the context. However, simply setting the context to an empty set in these models does not yield more expressive generative SP models. In the absence of context, ANPs and GNPs become standard NPs and GPs respectively, without any additional expressivity. CONVNPs can indeed be adjusted for generative SP models where the latent variables are *random functions*. However, it remains unclear how to perform Bayesian inference in function space [Foong et al., 2020]. Conditional NP families, e.g. CNPs [Garnelo et al., 2018a], Convolutional Conditional Neural Processes (CONVCNPs) [Gordon et al., 2019] are another category of predictive SP models, which make a strong assumption that all targets are independent given the context. Recently, Neural Diffusion Processes (NDPs) [Dutordoir et al., 2022] provide an alternative to the NP models above, showing promising predictive performance. However, both marginal and conditional consistency are sacrificed in NDPs.

4.5 Experiments

Our experiments aim to answer the questions: 1) Can the proposed framework of MNPs offer better expressivity than NPs? 2) Compared to Bayesian nonparametric models, do neural parameterised SPs have advantages, especially on highly non-Gaussian data? 3) How well do MNPs perform on scientific problems? All experiments are performed using PyTorch [Paszke et al., 2019b]. For details about datasets, please see Section 4.7.3. For additional experimental details such as hyperparameters and architectures, please refer to Section 4.7.3.1.

The experiments below demonstrate that our extension to the NP framework indeed enhances expressivity by generalizing the density functions of NP. However, combining attention or convolutions with the MNP is non-trivial, so we leave that for future work. As a result, we did not include empirical comparisons with ANP, CONVNP etc. In this discussion, we do not claim having generalized density functions alone can lead to best-performing NP models.

4.5.1 1D function regression

We first consider 1D function regression problems to perform controlled comparisons between GPs, CBPs, NPs, and MNPs. Table 4.1 shows results across a range of datasets, including samples from the GP prior (we consider three different kernels) as well as more challenging non-GP data.

For datasets sampled from GPs, we also include the performance of oracle GPs with the right hyperparameters for generating the data. As shown, GPs and CBPs are close to the oracle except for samples generated using a periodic kernel, where CBPs struggle to learn the right kernel hyperparameters due to the difficulty of optimisation. For neural parameterised models, the performance gaps between the MNPs and the oracle GPs are much smaller than for NPs.

For non-GP samples such as monotonic functions, convex functions and samples from certain SDEs, MNPs perform the best across all models. CBPs obtain better marginal log-likelihood than GPs because the pointwise marginals are transformed using normalising flows, but their underlying copula processes (which capture the

Table 4.1: **Estimated marginal log-likelihood of SPs on 1D function regression problems**. We compare (a) Oracle models (when available). (b) GPs with optimised hyperparameters for additive kernels with three component kernels including an RBF kernel, a Matern kernel and a periodic kernel. (c) CBPs which combine Gaussian copula processes with neural spline flows [Durkan et al., 2019]. (d) NPs. (e) MNPs with 7 transition steps. Each experiment is repeated 5 times and we report the mean and standard errors of the marginal log-likelihood normalised by the number of points in the target. When latent models such as MNPs, NPs are used, we obtain estimations of marginal log-likelihoods on test data using the IWAE objective Burda et al. [2016] with 20 latent samples.

	Samples from GPs			***Non-GP Data***		
Model	RBF Kernel	Matern Kernel	Periodic Kernel	Monotonic	Convex	SDEs
Oracle	2.846±0.012	2.709±0.013	0.641±0.006	—	—	—
GPs	**2.844**±0.013	**2.708**±0.014	0.419±0.013	0.633±0.059	2.976±0.224	1.719±0.034
CBPs	2.628±0.016	2.604±0.015	0.169±0.022	1.776±0.088	4.268±0.035	1.842±0.024
NPs	0.935±0.019	1.115±0.021	0.356±0.020	1.823±0.006	1.956±0.004	1.621±0.009
MNPs	2.491±0.024	2.290±0.021	**0.594**±0.032	**2.755**±0.010	**5.582**±0.081	**1.942**±0.019

Figure 4.3: Wheel contextual bandits with varying exploration parameter δ. The optimal actions are $a_1, \ldots, a_5$ when the context point are in blue, yellow, red, green and black regions respectively.

dependence between data points) are still Gaussian, restricting their ability to model highly non-Gaussian data compared to neural parameterised models e.g. MNPs, NPs.

4.5.2 Contextual bandits

Contextual bandits are a class of problems where agents repeatedly choose from a set of actions based on context and receive rewards. The goal is then to learn a policy that maximizes expected cumulative reward over time. Contextual bandits find applications in real-time decision making problems such as resource allocation and online advertising.

Similarly to Garnelo et al. [2018c], we use the wheel bandit problem [Riquelme et al., 2018] (see Section 4.7.3) to evaluate our approach: a unit circle is partitioned into

Table 4.2: **Results on wheel contextual bandit problems**. We use an increasing value of δ, where more exploration is needed with higher δ. We report the mean and standard deviation of both cumulative and simple regrets (a performance measure of the final policy, estimated by the mean cumulative regrets in the last 500 steps) over 100 trials. Results are normalised by the performance of a uniform agent which chooses actions with equal probability.

δ	**0.5**	**0.7**	**0.9**	**0.95**	**0.99**
Cumulative regrets					
Uniform	100.0±0.00	100.0±0.00	100.0±0.00	100.0±0.00	100.0±0.00
MAML	2.95±0.12	3.11±0.16	4.84±0.22	7.01±0.33	22.93±1.57
NPs	1.60±0.06	1.75±0.05	3.31±0.10	5.71±0.24	22.13±1.23
MNPs	**1.08**±0.00	**1.23**±0.01	**2.10**±0.01	**2.07**±0.01	**5.46**±0.05
Simple regrets					
Uniform	100.0±0.00	100.0±0.00	100.0±0.00	100.0±0.00	100.0±0.00
MAML	2.49±0.12	3.00±0.35	4.75±0.48	7.10±0.77	22.89±1.41
NPs	**1.04**±0.06	**1.26**±0.21	2.90±0.35	5.45±0.47	21.45±1.3
C-BRUNO	1.32±0.06	1.43±0.07	3.44±0.13	6.17±0.21	21.52±0.41
MNPs	**1.14**±0.06	**1.29**±0.08	**2.12**±0.16	**2.24**±0.16	**5.22**±0.38

5 regions (see Figure 4.3) whose sizes are controlled by an exploration parameter δ. There are 5 actions a_1, a_2, a_3, a_4, a_5, and their associated reward depend on a 2D contextual coordinate $X = (X_1, X_2)$ uniformly sampled from within the circle. If $||X|| \leq \delta$, a_1 is the optimal action with reward sampled from $\mathcal{N}(1.2, 0.01^2)$, and taking any other action would yield a reward $r \sim \mathcal{N}(1.0, 0.01^2)$. If $||X|| > \delta$, the optimal action depends on which of the remaining 4 region X falls into and choosing it would yield a reward $r \sim \mathcal{N}(50, 0.01^2)$. Under this circumstance, any other action yield a reward $\mathcal{N}(1.0, 0.01^2)$ except that a_1 receives $r \sim \mathcal{N}(1.2, 0.01^2)$. We follow the experimental setup from Garnelo et al. [2018c] and only include models with a pre-training procedure (see Section 4.7.3.1 for details). As can be seen in Table 4.2, MNPs significantly outperform baselines (taken from the results of Garnelo et al. [2018c]) across different exploration rates δ.

4.5.3 Geological inference

In geostatistics, one is often interested in inferring the geological structure of an area given only a sparse set of measurements. This problem has traditionally been tackled with variants of GP regression [Cressie, 1990] (often referred to as Kriging in this context). However, several geological patterns (e.g. fluvial patterns) are highly

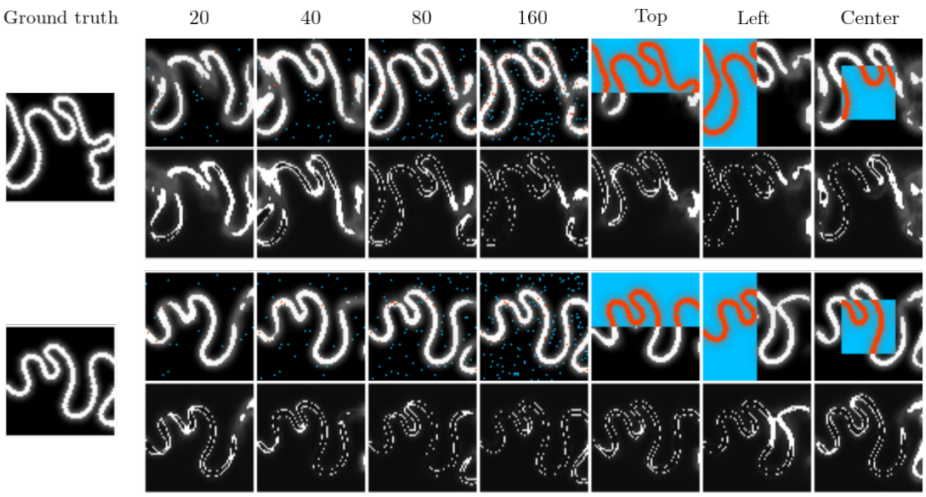


Figure 4.4: **Inferred geology conditioned on measurements**. Given a small number of physical measurements (red pixels indicate positive measurements while blue pixels indicate negative ones), we show the predictive mean (first row in each panel) and standard deviation (second row in each panel). Different columns correspond to different context sets. As can be seen, MNPs make predictions that are consistent with the data while showing larger uncertainty when fewer context points are available (or when a prediction is made far from a context point).

Table 4.3: **Test marginal log-likelihood on geology data**. Uniform context points are uniformly sampled from within the 2D square, while regional context includes the top/left half or the central square with half the size.

	Uniform Context				***Regional Context***		
$\lvert\mathcal{C}\rvert$	$N = 20$	40	80	160	Top	Left	Center
NPs	$-0.35_{\pm 0.30}$	$-0.31_{\pm 0.29}$	$-0.28_{\pm 0.27}$	$-0.25_{\pm 0.27}$	$-39.63_{\pm 304.08}$	$-22.03_{\pm 107.25}$	$-1.18_{\pm 4.20}$
MNPs	$\mathbf{0.67}_{\pm 0.31}$	$\mathbf{0.80}_{\pm 0.23}$	$\mathbf{0.98}_{\pm 0.18}$	$\mathbf{1.12}_{\pm 0.15}$	$\mathbf{0.44}_{\pm 0.59}$	$\mathbf{0.22}_{\pm 0.61}$	$\mathbf{0.65}_{\pm 0.39}$

complex and cannot be properly captured by these methods. To address this, several geostatistical models use a single training image to extract geological patterns and match these to the measurements [Strebelle, 2002, Zhang et al., 2006]. However, these approaches generally fail to produce realistic patterns capturing the variability of real geology.

More recently, deep learning approaches have been applied to this problem. Dupont et al. [2018], Zhang et al. [2019], Chan and Elsheikh [2019] train GANs on large geological datasets and use this as a prior for inferring geological structure given sparse measurements. However, the resulting models do not provide any meaningful uncertainty estimates, which are crucial for decision making in several applications. As MNPs provide uncertainty estimates they may therefore be a compelling approach

for this problem.

To evaluate MNPs on this problem, we introduce the GeoFluvial dataset, containing more than 25k simulations of fluvial patterns as 128×128 grayscale images. The dataset was generated using the open source `meanderpy` [Sylvester et al., 2019] package, which itself simulates the geological patterns as SPs (see Section 4.7.3 for details). We train our model on a training set of 20k simulations and evaluate it on a test set of 5k simulations. We test our trained model on a variety of context point configurations, corresponding to geological measurements taken at various spatial locations. Specifically, we consider uniformly sampled measurements (at 20, 40, 80, and 160 locations) as well as scenarios where measurements are spatially restricted to a certain area (top, left, and in the center of the square). Quantitative results are shown in Table 4.3 and qualitative results in Figure 4.4. As can be seen, our model achieves better likelihoods than NPs on this problem for all context point configurations, while also generating patterns that match the geological context. Further, as can be seen in Figure 4.4, the uncertainty estimates of the model are consistent with expectations, i.e. the model is more uncertain far from measurement locations or when there is ambiguity in the direction of the fluvial pattern.

4.6 Discussion

Limitations and future work Because we apply only a finite number of Markov transitions, the entire computational graph containing intermediate states is held in memory for backpropagation. To alleviate this, an interesting direction for future work would be to consider continuous-time Markov transitions, i.e. stochastic differential equations in function space, which require less memory by using adjoint methods [Chen et al., 2018a, Li et al., 2020]. While Equation (4.8) provides a valid general construction of exchangeable and consistent MMTOs with latent variables, it may be feasible to investigate other forms of MMTOs that do not necessitate latent variables, thereby eliminating the need for variational approaches.

There are often inherent symmetries in data (e.g. translational symmetries for stationary SPs) and it may be possible to improve empirical performance of MNPs using (group equivariant) convolutions that preserve certain symmetries [Gordon et al., 2019, Kawano et al., 2020, Foong et al., 2020]. These improvements are, however, orthogonal to our contributions and combining them is an exciting direction for future work.

Conclusions We have introduced Markov Neural Processes (MNPs), a framework for constructing flexible neural generative SP models. MNPs use neural-parameterized Markov transition operators on function spaces to gradually transform a simple initial SP into a flexible one. We proved that our proposed neural transitions preserve exchangeability and consistency necessary to define valid SPs. Empirical studies demonstrate that we can obtain expressive models of SPs with promising performance on contextual bandits and geological data.

4.7 Supplementary materials

4.7.1 Proofs

Proof of proposition 4.3.1 (See page 47):

Proposition 4.3.1. *Let $F_\theta(\cdot; x, z) : \mathcal{Y} \mapsto \mathcal{Y}$ denote an invertible transformation between outputs, parameterized by the input and latent. If $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ is a valid Stochastic Process (SP) (i.e. it satisfies Equation (4.1) and Equation (4.2)) and*

$$y^{(0)}_{1:n} \sim \{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}, \quad z \sim p_\theta(z), \quad y_i = F_\theta(y^{(0)}_i; x_i, z). \tag{4.5}$$

then $\{p_{x_{1:n}}(y_{1:n})\}_{x_{1:n}}$ is also a valid SP.

Proof. Given an input x_i and a latent variable z, y_i is obtained by applying an invertible transformation F_θ to $y^{(0)}_i$. Therefore, we can represent $p(y_i|y^{(0)}_i; x_i, z)$ using the Dirac delta:

$$p(y_i|y^{(0)}_i; x_i, z) = \delta(y_i - F_\theta(y^{(0)}_i; x_i, z)) \tag{4.15}$$

Furthermore, according to Equation (4.5), these transformations are independently applied given the latent variable z. So we have:

$$p(y_{1:n}|y^{(0)}_{1:n}; x_{1:n}, z) = \prod_{i=1}^{n} \delta(y_i - F_\theta(y^{(0)}_i; x_i, z)) \tag{4.16}$$

Hence

$$p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) = \int p_\theta(z) p_\theta(y_{1:n}|y^{(0)}_{1:n}; x_{1:n}, z) \mathrm{d}z \tag{4.17}$$

$$= \int p_\theta(z) \prod_{i=1}^{n} \delta(y_i - F_\theta(y^{(0)}_i; x_i, z)) \mathrm{d}z \tag{4.18}$$

□

As per Proposition 4.3.3, $\{p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\}_{x_{1:n}}$ is exchangeable and consistent. Furthermore, because $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ is a valid SP, $\{p_{x_{1:n}}(y_{1:n})\}_{x_{1:n}}$ is also a valid SP

4.7.1.1 Proof of proposition 4.3.2 (See page 48)

Proposition 4.3.2. *If the collection of marginals before transition $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ is consistent and exchangeable (as per Equations (4.1) and (4.2)) and the collection of marginal Markov transition operators (*MMTOs*) $\{p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\}_{x_{1:n}}$ is also consistent and exchangeable (as per Equations (4.6) and (4.7)), then the collection of marginals after transition $\{p_{x_{1:n}}(y_{1:n})\}$ is also consistent and exchangeable, hence defining a valid* SP.

Proof. According to the definitions of consistency and exchangeability for both the collection of marginals $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ and the collection of transition operators $\{p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\}_{x_{1:n}}$, we have the following (Equations (4.1), (4.2), (4.6) and (4.7)):

$$\begin{aligned}
\int p_{x_{1:n}}(y^{(0)}_{1:n})\mathrm{d}x_{m+1:n} &= p_{x_{1:m}}(y^{(0)}_{1:m}) \\
p_{\pi(x_{1:n})}(\pi(y^{(0)}_{1:n})) &= p_{x_{1:n}}(y^{(0)}_{1:n}) \\
\int p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\,\mathrm{d}y_{m+1:n} &= p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m}) \\
p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) &= p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y^{(0)}_{1:n}))
\end{aligned}$$

where $1 < m < n$.

The Markov transition on the function outputs are described by:

$$p_{x_{1:n}}(y_{1:n}) = \int p_{x_{1:n}}(y^{(0)}_{1:n})p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})dy^{(0)}_{1:n}$$

For any finite sequence $x_{1:n}$, we have

$$\begin{aligned}
\int p_{x_{1:n}}(y_{1:n})\mathrm{d}y_{m+1:n} &= \int \Big(\int p_{x_{1:n}}(y^{(0)}_{1:n}) p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\mathrm{d}y^{(0)}_{1:n} \Big)\mathrm{d}y_{m+1:n} \\
&= \int p_{x_{1:n}}(y^{(0)}_{1:n}) \Big(\int p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\mathrm{d}y_{m+1:n} \Big)\mathrm{d}y^{(0)}_{1:n} \\
&= \int p_{x_{1:n}}(y^{(0)}_{1:n}) p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m})\mathrm{d}y^{(0)}_{1:n} \\
&= \int \Big(\int p_{x_{1:n}}(y^{(0)}_{1:n})\mathrm{d}y^{(0)}_{m+1:n} \Big) p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m})\mathrm{d}y^{(0)}_{1:m} \\
&= \int p_{x_{1:m}}(y^{(0)}_{1:m}) p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m})\mathrm{d}y^{(0)}_{1:m} \\
&= \int p_{x_{1:m}}(y_{1:m})\mathrm{d}y^{(0)}_{1:m} \qquad (4.19)
\end{aligned}$$

Furthermore, we have

$$\begin{aligned}
p_{\pi(x_{1:n})}(\pi(y_{1:n})) &= \int p_{\pi(x_{1:n})}(\pi(y^{(0)}_{1:n})) p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y^{(0)}_{1:n}))\mathrm{d}\pi(y^{(0)}_{1:n}) \\
&= \int p_{x_{1:n}}(y^{(0)}_{1:n}) p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\mathrm{d}y^{(0)}_{1:n} \\
&= p_{x_{1:n}}(y_{1:n}) \qquad (4.20)
\end{aligned}$$

Therefore, the collection of marginals $\{p_{x_{1:n}}(y_{1:n})\}_{x_{1:n}}$ are both consistent and exchangeable (Equations (4.19) and (4.20)), hence defining a valid SP.

□

Proof of proposition 4.3.3 (See page 49):

Proposition 4.3.3. MMTOs *in the form of Equation* (4.8) *are consistent and exchangeable.*

Proof. Recall that MMTOs in Equation (4.8) write as:

$$p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n};\theta) = \int p_\theta(z) \prod_{i=1}^{n} \delta(y_i - F_\theta(y^{(0)}_i; x_i, z))\,\mathrm{d}z, \qquad (4.21)$$

It is a special case of a more general form:

$$p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n};\theta) = \int p_\theta(z) \prod_{i=1}^{n} p_\theta(y_i \,|\, y^{(0)}_i, x_i, z)\mathrm{d}z, \qquad (4.22)$$

Equation (4.22) becomes Equation (4.21) when $p_\theta(y_i \mid y_i^{(0)}, x_i, z) = \delta(y_i - F_\theta(y_i^{(0)}; x_i, z))$ is a δ-distribution. Below we prove that MMTOs are consistent and exchangeable for the general form. We have

$$\begin{aligned}
\int p_{x_{1:n}}(y_{1:n}|y_{1:n}^{(0)};\theta)\mathrm{d}y_{m+1:n} &= \int \Big(\int p_\theta(z) \prod_{i=1}^{n} p_\theta(y_i \mid y_i^{(0)}, x_i, z)\,\mathrm{d}z \Big)\,\mathrm{d}y_{m+1:n} \\
&= \int p_\theta(z) \prod_{i=1}^{m} p_\theta(y_i \mid y_i^{(0)}, x_i, z) \Big(\int \prod_{i=m+1}^{n} p_\theta(y_i \mid y_i^{(0)}, x_i, z)\mathrm{d}y_{m+1:n} \Big)\,\mathrm{d}z \\
&= \int p_\theta(z) \prod_{i=1}^{m} p_\theta(y_i \mid y_i^{(0)}, x_i, z)\,\mathrm{d}z \\
&= p_{x_{1:m}}(y_{1:m}|y_{1:m}^{(0)};\theta) \qquad (4.23)
\end{aligned}$$

and

$$\begin{aligned}
p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y_{1:n}^{(0)});\theta) &= \int p_\theta(z) \prod_{i=\pi(1)}^{\pi(n)} p_\theta(y_i \mid y_i^{(0)}, x_i, z)\mathrm{d}z \\
&= \int p_\theta(z) \prod_{i=1}^{n} p_\theta(y_i \mid y_i^{(0)}, x_i, z)\mathrm{d}z \\
&= p_{x_{1:n}}(y_{1:n}|y_{1:n}^{(0)};\theta) \qquad (4.24)
\end{aligned}$$

Therefore, the collection $\{p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y_{1:n}^{(0)});\theta)\}_{x_{1:n}}$ is both consistent (Equation (4.23)) and exchangeable (Equation (4.24)). □

Derivation of Markov Neural Process marginal densities:

T step Markov Neural Processes (MNPs) have marginal densities in the following form (as in Equation (4.9)):

$$\begin{aligned}
p_{x_{1:n}}(y_{1:n};\theta) &= \int p_\theta(z^{(1:T)}) p_{x_{1:n}}(y_{1:n}^{(0)}) \prod_{t=1}^{T}\prod_{i=1}^{n} p_\theta^{(t)}(y_i^{(t)} \mid y_i^{(t-1)}, x_i, z^{(t)})\mathrm{d}y_{1:n}^{(0:T-1)}\mathrm{d}z^{(1:T)} \\
&= \int p_\theta(z^{(1:T)}) p_{x_{1:n}}(y_{1:n}^{(0)}) \prod_{t=1}^{T}\prod_{i=1}^{n} \Big|\det \frac{\partial F_\theta^{(t)}(y_i^{(t-1)}; x_i, z^{(t)})}{\partial y_i^{(t-1)}}\Big|\mathrm{d}z^{(1:T)}
\end{aligned}$$

Proof. Given the sequence of latent variables $z^{(1:n)}$, our model becomes normalising flows on the finite sequence of function outputs $y_{1:n}$, with a prior distribution $p_{x_{1:n}}(y_{1:n}^{(0)})$, and the invertible transformation at each step $F_\theta^{(t)}(\cdot; x_i, z^{(t)})$. According to Rezende

and Mohamed [2015], Papamakarios et al. [2019], we have

$$\begin{aligned} p_{x_{1:n}}(y_{1:n}|z^{(1:T)};\theta) &= p_{x_{1:n}}(y_{1:n}^{(T)}|z^{(1:T)};\theta) \\ &= p_{x_{1:n}}(y_{1:n}^{(T-1)}|z^{(1:T-1)};\theta)\prod_{i=1}^{n}\left|\det\frac{\partial F_\theta^{(T)}(y_i^{(T-1)};x_i,z^{(T)})}{\partial y_i^{(T-1)}}\right| \\ &= \cdots \\ &= p_{x_{1:n}}(y_{1:n}^{(0)})\prod_{t=1}^{T}\prod_{i=1}^{n}\left|\det\frac{\partial F_\theta^{(t)}(y_i^{(t-1)};x_i,z^{(t)})}{\partial y_i^{(t-1)}}\right| \end{aligned}$$

The marginal density $p_{x_{1:n}}(y_{1:n};\theta)$ can be computed as:

$$\begin{aligned} p_{x_{1:n}}(y_{1:n};\theta) &= \int p_\theta(z^{(1:T)})p_{x_{1:n}}(y_{1:n}|z^{(1:T)};\theta)\mathrm{d}z^{(1:T)} \\ &= \int p_\theta(z^{(1:T)};\theta)p_{x_{1:n}}(y_{1:n}^{(0)})\prod_{t=1}^{T}\prod_{i=1}^{n}\left|\det\frac{\partial F_\theta^{(t)}(y_i^{(t-1)};x_i,z^{(t)})}{\partial y_i^{(t-1)}}\right|\mathrm{d}z^{(1:T)} \end{aligned} \tag{4.25}$$

□

4.7.2 Implementation details

4.7.3 Data

Gaussian Process (GP) samples Three 1D function datasets were created, each comprising samples from Gaussian processes (GPs) with different kernel functions: RBF (length scale 0.25), Matern-2.5 (length scale 0.5), and Exp-Sine-Squared (length scale 0.5 and periodicity 0.5). Observation noise variances were set at 0.0001 for RBF and Matern kernels, and 0.001 for the Exp-Sine-Squared kernel. Function inputs spanned from -2.0 to 2.0. To accelerate sampling, identical input locations were employed for every 20 function instances. For this dataset, context size varies randomly from 2 to 50.

Monotonic functions Our generation of monotonic functions starts by sampling $N \sim \text{Poisson}(5.0)$ to determine the number of interpolation nodes. We then sample $N+1$ increments $X_{\text{increments}}$ sampled from a Dirichlet distribution. These increments are increased by 0.01 to avoid excessively small values, and are then normalized such that their sum is 4.0. The final X values for interpolation nodes are obtained by adding -2.0 to the cumulative sum of these increments so that these X values are within

the range $[-2.0, 2.0]$. For each X value, a corresponding Y value is sampled from a Gamma distribution $Y \sim \text{Gamma}(2, 1)$. The cumulative sum of Y values ensures monotonicity. A PCHIP interpolator [Fritsch and Butland, 1984] is then created using these interpolation nodes (X and Y values) to generate function outputs. Given the functions, we randomly sample 128 X values and compute their corresponding function values. Note that these X values are now used to evaluate the functions, rather than serving as interpolation nodes. The function values are normalized to the range $[-1.0, 1.0]$. Finally, Gaussian observation noise with a standard deviation of 0.01 is added to these function values. For this dataset, context size varies randomly from 2 to 20.

Convex functions To create a dataset of convex functions, we compute integrals of the monotonic functions previously created. These convex functions are then randomly shifted and rescaled to increase diversity. The function values are normalized to the range $[-1.0, 1.0]$. Finally, Gaussian observation noise with a standard deviation of 0.01 is added to these function values. Context sizes varied randomly from 2 to 20.

Stochastic differential equations samples We create a dataset of 1D functions, each of which represents a solution to a Stochastic Differential Equation (SDE). This SDE is defined by the drift function $f(x, t) = -(a + x \cdot b^2) \cdot (1 - x^2)$ and the diffusion function $g(x, t) = b \cdot (1 - x^2)$, with constants a and b both set to 0.1. The function sets up a time span that includes 128 uniformly distributed points within the range of $[-5.0, 5.0]$. We then uniformly sample an initial condition, x_0, between 0.2 and 0.6. We use the `sdeint.stratKP2iS` function from the `sdeint` library to generate a solution to the SDE. This solution forms a 1D function that depicts a trajectory of the SDE across the defined time span, originating from the initial condition x_0. Lastly, we randomly alter the context sizes between 2 and 50.

For all the aforementioned datasets, we use the following set sizes: 50000 for the training set, 5000 for the validation set, and 5000 for the test set.

Geological data We generate the GeoFluvial dataset using the `meanderpy` [Sylvester et al., 2019] package. We first run simulations using the numerical model of meandering in `meanderpy` with default parameters except for channel depth which we change from 16m to 6m. The resulting simulations correspond to images of shape (800, 4000). We then extract 3 random non-overlapping crops of shape (700, 700) from these images,

which are resized to (128, 128) and are used as data. We ran $\lceil 25000/3 \rceil$ simulations, resulting in a dataset of 25,000 images.

4.7.3.1 Model architectures and hyperparameters

Permutation equivariant/invariant neural networks on sets We implement two versions of neural modules which operate on sets, both of which preserve the permutation symmetry of the sets. They are known as deep sets [Zaheer et al., 2017] and set transformers [Lee et al., 2019a]. To obtain an invariant representation, we used sum pooling for deep sets, and pooling by multi-head attention (PMA) [Lee et al., 2019a] for set transformers. Our experiments primarily employed set transformers, chosen for their stronger ability to model interactions between datapoints. However, for the wheel contextual bandit experiments, the context set often expanded to tens of thousands. To circumvent memory issues, we use deep sets in this instance. For set transformers, we stack two layers of set attention blocks (SABs) with a hidden dimension of 64 and 4 heads. This is followed by a single layer of PMA, which was subsequently followed by a linear map. In the case of using deep sets, we use a shared instance-wise Multi-Layer Perceptron (MLP) that has two hidden layers and a hidden dimension of 64. This MLP processes the concatenation of function inputs and outputs. Following this, we add a sum aggregation which is then succeeded by a single-layer MLP with ReLU (Rectified Linear Unit) activation.

Conditional normalising flows The instance-wise invertible transformation $F_\theta^{(t)}$ at each time step t is parameterised as a rational quadratic spline flow [Durkan et al., 2019]. Note that we do not share parameters among iterations. To condition on an input x_i and a latent variable z, we use a Multi-Layer Perceptron (MLP) which takes in x_i, z and produces the parameters for configuring a one-layer spline flow with 10 bins.

Multi-Layer Perceptrons (MLPs) Except for the deep sets, we use MLPs to parameterise continuous functions in two places. Firstly, we use it as a conditioning component in conditional normalising flows $F_\theta^{(t)}$ (as we mentioned above). It has two hidden layers and a hidden dimension of 128. Secondly, we use it to parameterise $\mu_\phi^{(t)}$ and $\sigma_\phi^{(t)}$ in the inference model (see Section 4.3.3). It has two hidden layers and a hidden dimension of 64.

We adopt the same pretraining approach as Garnelo et al. [2018c] for the contextual bandits problem, pretraining the model on 64 wheel problems $\{\delta_i\}_{i=1}^{64}$, where $\delta_i \sim \mathcal{U}(0, 1)$. Each wheel δ_i has a context size ranging from 10 to 512 and a target size varying between 1 and 50. Here each data point is a tuple (X, a, r). Because the context size can grow to tens of thousands during test time, for computational efficiency, we use deep sets to implement SetEncoder and a linear flow rather than a spline flow to parameterise $F_\theta^{(t)}$.

For optimisation, we use Adam [Kingma and Ba, 2014] optimiser with a learning rate of 0.0001. We use a batch size of 100 for 1D synthetic data and a batch size of 20 for the geological data. In experiments, we find that it is often beneficial to encode x_i with Fourier features [Tancik et al., 2020], and we use 80 frequencies randomly sampled from a standard normal.

4.7.3.2 Computational costs and resources

In our current implementation,the invertible transformations $F_\theta^{(t)}$ in the generative model and the mean/variance function $\mu_\phi^{(t)}\sigma_\phi^{(t)}$ in the inference model do not share parameters across iterations. However, we do share the SetEncoder across iterations. Consequently, with our MNP, both memory usage and computing time increase linearly with the number of steps. If the SetEncoder employs set transformers, the computational cost becomes $O(m^2)$, where m stands for the context size. This cost, however, can be decreased to $O(mk)$ by replacing set attention blocks (SABs) with induced set attention blocks (ISABs), where k denotes the number of inducing points. If deep sets are used to implement SetEncoder, the computational cost reduces to $O(m)$, despite the inference model being less expressive.

Training MNP is indeed resource-intensive; however, inference in MNP simply requires a forward pass. On a single GeForce GTX 1080 GPU card, a standard 7-step MNP takes approximately one day to train for $200k$ steps on 1D functions. If we use 20 latent samples to evaluate the marginal log-likelihood using the IWAE objective Burda et al. [2016], inference runs typically in a few seconds for a batch of 100 functions.

4.7.4 Broader impacts

Our work presents MNPs, a novel approach to construct SP using neural parameterised Markov transition operators. The broader impacts of this work have many aspects. Firstly, the proposed models are more flexible and expressive than traditional SP models and Neural Processes (NPs), enabling them to handle more complex patterns that arise in many applications. Moreover, the exchangeability and consistency in MNPs could improve the robustness and reliability of neural SP models. This could lead to more trustworthy systems, which is a critical aspect in high-stakes applications. However, as with any machine learning system, there are potential risks. For example, the complexity of these models could exacerbate issues related to interpretability and transparency, making it more difficult for humans to understand and control their behaviour.

Chapter 5

Group Equivariant Subsampling

5.1 Introduction

Convolutional Neural Networks (CNNs) are known to be more data efficient and show better generalisation on perceptual tasks than fully-connected networks, due to translation equivariance encoded in the convolutions: when the input image/feature map is translated, the output feature map also translates by the same amount. In typical CNNs, convolutions are used in conjunction with subsampling operations, in the form of pooling or strided convolutions, to reduce the spatial dimensions of feature maps and to allow receptive field to grow exponentially with depth. Subsampling/upsampling operations are especially necessary for convolutional autoencoders (ConvAEs) [Masci et al., 2011] because they allow efficient dimensionality reduction. However, it is known that subsampling operations implicit in strided convolutions or pooling layers are *not* translation equivariant [Zhang, 2019], hence CNNs that use these components are also not translation invariant. Therefore such CNNs and ConvAEs are not guaranteed to generalise to arbitrarily translated inputs despite their convolutional layers being translation equivariant.

Previous work, such as Zhang [2019], Chaman and Dokmanić [2020], has investigated how to enforce translation invariance on CNNs, but does not study equivariance with respect to symmetries beyond translations, such as rotations or reflections. In this work, we first describe subsampling/upsampling operations that preserve exact translation equivariance. The main idea is to sample feature maps on an input-dependent grid rather than a fixed one as in pooling or strided convolutions,

and the grid is chosen according to a *sampling index* computed from the inputs (see Figure 5.1). Simply replacing the subsampling/upsampling in standard CNNs with such translation equivariant subsampling/upsampling operations leads to CNNs and transposed CNNs that can map between spatial inputs and low-dimensional representations in a translation equivariant manner.

We further generalise the proposed subsampling/upsampling operations from translations to arbitrary groups, proposing *group equivariant subsampling/upsampling*. In particular we identify subsampling as mapping features on groups G to features on subgroups K (vice versa for upsampling), and identify the sampling index as a coset in the quotient space G/K. We note that group equivariant subsampling is different to *coset pooling* introduced in Cohen and Welling [2016], which instead gives features on the quotient space G/K, and discuss differences in detail in Section 5.4. Similar to the translation equivariant subsampling/upsampling, group equivariant subsampling/upsampling can be used with group equivariant convolutions to produce group equivariant CNNs. Using such group equvariant CNNs we can construct group equivariant autoencoders (GAEs) that separate representations into an invariant part and an equivariant part.

While there is a growing body of literature on group equivariant CNNs (G-CNNs) [Cohen and Welling, 2016, 2017, Worrall et al., 2017, Weiler et al., 2018b,a, Thomas et al., 2018, Weiler and Cesa, 2019b], such equivariant convolutions usually preserve the spatial dimensions of the inputs (or lift them to even higher dimensions) until the final invariant pooling layer. There is a lack of exploration on how to reduce the spatial dimensions of such feature maps while preserving exact equivariance, to produce low-dimensional equivariant representations. This work attempts to fill in this gap. Such low-dimensional equivariant representations can be employed in representation learning methods, allowing various advantages such as interpretability, out-of-distribution generalisation, and better sample complexity. When using such learned representations in downstream tasks such as abstract reasoning, reinforcement learning, video modelling, scene understanding, it is especially important for representations to be equivariant rather than invariant in these tasks, because transformations and how they act on feature spaces are critical information, rather than nuisance as in image classification problems.

In summary, we make the following contributions: (i) We propose subsampling/upsampling operations that preserve translational equivariance. (ii) We generalise the proposed subsampling/upsampling operations to arbitrary symmetry groups. (iii) We use

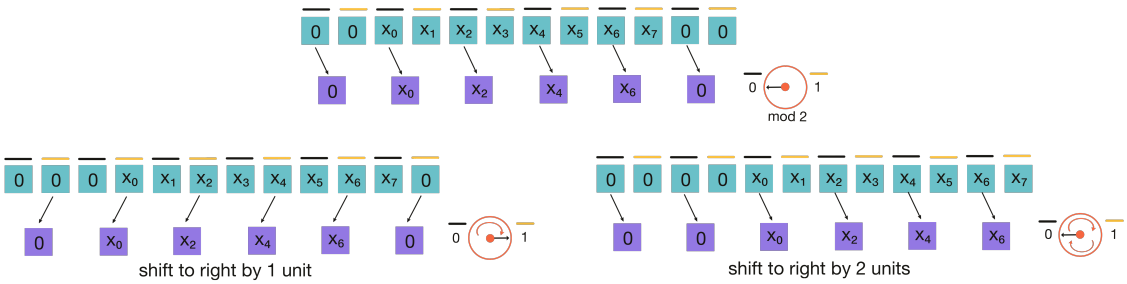


Figure 5.1: Equivariant subsampling on 1D feature maps with a scale factor $c = 2$. The input feature map has length 8, and initially we sample from odd positions determined by Equation (5.1) (top). When the original feature map is shifted to the right by 1 unit (bottom left), the sampling index becomes 1, so we instead sample from even positions. When the feature map is shifted to the right by 2 units (bottom right), we again sample from odd positions, but the outputs have been shifted to the right by 1 unit correspondingly.

equivariant subsampling/upsampling operations to construct GAEs that gives low-dimensional equivariant representations. (iv) We empirically show that representations learned by GAEs enjoys many advantages such as interpretability, out-of-distribution generalisation, and better sample complexity.

5.2 Equivariant subsampling and upsampling

5.2.1 Translation equivariant subsampling for CNNs

In this section we describe the proposed translation equivariant subsampling scheme for feature maps in standard CNNs. Later in Section 5.2.2, we describe how this can be generalised to group equivariant subsampling for feature maps on arbitrary groups.

Standard subsampling Feature maps in CNNs can be seen as functions defined on the integer grid, e.g. $\mathbb{Z}$ for 1D feature maps, and $\mathbb{Z}^2$ for 2D. Hence we represent feature maps as $f : \mathbb{Z} \to R^d$, where d is the number of feature map channels. For simplicity, we start with 1D and move on to the 2D case. Typically, subsampling in CNNs is implemented as either strided convolution or (max) pooling, and they can be

decomposed as

$$\begin{aligned}\text{CONV}_k^c &= \text{SUBSAMPLING}^c \circ \text{CONV}_k^1 \\ \text{MAXPOOL}_k^c &= \text{SUBSAMPLING}^c \circ \text{MAXPOOL}_k^1\end{aligned}$$

where subscripts denote kernel sizes and superscripts indicate strides. $c \in \mathbb{N}$ is the scale factor for SUBSAMPLING, and this operation simply restricts the input domain of the feature map from $\mathbb{Z}$ to $c\mathbb{Z}$, without changing the corresponding function values.

Translation equivariant subsampling In our equivariant subsampling scheme, we instead restrict the input domain to $c\mathbb{Z} + i$, the integers $\equiv i \mod c$, where i is a sampling index determined by the input feature map. The key idea is to choose i such that it is shifts by $t (\mathrm{mod} c)$ when the input is translated by t, to ensure that the same features are subsampled upon translation. Let i be given by the mapping $\Phi_c : \mathcal{I}_\mathbb{Z} \to \mathbb{Z}/c\mathbb{Z}$. $\mathcal{I}_\mathbb{Z}$ denotes the space of vector functions on $\mathbb{Z}$ and $\mathbb{Z}/c\mathbb{Z}$ is the space of remainders upon division by c.

$$i = \Phi_c(f) = \mathrm{mod}(\arg\max_{x \in \mathbb{Z}} \|f(x)\|_1, c) \tag{5.1}$$

where $\|\cdot\|_1$ denotes L^1-norm (other choices of norm are equally valid). Other choices for Φ_c are equally valid as long as they satisfy translation equivariance, ensuring that the same features are subsampled upon translation of the input:

$$\Phi_c(f(\cdot - t)) = \mathrm{mod}(\Phi_c(f) + t, c). \tag{5.2}$$

Note that this holds for Equation (5.1) provided the argmax is unique, which we assume for now (see Section 5.7.1.1 for a discussion of the non-unique case). We can decompose the subsampled feature map defined on $c\mathbb{Z} + i$ into its values and the offset index i, expressing it as $[f_b, i] \in (\mathcal{I}_{c\mathbb{Z}}, \mathbb{Z}/c\mathbb{Z})$, where f_b is the translated output feature map such that $f_b(cx) = f(cx + i)$ for $x \in \mathbb{Z}$.

The subsampling operation described above, which maps from $\mathcal{I}_\mathbb{Z}$ to $(\mathcal{I}_{c\mathbb{Z}}, \mathbb{Z}/c\mathbb{Z})$ is translation equivariant: when the feature map f is translated to the right by $t \in \mathbb{Z}$, one can verify that f_b will be translated to the right by $c\lfloor \frac{i+t}{c} \rfloor$, and the sampling index for the translated inputs would become $\mathrm{mod}(i + t, c)$. We provide an illustration for $c = 2$ in Figure 5.1, and describe formal statements and proofs later for the general cases in Section 5.2.2.

Multi-layer case For the subsequent layers, the feature map f_b is fed into the next convolution, and the sampling index i is appended to a list of outputs. When the above translation equivariant subsampling scheme is interleaved with convolutions in this way, we obtain an exactly translation equivariant CNN, where each subsampling layer with scale factor c_k produces a sampling index $i_k \in \mathbb{Z}/c_k\mathbb{Z}$. Hence the equivariant representation output by the CNN with L subsampling layers is a final feature map f_L and a L-tuple of sampling indices $(i_1, \ldots, i_L)$. This tuple can in fact be expressed equivalently as a single integer by treating the tuple as mixed radix notation and converting to decimal notation. We provide details of this multi-layer case in Section 5.7.1.2, including a rigorous formulation and its equivariance properties.

Translation equivariant upsampling As a counterpart to subsampling, upsampling operations increase the spatial dimensions of feature maps. We propose an equivariant upsampling operation that takes in a feature map $f \in \mathcal{I}_{c\mathbb{Z}}$ and a sampling index $i \in \mathbb{Z}/c\mathbb{Z}$, and outputs a feature map $f_u \in \mathcal{I}_{\mathbb{Z}}$, where we set $f_u(cx + i) = f(cx)$ and $\mathbf{0}$ everywhere else. This works well enough in practice, although in conventional upsampling the output feature map is often a smooth interpolation of the input feature map. To achieve this with equivariant upsampling, we can additionally apply average pooling with stride 1 and kernel size > 1.

2D Translation equivariant subsampling When feature maps are $2D$, they can be represented as functions on $\mathbb{Z}^2$. The sampling index becomes a 2-element tuple given by:

$$(x^*, y^*) = \arg\max_{(x,y)\in\mathbb{Z}^2} \|f(x)\|_1$$
$$(i, j) = (\mathrm{mod}(x^*, c), \mathrm{mod}(y^*, c))$$

and we subsample feature maps by restricting the input domain to $c\mathbb{Z}^2 + (i, j)$. The multi-layer construction and upsampling is analogous to the 1D-case.

5.2.2 Group equivariant subsampling and upsampling

In this section, we propose group equivariant subsampling by starting off with the 1D-translation case in Section 5.2.1, and provide intuition for how each component of this special case generalises to arbitrary discrete groups G. We then proceed to

mathematically formulate group equivariant subsampling, and prove that it is indeed G-equivariant.

Feature maps on groups First recall that the feature maps for the 1D-translation case were defined as functions on $\mathbb{Z}$, or $f \in \mathcal{I}_{\mathbb{Z}}$ for short. To extend this to the general case, we consider feature maps f as functions on a group G, i.e. $f \in \mathcal{I}_G = \{f : G \to V\}$[1] where V is a vector space, as is done in e.g. group equivariant CNNs (G-CNNs) [Cohen and Welling, 2016]. Note that translating feature maps f on $\mathbb{Z}$ by displacement u is effectively defining a new feature map $f'(\cdot) = f(\cdot - u)$. In the general case, we say that the group action on the feature space is given by

$$[\pi(u)f](g) = f(u^{-1}g) \tag{5.3}$$

where π is a group representation describing how $u \in G$ acts on the feature space.

Recap: translation equivariant subsampling Recall that standard subsampling that occurs in pooling or strided convolutions for 1D translations amounts to restricting the domain of the feature map from $\mathbb{Z}$ to $c\mathbb{Z}$, whereas equivariant subsampling also produces a sampling index $i \in \mathbb{Z}/c\mathbb{Z}$, an integer mod c, and that this is equivalent to restricting the input domain to $c\mathbb{Z} + i$. i is given by the translation equivariant mapping $\Phi_c : \mathcal{I}_{\mathbb{Z}} \to \mathbb{Z}/c\mathbb{Z}$. We can translate the input domain back to $c\mathbb{Z}$, and represent the output of subsampling as $[f_b, i] \in (\mathcal{I}_{c\mathbb{Z}}, \mathbb{Z}/c\mathbb{Z})$, where f_b is the translated output feature map and $f_b(cx) = f(cx + i)$ for $x \in \mathbb{Z}$.

Group equivariant subsampling Similarly in the general case, for a feature map $f \in \mathcal{I}_G$, standard subsampling can be seen as restricting the domain from the group G to a subgroup K, whereas equivariant subsampling additionally produces a sampling index $pK \in G/K$, where the quotient space $G/K = \{gK : g \in G\}$ is the set of (left) *cosets* of K in G. Note that we have rewritten i as p to distinguish between the 1D translation case and the general group case. This is equivalent to restricting the f to the coset pK. The choice of the coset pK is given by equivariant map $\Phi : \mathcal{I}_G \to G/K$ (the action of G on G/K is given by $u(gK) = (ug)K$ for $u, g \in G$), such that $pK = \Phi(f)$. This restriction of f to pK can also be thought of as having

[1]This is not to be confused with the space of Mackey functions in, e.g., Cohen et al. [2019], and rather it is the space of unconstrained functions on G.

an output feature map f_b on K and choosing a coset representative element $\bar{p} \in pK$, such that $f_b(k) = f(\bar{p}k)$. This choice of coset representative is described by a function $s : G/K \to G$, such that $\bar{p} = s(pK)$. The function s is called a section and should satisfy $s(pK)K = pK$.

Now let us formulate subsampling and upsampling operations $S_b{\downarrow}_K^G$ and $S_u{\uparrow}_K^G$ mathematically and prove its G-equivariance. Let $\mathcal{I}_K = \{f : K \to V'\}$ be the space of feature map on K. $S_b{\downarrow}_K^G$ takes in a feature map $f \in \mathcal{I}_G$ and produces a feature map $f_b \in \mathcal{I}_K$ and a coset in G/K. In reverse, the upsampling operation $S_u{\uparrow}_K^G$ takes in a feature map in $\mathcal{I}_K$, a coset in G/K, and produces a feature map in $\mathcal{I}_G$. We use a section $s : G/K \to G$ to represent a coset with a representative element in G, and point out that equivariance holds for any choice of s.

Formally, given an equivariant map $\Phi : \mathcal{I}_G \to G/K$ (we will discuss how to construct such a map in Section 5.2.3), and a fixed section $s : G/K \to G$ such that $\bar{p} = s(pK)$, the subsampling operation $S_b{\downarrow}_K^G : \mathcal{I}_G \to \mathcal{I}_K \times G/K$ is defined as:

$$\begin{aligned} pK = \Phi(f), \quad f_b(k) = f(\bar{p}k) \text{ for } k \in K \\ [f_b, pK] = S_b{\downarrow}_K^G(f; \Phi), \end{aligned} \tag{5.4}$$

while the upsampling operation $S_u{\uparrow}_K^G : \mathcal{I}_K \times G/K \to \mathcal{I}_G$ is defined as:

$$\begin{aligned} f_u(g) = f(\bar{p}^{-1}g) \text{ if } g \in K \text{ else } \mathbf{0} \\ f_u = S_u{\uparrow}_K^G(f, pK). \end{aligned} \tag{5.5}$$

To make the output of the upsampling dense rather than sparse, one can apply arbitrary equivariant smoothing functions such as average pooling with stride 1 and kernel size > 1, to compensate for the fact that we extend with $\mathbf{0}$s rather than values close to their neighbours. In practice, we observe that upsampling without any smoothing function works well enough.

The statement on the equivariance of $S_b{\downarrow}_K^G$ and $S_u{\uparrow}_K^G$ requires we specify the action of G on the space $\mathcal{I}_K \times G/K$, which we denote as π'. For any $u \in G$,

$$\begin{aligned} p'K = upK, \quad f'_b = \pi(\bar{p}'^{-1}u\bar{p})f_b \\ [f'_b,\ p'K] = \pi'(u)[f_b,\ pK] \end{aligned} \tag{5.6}$$

Lemma 5.2.1. *π' defines a valid group action of G on the space $\mathcal{I}_K \times G/K$.*

We can now state the following equivariance property (See Section 5.7.3 for a proof):

Proposition 5.2.1. *If the action of group G on the space $\mathcal{I}_G$ and $\mathcal{I}_K \times G/K$ are specified by π, π' (as defined in Equations (5.3) and (5.6)), and $\Phi : \mathcal{I}_G \to G/K$ is an equivariant map, then the operations $S_b{\downarrow}_K^G$ and $S_u{\uparrow}_K^G$ as defined in Equations (5.4) and (5.5) are equivariant maps between $\mathcal{I}_G$ and $\mathcal{I}_K \times G/K$.*

5.2.3 Constructing Φ

We use the following simple construction of the equivariant mapping $\Phi : \mathcal{I}_G \to G/K$ for subsampling/upsampling operations, although any equivariant mapping would suffice. For an input feature map $f \in \mathcal{I}_G$, we define

$$pK = \Phi(f) := (\arg\max_{g \in G} \|f(g)\|_1)K \tag{5.7}$$

Provided that the argmax is unique, it is easy to show that $(up) \cdot K = \Phi(\pi(u)f)$, hence Φ is equivariant. In practice one can insert arbitrary equivariant layers to f before and after we take the norm $\|\cdot\|_1$ to avoid a non-unique argmax (see Section 5.7.4).

In theory, there could exist cases where the argmax is always non-unique. We provide a more complex construction of Φ that deals with this case in Section 5.7.1.1.

5.3 Application: Group equivariant autoencoders

Group equivariant autoencoders (GAEs) are composed of alternating G-convolutional layers and equivariant subsampling/upsampling operations for the encoder/decoder. One important property of GAEs is that the final subsampling layer of the encoder subsamples to a feature map defined on the trivial group $\{e\}$, outputting a vector (instead of a feature map) that is *invariant*. For the 1D-translation case, suppose the input to the final subsampling layer is a feature map f defined on $\mathbb{Z}$. Then the final layer produces an invariant vector $f_b(0) = f(i_L)$ where $i_L = \arg\max_{x \in \mathbb{Z}} \|f(x)\|_1$. Note that there is no scale factor c_L here. Intuitively we can think of this as setting the scale factor $c_L = \infty$. Hence the encoder of the GAE outputs a representation that is disentangled into an invariant part $z_{\text{inv}} = f_b(0)$ (the vector output by the final subsampling layer) and an equivariant part $z_{\text{eq}} = (i_1, ..., i_L)$.

For the general group case, instead of specifying scale factors as in Section 5.2.1, we specify a sequence of nested subgroups $G = G_0 \geq G_1 \geq \cdots \geq G_L = \{e\}$, where the feature map for layer l is defined on subgroup G_L. For example, for the $p4$ group $G = \mathbb{Z} \rtimes \mathsf{C}_4$, we can use the following sequence for subsampling: $\mathbb{Z} \rtimes \mathsf{C}_4 \geq 2\mathbb{Z} \rtimes \mathsf{C}_4 \geq 4\mathbb{Z} \rtimes \mathsf{C}_4 \geq 8\mathbb{Z} \rtimes \mathsf{C}_2 \geq \{e\}$. Note that for the final two layers of this example, we are subsampling translations and rotations jointly.

We lift the input defined on the homogeneous input space to $\mathcal{I}_G$, and treat $f_0 \in \mathcal{I}_G$ as inputs to the autoencoders. The group equivariant encoder ENC can be described as follows:

$$\begin{aligned} [f_l,\ p_l G_l] &= S_b{\downarrow}_{G_l}^{G_{l-1}}(\text{G-CNN}_{l-1}^{E}(f_{l-1}); \Phi_l) \\ [z_{\text{inv}}, z_{\text{eq}}] &= [f_L(e),\ (p_1 G_1, p_2 G_2, \ldots, p_L G_L)] \end{aligned} \tag{5.8}$$

where $l = 1, \ldots, L$ and $\text{G-CNN}_l(\cdot)$ denotes G-convolutional layers before the lth subsampling layer.

The decoder DEC simply goes in the opposite direction, and can be written formally as:

$$\begin{aligned} f_L \text{ is defined on } G_L = \{e\} \text{ and } f_L(e) &= z_{\text{inv}} \\ f_{l-1} = \text{G-CNN}_{l-1}^{D}(S_u{\uparrow}_{G_l}^{G_{l-1}}(f_l,\ p_l G_l)) & \end{aligned} \tag{5.9}$$

where $l = L, \ldots, 1$ and $\hat{f} = f_0$ gives the final reconstruction.

Recall from Section 5.2.1 that the tuple $(i_1, \ldots, i_L)$ can be expressed equivalently as a single integer. Similarly, the tuple $(p_1 G_1, p_2 G_2, \ldots, p_L G_L)$ can be expressed as a single group element in G. We show in Section 5.7.1.2 that the action implicitly defined on the tuple via Equation (5.6) simplifies elegantly to the left-action on the single group element in G.

We now have the following properties for the learned representations (see Section 5.7.3 for a proof):

Proposition 5.3.1. *When* ENC *and* DEC *are given by Equations* (5.8) *and* (5.9)*, and the group actions are specified as in Equation* (5.3) *and Equation* (5.6)*, for any $g \in G$ and $f \in \mathcal{I}_G$, we have*

$$\begin{aligned} [z_{inv}, g \cdot z_{eq}] &= \text{ENC}(\pi(g) f) \\ \pi(g)\hat{f} &= \text{DEC}(z_{inv}, g \cdot z_{eq}) \end{aligned}$$

5.4 Related work

Group equivariant neural networks The equivariant subsampling/upsampling that we propose deals with feature maps (functions) defined on the space of the group G or its subgroups K, which transform under the *regular representation* with the group action. Hence our equivariant subsampling/upsampling is compatible with *lifting-based* group equivariant neural networks defined on discrete groups [Cohen and Welling, 2016, Hoogeboom et al., 2018, Romero and Hoogendoorn, 2020, Romero et al., 2020] that define a mapping between feature maps on G. We also discuss the extension of group equivariant subsampling to be compatible with those defined on continuous/Lie groups [Cohen et al., 2018a, Esteves et al., 2018, Finzi et al., 2020, Bekkers, 2020, Hutchinson et al., 2021] in Section 5.6. This is in contrast to group equivariant neural networks that do not use lifting and use *irreducible representations*, defining mappings between feature maps on the input space $\mathbf{X}$. [Cohen and Welling, 2017, Worrall et al., 2017, Thomas et al., 2018, Kondor et al., 2018, Weiler et al., 2018b,a, Weiler and Cesa, 2019b, Esteves et al., 2020, Fuchs et al., 2020].

Coset pooling In particular, Cohen and Welling [2016] propose *coset pooling*, which is also a method for equivariant subsampling. Here a feature map f on G is mapped onto a feature map $\Phi(f)$ on G/K (as opposed to K, for our equivariant subsampling) as follows:

$$\Phi(f)(gK) = \text{POOL}_{k \in K} f(gk) \tag{5.10}$$

such that the feature values on the coset gK are pooled. For the 1D-translation case, where $G = \mathbb{Z}, K = c\mathbb{Z}$, this amounts to pooling over every cth pixel, which disrupts the locality of features as opposed to our equivariant subsampling that preserves locality, and hence is more suitable to use with convolutions for translation equivariance. See Figure 5.2 for a visual comparison. As such, the $p4$-CNNs in Cohen and Welling [2016] use standard max pooling with stride=2 rather than coset pooling for $\mathbb{Z}^2$, and coset-pooling is only used in the final layer to pool over feature maps across 90-degree rotations, achieving exact rotation equivariance but imperfect translation equivariance. In our work, we use translation equivariant subsampling in the earlier layers and rotation equivariant subsampling in the final layers to achieve exact roto-translation equivariance.

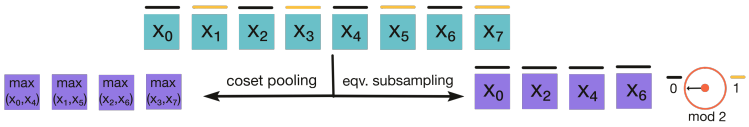


Figure 5.2: Coset (max) pooling vs. equivariant subsampling.

Unsupervised disentangling and object discovery GAEs produce *global* equivariant (z_{eq}) and invariant (z_{inv}) representations, effectively separating position and pose information with other semantic information in single-object images. This relates to unsupervised disentangling [Higgins et al., 2017, Chen et al., 2018b, Kim and Mnih, 2018, Zhao et al., 2017] where different factors of variation in the data are separated in different dimensions of a low-dimensional representation. However unlike equivariant subsampling, there is no guarantee of any equivariance in the low-dimensional representation, making the resulting disentangled representations less interpretable. Works on unsupervised object discovery [Burgess et al., 2019, Greff et al., 2019, Engelcke et al., 2020, Locatello et al., 2020] learn object-centric representations, and we showcase GAEs in MONet [Burgess et al., 2019] where we replace their VAE with a V-GAE in order to separate position and pose information and learn more interpretable representations of objects in a data-efficient manner.

Shift-invariance in CNNs As early as Simoncelli et al. [1992], it has been discussed that shift-invariance cannot hold for conventional subsampling. Although standard subsampling operations such as pooling or strided convolutions are not *exactly* shift invariant, they do not prevent strong performance on classification tasks [Scherer et al., 2010]. Nonetheless, Zhang [2019] integrates anti-aliasing to improve shift-invariance, showing that it leads to better performance and generalisation on classification tasks. Chaman and Dokmanić [2020] explore a similar strategy to our equivariant subsampling by partitioning feature maps into polyphase components and select the component with the highest norm. However, unlike the proposed group equivariant subsampling/upsampling which tackle general equivariance for arbitrary discrete groups, both works focus only on translation invariance.

5.5 Experiments

In this section, we compare the performance of GAEs with equivariant subsampling to their non-equivariant counterparts that use standard subsampling/upsampling in object-centric representation learning. We show that GAEs give rise to more interpretable representations that show better sample complexity and generalisation than their non-equivariant counterparts.

Models and baselines (G-)Convolutional autoencoders (G)ConvAE are composed of alternating (G-)convolutional layers and subsampling/upsampling operations with a final MLPs applied to the flattened feature maps. We categorize models by the types of equivariance preserved by the convolutional layers. We consider three different discrete symmetry groups: $p1$ (only translations), $p4$ (composition of translations and 90 degree rotations), $p4m$ (composition of translations, 90 degree rotations and mirror reflection). The baseline models are: ConvAE-$p1$ (standard convolutional autoencoders), GConvAE-$p4$, GConvAE-$p4m$, where the corresponding equivariance is preserved in the (G-)convolutional layers but not in the subsampling/upsampling operations. The equivariant counterparts of these baseline models are GAE-$p1$, GAE-$p4$, GAE-$p4m$, where the subsampling/upsampling operations are also equivariant. For baseline models, we use a scale factor of 2 for all subsampling/upsampling layers. For GAEs, we subsample first the translations, then rotations, followed by reflections, all with scale factor 2. e.g. for GAE-$p4m$, the feature maps at each layer are defined on the following chain of nested subgroups: $\mathbb{Z}^2 \rtimes (\mathsf{C}_4 \rtimes \mathsf{C}_2) \geq (2\mathbb{Z})^2 \rtimes (\mathsf{C}_4 \rtimes \mathsf{C}_2) \geq (4\mathbb{Z})^2 \rtimes (\mathsf{C}_4 \rtimes \mathsf{C}_2) \geq (8\mathbb{Z})^2 \rtimes (\mathsf{C}_4 \rtimes \mathsf{C}_2) \geq (16\mathbb{Z})^2 \rtimes (\mathsf{C}_2 \rtimes \mathsf{C}_2) \geq \{e\}$. As in Cohen and Welling [2016], we rescale the number of channels such that the total number of parameters of these models roughly match each other.

Training objectives For single-object autoencoders including ConvAEs and GAEs, we train these models to reconstruct the inputs by minimizing the Mean Squared Error (MSE) loss between the inputs and output reconstructions. For multiple-object scenarios, we follow the approach described in Burgess et al. [2019], which uses a variational encoder (VAE) loss.

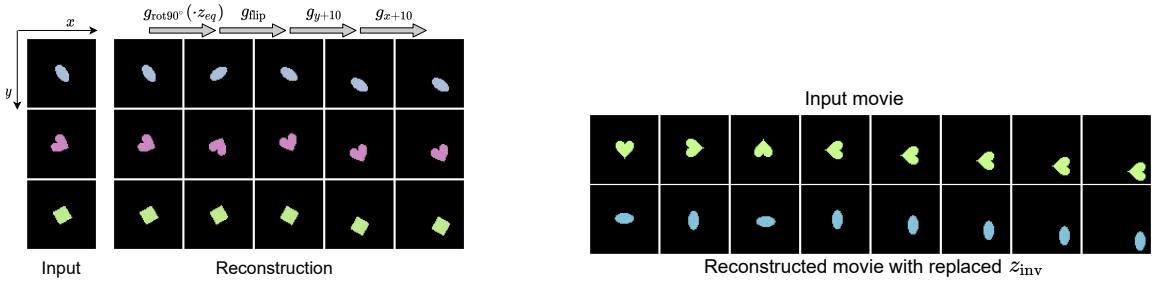


Figure 5.3: (Left) Manipulating reconstructions by modifying the equivariant part z_{eq}. The second column are the original reconstructions, which match the inputs well. The subsequent columns are reconstructions decoded from modified z_{eq}. We transform z_{eq} with a sequence of group elements, and show the resulting reconstructions. (Right) Manipulating reconstruction shape by modifying z_{inv}.

Data To demonstrate basic properties of GAEs and compare sample complexity under the single object scenario, we use Colored-dSprite [Matthey et al., 2017] and a modification of FashionMNIST [Xiao et al., 2017], where we first apply zero-padding to reach a size of 64×64, followed by random shifts, rotations and coloring. For multi-object datasets, we use Multi-dSprites [Kabra et al., 2019] and CLEVR6 which is a variant of CLEVR [Johnson et al., 2017] with up to 6 objects. All input images are resized to a resolution of 64×64.

See Section 5.7.4 and our reference implementation [2] for more details on hyperparameters and data preprocessing. Our implementation is built upon open source projects Harris et al. [2020], Paszke et al. [2019a], Yadan [2019], Weiler and Cesa [2019a], Engelcke et al. [2020], Hunter [2007], Waskom [2021].

5.5.1 Basic properties: Equivariance, disentanglement and out-of-distribution generalization

Equivariance The encoder-decoder pipeline in GAEs is exactly equivariant. In Figure 5.3, we train GAE-*p4m* on 6400 examples from Colored-dSprites, and we show how to manipulate reconstructions by manipulating the equivariant representation z_{eq} (left). If an image x is encoded into $[z_{\text{inv}}, z_{\text{eq}}]$, then decoding $[z_{\text{inv}}, g \cdot z_{\text{eq}}]$ will give $g \cdot \hat{x}$ where $\hat{x}$ is the reconstruction of x.

[2] https://github.com/jinxu06/gsubsampling

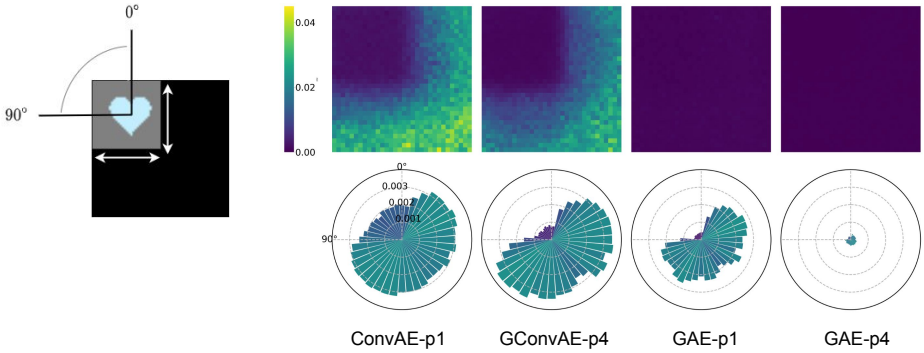


Figure 5.4: Generalisation to out-of-distribution object locations and poses. During training, we constrain shapes to be in the top-left quarter, and the orientation to be always less than 90 degrees. On the right, we compare the error of reconstructions of different models generalise on objects at unseen locations in the first row, and how they generalise to unseen orientations in the second row.

Disentanglement The learned representations in GAEs are disentangled into an invariant part z_{inv} and an equivariant part z_{eq}. In Figure 5.3 (left), we vary the equivariant part while the invariant part remains the same. In Figure 5.3 (right), we show the frames of a movie of a heart, and show its reconstruction after replacing z_{inv} representing a heart with that of an ellipse. Note that the ellipse shape undergoes the same sequence of transformations as the heart.

Out-of-distribution generalisation GAEs can generalise to data with unseen object locations and poses. We train an GAE-$p4$ on 6400 constrained training examples, where we only use examples with locations in the top-left quarter and orientations within $[0, 90]$ degrees, as shown in Figure 5.4. During test time, we evaluate mean squared error (MSE) of reconstructions on unfiltered test data to see how models generalise to unseen location and poses. Both ConvAE-$p1$ and GConvAE-$p4$ cannot generalise well to object poses out of their training distribution. In contrast, GAE-$p1$ generalise to any locations without performance degradation but not to unseen orientations, while GAE-$p4$, which encodes both translation and rotation equivariance, generalises well to all locations and orientations. We only use heart shapes for evaluation, because the square and ellipse have inherent symmetries.

5.5.2 Single object

Since GAEs are fully equivariant and can generalize to unseen object poses, it is natural to conjecture that such models can significantly improve data efficiency when

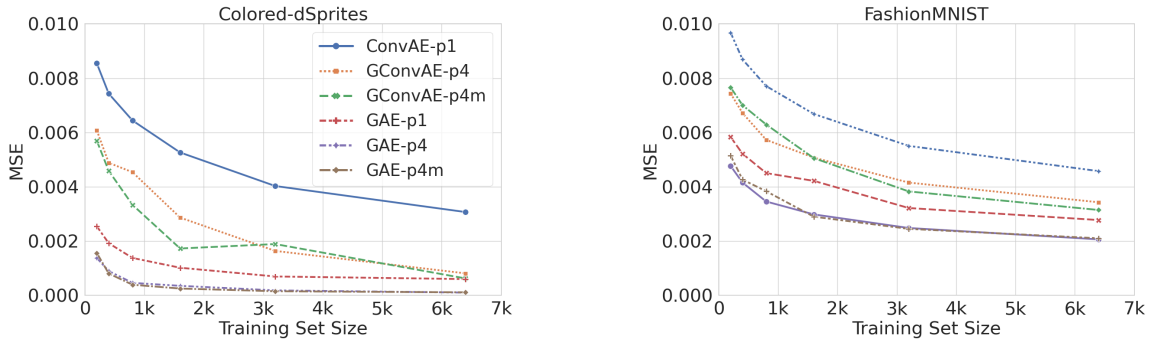


Figure 5.5: Reconstruction error on single object datasets

symmetry-transformed data points are also plausible samples from the data distribution. We test this hypothesis on Colored-dSprites and transformed FashionMNIST, and the results are shown in Figure 5.5. On both datasets, equivariant autoencoders significantly outperform their non-equivariant counterparts for all considered training set sizes. In fact, as shown in the figure, equivariant models trained with a smaller training set size is often comparable to baseline models trained on a larger training set. Furthermore, the results demonstrate that it is beneficial to consider symmetries beyond translations in these problems: for both non-equivariant and equivariant models, variants that encode rotation and reflection symmetries consistently show better performance compared to models that only consider the translation symmetry.

5.5.3 Multiple objects

In multi-object scenes, it is often more interesting to consider local symmetries associated with objects rather than the global symmetry for the whole image. To exploit object symmetries in image data, one needs to first discover objects and separate them from the background, which is a challenging problem on its own. Currently, GAEs do not have inherent capability to solve these problems. In order to investigate whether our models could improve data efficiency in multi-object settings, we rely on recent work on unsupervised object discovery and only use GAEs to model object components. More specifically, we explored replacing component VAEs in MONet [Burgess et al., 2019] with V-GAEs (probabilistic version of our GAEs, where a standard Gaussian prior is put on z_{inv} and z_{eq} remains deterministic), and train models end-to-end. Again we study the low data regime to show results on data efficiency.

We train models on Multi-dSprites and CLEVR6 with training set sizes 3200, 6400 and

Table 5.1: Reconstruction error MSE ($\times 10^{-3}$) (mean(stddev) across 5 seeds) on multi-object datasets

Dataset	Multi-dSprites			CLEVR6		
Training Set Size	3200	6400	12800	3200	6400	12800
MONet	2.661(0.382)	1.385(0.235)	0.326(0.076)	0.673(0.059)	0.562(0.057)	0.546(0.056)[1]
MONet-GAE-$p1$	0.659(0.103)	0.359(0.025)	0.264(0.042)	0.473(0.064)	0.432(0.052)	0.388(0.016)
MONet-GAE-$p4$	0.563(0.195)	0.317(0.060)	0.231(0.067)	0.461(0.025)	0.414(0.022)	0.413(0.018)

Table 5.2: Foreground segmentation performance in terms of ARI (mean(stddev) across 5 seeds)

Dataset	Multi-dSprites			CLEVR6		
Training Set Size	3200	6400	12800	3200	6400	12800
MONet	0.597(0.022)	0.747(0.049)	0.891(0.009)	0.829(0.055)	0.878(0.023)	0.865(0.033)[1]
MONet-GAE-$p1$	0.762(0.049)	0.823(0.042)	0.889(0.013)	0.921(0.015)	0.917(0.032)	0.920(0.025)
MONet-GAE-$p4$	0.753(0.089)	0.833(0.072)	0.902(0.025)	0.878(0.055)	0.914(0.012)	0.910(0.011)

[1] *We excluded 2 outliers here as the baseline MONet occasionally fails during late-phase training.*

12800. We consider two evaluation metrics: mean squared error (MSE) to measure the overall reconstruction quality, and adjusted rand index (ARI), which is a clustering similarity measure ranging from 0 (random) to 1 (perfect) to measure object segmentation. As in Burgess et al. [2019], we only use foreground pixels to compute ARI. Component VAEs in MONet use spatial broadcast decoders [Watters et al., 2019] that broadcast the latent representation to a full scale feature map before feeding them into the decoders, and the decoders therefore do not need upsampling. It has the implicit effect of encouraging the smoothness of the decoder outputs. To encourage similar behaviour, we add average pooling layers with stride 1 and kernel size 3 to our equivariant decoders. As shown in Table 5.1, using GAEs to model object components significantly improves reconstruction quality, which is consistent with our findings in single-object scenario. As shown in Table 5.2, using GAEs to model object components also leads to better object discovery in the low data regimes, but this advantage seems to diminish as the dataset becomes sufficiently large.

5.6 Conclusions, limitations and future work

Conclusions We have proposed subsampling/upsampling operations that *exactly* preserve translation equivariance, and generalised them to define *exact* group equiv-

ariant subsampling/upsampling for discrete groups. We have used these layers in GAEs that allow learning low-dimensional representations that can be used to reliably manipulate pose and position of objects, and further showed how GAEs can be used to improve data efficiency in multi-object representation learning models.

Limitations and future work Although the equivariance properties of subsampling layers also hold for Lie groups, we have not discussed the practical complexities that arise with the continuous case, where feature maps are only defined on a finite subset of the group rather than the whole group. We leave this as important future work, as well as application of equivariant subsampling for tasks other than representation learning where equivariance/invariance is desirable e.g. object classification, localization. Another limitation is that our work focuses on global equivariance, like most other works in the literature. An important direction is to extend to the case of local equivariances e.g. object-specific symmetries for multi-object scenes.

5.7 Supplementary materials

5.7.1 Equivariant subsampling and upsampling

5.7.1.1 Constructing Φ

In Section 5.2.3, we provide a simple construction of the equivariant map $\Phi : \mathcal{I}_G \to G/K$ which gives the sampling indexes. The construction is a valid one if the argmax is unique. In practice one can insert arbitrary equivariant layers to f before and after we take the norm $\|\cdot\|_1$ to avoid a non-unique argmax (see Section 5.7.4). In practice, inserting smoothing layers such as convolutions or pooling can also improve the stability of Φ when introducing noise, so that perturbation of the inputs does not dramatically change the output of the subsampling/upsampling layers. However, in theory, there could be cases that the argmax is always non-unique. We discuss this case below and provide a more complex construction for it.

One cannot avoid a non-unique argmax in Equation (5.7) when the input feature map $f \in \mathcal{I}_G$ has inherent symmetries, i.e. there exists $u \in G, u \neq e$, such that $f = \pi(u)f$. Assuming there is a unique argmax g^* such that $g^* = \arg\max_{g \in G} \|f(g)\|_1$, we would

have:

$$f(u \cdot g^*) = f(g^*) = \max_{g \in G} \|f(g)\|_1$$

Therefore $u \cdot g^*$ is also a valid argmax, hence the argmax is not unique. For example, when f is a feature map representing a center-aligned circle, we would have $f = \pi(u)f$, where $u \in O(2)$ is associated with an arbitrary rotation around the center. One cannot find a unique argmax g^* for this example, because the feature map would take the same function values at $u \cdot g^*$.

Under the circumstance described above, the argmax operation would return a set of elements where each one attains the function's largest values. We denote it as $S^* = \arg\max_{g \in G} \|f(g)\|_1$, where S^* is a subset of G. To obtain the sampling index (a coset) pK, we sample uniformly from the set S^*, and let Φ outputs pK where $p \sim S^*$. In this case, the map Φ is still equivariant in distribution even though it is now a stochastic map.

Note that it is possible to consider more sophisticated solutions or even use learnable modules for Φ, which we leave for future work.

5.7.1.2 Multiple subsampling layers

Translation equivariant subsampling We can stack convolutional and translation equivariant subsampling layers to construct exactly translation equivariant CNNs. Unlike standard CNNs, each translation equivariant subsampling layer with a scale factor c_k outputs a subsampling index i_k in addition to the feature maps. Hence the equivariant representation output by the CNN with L subsampling layers is a final feature map f_L and a L-tuple of sampling indices $(i_1, ..., i_L)$.

In the multi-layer case, the l-th subsampling layer takes in a feature map f on $\prod_{k=1}^{l-1} c_k\mathbb{Z}$ and outputs: 1) a feature map on $\prod_{k=1}^{l} c_k\mathbb{Z}$ and 2) a subsampling index $i_l \in \prod_{k=1}^{l-1} c_k\mathbb{Z} / \prod_{k=1}^{l} c_k\mathbb{Z} \cong \mathbb{Z}/c_l\mathbb{Z}$ given by:

$$(\prod_{k=1}^{l-1} c_k) \cdot i_l = p_l = \Phi_c(f) = \mathrm{mod}(\arg\max_{x \in (\prod_{k=1}^{l-1} c_k)\mathbb{Z}} \|f(x)\|_1, \prod_{k=1}^{l} c_k)$$

This is equivalent to treating the input feature map f as a feature map f' defined on $\mathbb{Z}$ (i.e. mapping the support of f from $\prod_{k=1}^{l-1} c_k\mathbb{Z}$ to $\mathbb{Z}$ via division by $\prod_{k=1}^{l-1} c_k$),

and the subsampling layer outputting: 1) a feature map on $c_l\mathbb{Z}$ and 2) a subsampling index $i_l \in \mathbb{Z}/c_l\mathbb{Z}$ given by:

$$i_l = \mathrm{mod}(\arg\max_{x\in\mathbb{Z}} \|f'(x)\|_1, c_l)$$

Hence the tuple $(i_1, ..., i_L)$ that contains the sampling indices of all layers can be expressed equivalently as a single integer:

$$r_{\text{eq}} = \sum_{l=1}^{L} p_l = \sum_{l=1}^{L} (\prod_{k=1}^{l-1} c_k) \cdot i_l$$

where $r_{\text{eq}} \in \mathbb{Z}/(\prod_{k=1}^{L} c_k)\mathbb{Z}$. Note that the conversion between r_{eq} and $(i_1, ..., i_L)$ can be seen as the conversion between *mixed radix notation* and decimal notation. Mixed radix notation is a mixed base numeral system where the numerical base varies from position to position, as opposed to base-n systems that have the same base for all positions[3]. Thus there is an one-to-one correspondence between the two. Moreover, when the input feature map is translated to the right by $t \in \mathbb{Z}$, r_{eq} would become $\mathrm{mod}(r_{\text{eq}} + t, \prod_{k=1}^{L} c_k)$. See the statement of this result for the general group case in Proposition 5.7.1 and its proof in Section 5.7.3.

Group equivariant subsampling Similarly, given an input feature map $f \in \mathcal{I}_G$, we can construct CNNs/G-CNNs with multiple equivariant subsampling layers by specifying a sequence of nested subgroups $G = G_0 \geq G_1 \geq \cdots \geq G_L$. The l-th subsampling layer takes in a feature map on G_{l-1}, outputs a feature map on G_l and a sampling index $p_l G_l \in G_{l-1}/G_l$. Formally, the l-th subsampling layer can be written as:

$$S_b{\downarrow}_{G_l}^{G_{l-1}} : \mathcal{I}_{G_{l-1}} \to \mathcal{I}_{G_l} \times G_{l-1}/G_l$$

The equivariant representation output by the CNNs/G-CNNs with L subsampling layers is a feature map in $f_L \in G_L$ and a L-tuple $(p_1G_1, p_2G_2, \ldots, p_LG_L)$.

Similar to the 1D translation case, the sampling index tuple $(p_1G_1, p_2G_2, \ldots, p_LG_L)$ can be expressed equivalently as a single element in the quotient space G/G_L:

$$r_{\text{eq}} = (\bar{p}_1\bar{p}_2 \ldots \bar{p}_L)G_L = \nu(p_1G_1, p_2G_2, \ldots, p_LG_L) \tag{5.11}$$

[3]A commonly used example of mixed radix notation is to express time, where e.g. 12:34:56 has a base of 24 for the hour digit, base 60 for the minute digit and base 60 for the second digit.

where $\bar{p}_l$ denote the coset representive for the quotient space G_{l-1}/G_l. ν is a bijection from r_{eq} to the tuple, whose inverse can be computed by the following recursive procedure:

$$\begin{aligned} p'_1 G_L &= r_{\text{eq}} \\ p'_l &= \bar{p}'^{-1}_{l-1} \cdot p'_{l-1} \\ (p'_1 G_1, p'_2 G_2, \dots, p'_L G_L) &= \nu^{-1}(r_{\text{eq}}) \end{aligned} \tag{5.12}$$

Proposition 5.7.1. *ν^{-1} is the inverse of ν, hence ν is bijective. And $\forall u \in G$ we have:*

$$u \cdot \nu(p_1 G_1, p_2 G_2, \dots, p_L G_L) = \nu(u \cdot (p_1 G_1, p_2 G_2, \dots, p_L G_L)).$$

5.7.2 Group equivariant autoencoders

In Section 5.7.1.2 we discussed that we can stack multiple subsampling layers by specifying a sequence of nested groups $G = G_0 \geq G_1 \geq \cdots \geq G_L$, and the CNN/G-CNNs with L subsampling layers would produce a feature map on G_L and a tuple $z_{\text{eq}} = (p_1 G_1, p_2 G_2, \dots, p_L G_L)$. Furthermore, we know from Proposition 5.7.1 that there is an one-to-one correspondence between the tuple representation z_{eq} and the single group element representation $r_{\text{eq}} = \nu(z_{\text{eq}}) \in G/G_L$. For group equivariant autoencoders, we specify a sequence of subgroups but with $G_L = \{e\}$. In this case, r_{eq} would simply become a group element in G. And the group action simplifies to left-multiplying the corresponding group elements.

Although one can simply use the tuple output by the encoder to perform upsampling in the decoder (and hence use the same sequence of nested subgroups), this is not strictly necessary as one can use a different sequence of nested subgroups for the decoder and obtain the tuple using the decomposition procedure in Equation (5.12). Moreover, for more efficient implementation of GAEs, one does not need to pass through Φ in Equation (5.7) for every subsampling layer. It would suffice to obtain the tuple of subsampling indexes from the first subsampling layer using:

$$(p_1 G_1, p_2 G_2, \dots, p_L G_L) = \nu^{-1}(\arg\max_{g \in G} \|f(g)\|_1) \tag{5.13}$$

5.7.3 Proofs

Proof of lemma 5.2.1 (See page 74)

Lemma 5.2.1. *π' defines a valid group action of G on the space $\mathcal{I}_K \times G/K$.*

Proof. Since $\bar{p}$ and $\overline{up}$ are coset representives for pK and $(up)K$, we can let $p = \bar{p}k_p$, $up = \overline{up}k_{up}$, where $k_p, k_{up} \in K$. From Equation (5.6), note that $\bar{p}' = \overline{up} = upk_{up}^{-1}$. Hence

$$\begin{aligned} \bar{p}'^{-1}u\bar{p} &= (upk_{up}^{-1})^{-1}u(pk_p^{-1}) \\ &= k_{up}p^{-1}u^{-1}upk_p^{-1} \\ &= k_{up}k_p^{-1} \in K \end{aligned} \tag{5.14}$$

Hence $\pi'(u)$ (as defined in Equation (5.6)) defines a transformation from the space $\mathcal{I}_K \times G/K$ to itself.

To prove π' is a group action, we would like to show that for all $u, u' \in G$

$$\pi'(u')(\pi'(u)[f_b,\ pK]) = \pi'(u'u)[f_b,\ pK]$$

Let $[f_b',\ p'K] = \pi'(u)[f_b,\ pK]$ and $[f_b'',\ p''K] = \pi'(u'u)[f_b,\ pK]$, by the definition of π' in Equation (5.6), we have

$$p''K = ((u'u)p)K = u'(upK) = u'(p'K)$$

and

$$f_b'' = \pi(\bar{p}''^{-1}(u'u)\bar{p})f_b = \pi(\bar{p}''^{-1}u'\bar{p}')\pi(\bar{p}'^{-1}u\bar{p})f_b = \pi(\bar{p}''^{-1}u'\bar{p}')f_b'.$$

Hence

$$[f_b'',\ p''K] = \pi'(u')[f_b',\ p'K]$$

It is easy to also check that

$$[f_b,\ pK] = \pi'(e)[f_b,\ pK]$$

Therefore π' defines a valid group action. □

Proof of proposition 5.2.1 (See page 75)

Proposition 5.2.1. *If the action of group G on the space $\mathcal{I}_G$ and $\mathcal{I}_K \times G/K$ are specified by π, π' (as defined in Equations (5.3) and (5.6)), and $\Phi : \mathcal{I}_G \to G/K$ is an equivariant map, then the operations $S_b{\downarrow}_K^G$ and $S_u{\uparrow}_K^G$ as defined in Equations (5.4) and (5.5) are equivariant maps between $\mathcal{I}_G$ and $\mathcal{I}_K \times G/K$.*

Proof. We first define a *restrict* operation on $f \in \mathcal{I}_G$ and an *extend* operation on $f_1 \in \mathcal{I}_K$:

$$f{\downarrow}_K^G(k) = f(k), \quad k \in K$$
$$f_1{\uparrow}_K^G(g) = \begin{cases} f_1(g) & g \in K \\ \mathbf{0} & g \notin K \end{cases}$$

where $f{\downarrow}_K^G \in \mathcal{I}_K$ and $f_1{\uparrow}_K^G \in \mathcal{I}_G$.

Recall that $s : G/K \to G$ is a function choosing a coset representive $\bar{p}$ for each coset pK. Using the restrict operation, the subsampling operation $S_b{\downarrow}_K^G(f; \Phi)$ in Equation (5.4) can equivalently be described as:

$$pK = \Phi(f)$$
$$f_b = [\pi(\bar{p}^{-1})f]{\downarrow}_K^G$$
$$[f_b,\ pK] = S_b{\downarrow}_K^G(f; \Phi)$$

And the upsampling operation $S_u{\uparrow}_K^G$ can be rewritten using the extend operation as:

$$f_u = S_u{\uparrow}_K^G(f_1, pK) = \pi(\bar{p})(f_1{\uparrow}_K^G)$$

For any $u \in G$ let $f' = \pi(u)f$ and $[f'_b,\ p'K] = \pi'(u)[f_b,\ pK]$ where π and π' are specified in Equation (5.3) and Equation (5.6) respectively. Since Φ is equivariant, we have

$$\Phi(f') = \Phi(\pi(u)f) = u \cdot \Phi(f) = u \cdot pK = p'K \tag{5.15}$$

Recall that $\bar{p}'^{-1}u\bar{p} = k_{up}k_p^{-1}$ from Equation (5.14). Hence $\bar{p}'^{-1} = k_{up}k_p^{-1}\bar{p}^{-1}u^{-1}$ and we have

$$\begin{aligned} [\pi(\bar{p}'^{-1})f']{\downarrow}_K^G &= [\pi(k_{up}k_p^{-1}\bar{p}^{-1}u^{-1})f']{\downarrow}_K^G \\ &= \pi(k_{up}k_p^{-1})[\pi(\bar{p}^{-1})\pi(u^{-1})f']{\downarrow}_K^G \\ &= \pi(k_{up}k_p^{-1})[\pi(\bar{p}^{-1})f]{\downarrow}_K^G \\ &= \pi(k_{up}k_p^{-1})f_b = f'_b \end{aligned} \tag{5.16}$$

From Equations (5.15) and (5.16), $S_b{\downarrow}_K^G$ is equivariant, i,e.

$$\pi'(u)S_b{\downarrow}_K^G(f;\Phi) = S_b{\downarrow}_K^G(\pi(u)f;\Phi)$$

For the upsampling operation, let $[f_1', p_1'K] = \pi'(u)[f_1, p_1K]$ and $f_u' = \pi(u)f_u$. From Equation (5.6) we have $f_1' = \pi(\bar{p}'^{-1}u\bar{p})f_1$. Hence

$$\begin{aligned} S_u{\uparrow}_K^G(f_1', p'K) &= \pi(\bar{p}')f_1'{\uparrow}_H^G \\ &= \pi(\bar{p}')[\pi(\bar{p}'^{-1}u\bar{p})f_1]{\uparrow}_H^G \\ &= \pi(\bar{p}')\pi(\bar{p}'^{-1}u\bar{p})(f_1{\uparrow}_H^G) \\ &= \pi(u\bar{p})f_1{\uparrow}_H^G = \pi(u)f_u = f_u' \end{aligned}$$

Therefore, $S_u{\uparrow}_K^G$ is equivariant, i.e.

$$\pi(u)S_u{\uparrow}_K^G([f_1, p_1K]) = S_u{\uparrow}_K^G(\pi'(u)[f_1, p_1K])$$

□

Proof of proposition 5.7.1 (See page 87)

Proposition 5.7.1. *ν^{-1} is the inverse of ν, hence ν is bijective. And $\forall u \in G$ we have:*

$$u \cdot \nu(p_1G_1, p_2G_2, \ldots, p_LG_L) = \nu(u \cdot (p_1G_1, p_2G_2, \ldots, p_LG_L)).$$

Proof. Firstly, we prove that $\nu^{-1} \circ \nu$ is an identity map, i.e. $\nu^{-1} \circ \nu = \mathbb{1}_z$. Let $r_{\text{eq}} = \nu(p_1G_1, p_2G_2, \ldots, p_LG_L) = (\bar{p}_1\bar{p}_2 \ldots \bar{p}_L)G_L$ and $(p_1'G_1, p_2'G_2, \ldots, p_L'G_L) = \nu^{-1}(r_{\text{eq}})$. From Equation (5.12), we know that $p_1'G_L = r_{\text{eq}}$. Hence we can let $p_1' = \bar{p}_1\bar{p}_2 \ldots \bar{p}_L g_L$ where $g_L \in G_L$. Since $(\bar{p}_2\bar{p}_3 \ldots \bar{p}_L) \in G_1$, for $l = 1$ we have

$$\begin{aligned} \bar{p}_1' &= \bar{p}_1 \\ p_2' = \bar{p}_1'^{-1} \cdot p_1' &= \bar{p}_2\bar{p}_3 \ldots \bar{p}_L g_L \end{aligned}$$

And recursively, for $l = 1, \ldots, L$ we would have

$$\begin{aligned} \bar{p}_l' &= \bar{p}_l \\ p_{l+1}' &= \bar{p}_{l+1} \ldots \bar{p}_L g_L \end{aligned}$$

Hence $(p_1G_1, p_2G_2, \ldots, p_LG_L) = (p_1'G_1, p_2'G_2, \ldots, p_L'G_L)$ and $\nu^{-1} \circ \nu = \mathbb{1}_z$.

Secondly, we prove that $\nu \circ \nu^{-1}$ is also an identity map, i.e. $\nu \circ \nu^{-1} = \mathbb{1}_r$. Let $(p'_1 G_1, p'_2 G_2, \dots, p'_L G_L) = \nu^{-1}(r_{\text{eq}})$ and $r'_{\text{eq}} = \nu(p'_1 G_1, p'_2 G_2, \dots, p'_L G_L)$. From Equation (5.12), we have $r_{\text{eq}} = p'_1 G_L$ and $p'_l = \bar{p}'^{-1}_{l-1} \cdot p_{l-1}$. Hence

$$r_{\text{eq}} = p'_1 G_L = \bar{p}'_1 p'_2 G_L = \cdots = \bar{p}'_1 \bar{p}'_2 \dots \bar{p}'_{L-1} p'_L G_L = \bar{p}'_1 \bar{p}'_2 \dots \bar{p}'_L G_L = r'_{\text{eq}}$$

Therefore $\nu \circ \nu^{-1} = \mathbb{1}_r$ and ν is bijective.

Lastly, we prove ν is equivariant. Let $(p'_1 G_1, p'_2 G_2, \dots, p'_L G_L) = u \cdot (p_1 G_1, p_2 G_2, \dots, p_L G_L)$ where the group action is implied by Equation (5.6). From Equation (5.14), we know that when $u \in G$, $\pi(\bar{p}'^{-1}_1 u \bar{p}_1) \in G_1$. Recursively, we have

$$\bar{p}'^{-1}_l \dots \bar{p}'^{-1}_2 \bar{p}'^{-1}_1 u \bar{p}_1 \bar{p}_2 \dots \bar{p}_l \in G_l \tag{5.17}$$

for $l = 1, \dots, L$. When $l = L$, from $\bar{p}'^{-1}_L \dots \bar{p}'^{-1}_2 \bar{p}'^{-1}_1 u \bar{p}_1 \bar{p}_2 \dots \bar{p}_l \in G_L$, we have

$$\bar{p}'_1 \bar{p}'_2 \dots \bar{p}'_L G_L = u \cdot (\bar{p}_1 \bar{p}_2 \dots \bar{p}_L G_L)$$

Hence $u \cdot \nu(p_1 G_1, p_2 G_2, \dots, p_L G_L) = \nu(u \cdot (p_1 G_1, p_2 G_2, \dots, p_L G_L))$. So that the group action given by Equation (5.6) is simplified to the left-action on the single group element. □

Proof of proposition 5.3.1 (See page 76)

Proposition 5.3.1. *When* ENC *and* DEC *are given by Equations* (5.8) *and* (5.9)*, and the group actions are specified as in Equation* (5.3) *and Equation* (5.6)*, for any* $g \in G$ *and* $f \in \mathcal{I}_G$*, we have*

$$[z_{inv}, g \cdot z_{eq}] = \text{ENC}(\pi(g) f)$$
$$\pi(g) \hat{f} = \text{DEC}(z_{inv}, g \cdot z_{eq})$$

Proof. Let $f, f' \in \mathcal{I}_G$ be the input feature maps where $f' = \pi(g) f$. Let $[f_l,\ p_l G_l]$ and $[f'_l,\ p'_l G_l]$ be the feature maps and subsampling indexes output by the l-th subsampling layer for f and f' respectively. Let $[z_{\text{inv}}, z_{\text{eq}}] = \text{ENC}(f)$ and $[z'_{\text{inv}}, z'_{\text{eq}}] = \text{ENC}(f')$ and let $r_{\text{eq}} = \nu(z_{\text{eq}})$, $r'_{\text{eq}} = \nu(z'_{\text{eq}})$ where ν is given in Equation (5.11).

From Equation (5.6) and the equivariance of $\text{G-CNN}_l(\cdot)$, we have

$$f'_1 = \pi(\bar{p}'^{-1}_1 g \bar{p}_1) f_1$$

and recursively:

$$f'_l = \pi((\bar{p}'_1\bar{p}'_2\dots\bar{p}'_l)^{-1}g(\bar{p}_1\bar{p}_2\dots\bar{p}_l))f_l \tag{5.18}$$

where $l = 1,\dots,L$ and $(\bar{p}'_1\bar{p}'_2\dots\bar{p}'_l)^{-1}g(\bar{p}_1\bar{p}_2\dots\bar{p}_l) \in G_l$ (see Equation (5.17)).

Since $G_L = \{e\}$ when $l = L$, we have

$$(\bar{p}'_1\bar{p}'_2\dots\bar{p}'_L)^{-1}g(\bar{p}_1\bar{p}_2\dots\bar{p}_L) = e$$

Hence

$$f'(e) = f(e)$$
$$(\bar{p}'_1\bar{p}'_2\dots\bar{p}'_L) = g\cdot(\bar{p}_1\bar{p}_2\dots\bar{p}_L)$$

which can be rewritten as

$$z'_{\text{inv}} = z_{\text{inv}}$$
$$r'_{\text{eq}} = g\cdot r_{\text{eq}}$$

From Proposition 5.7.1 we have

$$z'_{\text{eq}} = \nu^{-1}(r'_{\text{eq}}) = \nu^{-1}(ur_{\text{eq}}) = g\cdot\nu^{-1}(r_{\text{eq}}) = g\cdot z_{\text{eq}}$$

Therefore, for the encoders we have $[z_{\text{inv}}, g\cdot z_{\text{eq}}] = \text{ENC}(\pi(g)f)$.

For the decoders, let

$$z'_{\text{eq}} = (p'_1G_1, p'_2G_2, \dots, p'_LG_L) = g\cdot z_{\text{eq}}$$

Since the feature map at the l-th subsampling layer is transformed according to Equation (5.18), the sampling index is transformed accordingly:

$$p'_lG_l = (\bar{p}'^{-1}_{l-1}\dots\bar{p}'^{-1}_2\bar{p}'^{-1}_1 g\bar{p}_1\bar{p}_2\dots\bar{p}_{l-1})p_lG_L$$

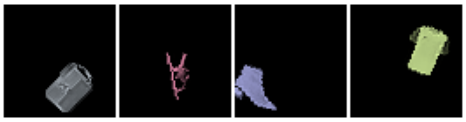
fFrom the definition of equivariant upsampling in Equation (5.5), we have

$$f'_{l-1} = \pi(\bar{p}'^{-1}_{l-1}\dots\bar{p}'^{-1}_2\bar{p}'^{-1}_1 g\bar{p}_1\bar{p}_2\dots\bar{p}_{l-1})f_{l-1}$$

where $l = L,\dots,1$. When $l = 1$, we have $f'_0 = \pi(g)f_0$ so that $\pi(g)\hat{f} = \text{DEC}(z_{\text{inv}}, g\cdot z_{\text{eq}})$.

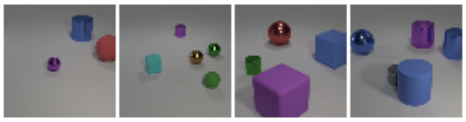
□

(a) Colored-dSprites



(b) FashionMNIST

(c) Multi-dSprites



(d) CLEVR6

Figure 5.6

5.7.4 Implementation details

In this section, we outline a few important implementation details, and leave other details to the reference code at https://github.com/jinxu06/gsubsampling.

5.7.4.1 Data

For Colored-dSprites and FashionMNIST datasets, we add colours to grayscale images from Matthey et al. [2017] and Xiao et al. [2017] by sampling random scaling for each channel uniformly between 0.5 and 1 following Locatello et al. [2019]. For FashionMNIST, we also apply zero-paddings to images to reach the size of 64×64. We then translate the images with random displacements uniformly sampled from $\{(x, y)| -18 \leq x, y \leq 18, x, y \in \mathbb{Z}\}$, and rotate the images with uniformly sampled degrees from $\{\frac{360 \times k}{32} | k = 0, \ldots, 32\}$. We use the original Multi-dSprites dataset as provided in Kabra et al. [2019]. For CLEVR6, we crop images from the original CLEVR [Johnson et al., 2017] at y-coordinates $(29, 221)$ bottom and top, and at x-coordinates $(64, 256)$ left and right as stated in Burgess et al. [2019]. We then resize the images to 64×64 so that we can use the same model for both multi-object datasets. We only use images with up to 6 objects in CLEVR following Greff et al. [2019]. For evaluation, we use a randomly sampled test set with 10000 examples that has no overlap with training data for all datasets. In Figure 5.6, we show examples from different datasets.

5.7.4.2 Model architectures

The equivariant map Φ One can insert arbitrary equivariant layers before and after we take the norm $\|\cdot\|_1$ in Equation (5.7). In experiments, we apply mean subtraction followed by average pooling with kernel size 5 before taking the norm, and apply Gaussian blur with kernel size 15 after taking the norm. They are inserted in the purpose of smoothing feature maps and avoiding non-unique argmax when possible (in Section 5.7.1.1 we discuss the case when non-unique argmax cannot be avoided). In practice, we use Equation (5.13) to obtain all subsampling indexes at the same time rather than passing through Φ multiple times.

Autoencoders For all single object experiments, we use 5 layers of (G-)equivariant convolutional layers in encoders, and the decoders mirror the architecture of the encoders except for the output layers. In baseline models, we use strided convolution as a way to perform subsampling/upsampling, while in equivariant models we use equivariant downsampling/upsampling. We rescale the number of channels such that the total number of parameters of the models roughly match one another. However, exact correspondence is not achievable because exact equivariant models use equivariant subsampling to transform feature maps into vectors at the final layer of the encoder, while baseline models apply flattening. Please see the reference implementation for other details about network architectures.

We use scale factor 2 for all subsampling and upsampling layers in baseline models. For GAE-$p1$, the feature maps at each layer are defined on the following chain of nested subgroups: $\mathbb{Z}^2 \geq (2\mathbb{Z})^2 \geq (4\mathbb{Z})^2 \geq (8\mathbb{Z})^2 \geq (16\mathbb{Z})^2 \geq \{e\}$. For For GAE-$p4$, we use $\mathbb{Z}^2 \rtimes \mathsf{C}_4 \geq (2\mathbb{Z})^2 \rtimes \mathsf{C}_4 \geq (4\mathbb{Z})^2 \rtimes \mathsf{C}_4 \geq (8\mathbb{Z})^2 \rtimes \mathsf{C}_4 \geq (16\mathbb{Z})^2 \rtimes \mathsf{C}_2 \geq \{e\}$. And for GAE-$p4m$, we use $\mathbb{Z}^2 \rtimes (\mathsf{C}_4 \rtimes \mathsf{C}_2) \geq (2\mathbb{Z})^2 \rtimes (\mathsf{C}_4 \rtimes \mathsf{C}_2) \geq (4\mathbb{Z})^2 \rtimes (\mathsf{C}_4 \rtimes \mathsf{C}_2) \geq (8\mathbb{Z})^2 \rtimes (\mathsf{C}_4 \rtimes \mathsf{C}_2) \geq (16\mathbb{Z})^2 \rtimes (\mathsf{C}_2 \rtimes \mathsf{C}_2) \geq \{e\}$.

Object discovery For baseline models, we adopt the exact same architecture as the original MONet [Burgess et al., 2019] using the implementation provided by Engelcke et al. [2020]. For MONet-GAEs, we simply replace Component VAEs in the original MONet with our V-GAEs. Both Component VAEs and V-GAEs have a latent size of 16.

5.7.4.3 Hyperparameters

For all single object experiments, we use Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.0001 and a batch size of 16. We use 16-bits precision to enable faster training and reduce memory consumption. For experiments on multi-object datasets, hyperparameters will match the original MONet [Burgess et al., 2019] except that we still use a batch size of 16 instead of 64 stated in the original paper. This is because we observed that in the low data regime, batch size 16 trains faster and performs no worse than batch size 64 for the problems we considered here.

5.7.4.4 Computational resources

In theory, the only computational overhead is caused by computing sampling indices, which is negligible compared to the forward pass of (G-)Convolutional layers. In practice, our current implementation uses `torch.gather` to perform subsampling, and relies on for-loops over data batches when applying group actions to feature maps, which we believe can be made more efficient. Hence on a single GeForce GTX 1080 GPU card, a standard GAE-p1 takes around 30 minutes to train for $100k$ steps, compared to 16 minutes for standard ConvAEs.

Chapter 6

Conclusions and Future Outlook

Throughout this thesis, we explored various approaches towards more data-efficient deep learning models.

In Chapter 1, we discussed the importance of data-efficient deep learning and the two recipes we will pursue: meta-learning and symmetries. To contextualize our work, in Chapter 2, we provided a brief introduction to meta-learning, neural processes, and the role of symmetries in deep learning.

In Chapter 3, we introduced MetaFun, a meta-learning method based on the encoder-decoder architecture similar to Conditional Neural Processes (CNPs). The main innovation of this work is to learn iterative updates to encode the context into the task representation directly in functions space. We demonstrated that CNP-style meta-learning method can match or surpass previous gradient-based methods on standard few-shot benchmarks. This work proved that meta-learning can be tackled under the conventional setting of supervised learning, providing a new ground to develop meta-learning algorithms.

To address meta-learning while factoring in uncertainty estimation, we introduced MNP in Chapter 4, a novel framework for constructing expressive neural generative SP models. By leveraging neural-parameterized Markov transition operators in function spaces, MNPs evolve a basic starting SP into a more flexible one. Our theoretical results confirmed that these neural transitions maintain the essential properties of exchangeability and consistency. Practical experiments further highlighted the potential of our approach on a variety of tasks. This work brings us one step closer

to constructing powerful neural SPs with theoretical guarantees. It also provides an effective means to construct data-efficient probabilistic models.

Meta-learning learns the prior from related tasks and the number of these tasks can be limited for some applications. So it is always good to incorporate prior known to us without relying on data, such as symmetries. In Chapter 5, we introduced group equivariant subsampling/upsampling, addressing the issue of broken symmetries observed in conventional subsampling or pooling layers. This work filled in the gap that previous work on equivariant models only focus on convolutional or attention layers. One can now construct equivariant models end-to-end such as equivariant autoencoders.

Having highlighted our research contributions, let us delve into potential future avenues and interesting open questions.

Firstly, few-shot learning, characterized by its ability to generalize from a limited number of examples, stands as a hallmark of intelligent systems. The fast advancements in this domain have been remarkable. Lately, the field is witnessing lots of new developments around large language models [Brown et al., 2020b], and in particular in-context learning [Xie et al., 2021, Min et al., 2022], which is analogous to CNP-type models but the backbone models are only trained for language modelling tasks. On the other hand, methodologies developed for meta-learning have found applications in other domains sharing similar hierarchical structures. This includes amortized optimization [Amos, 2022], differentiable optimization [Agrawal et al., 2019], Meta Optimal transport [Amos et al., 2022], and Meta-PDE [Qin et al., 2022]. These emerging areas or methods hold immense promise, and we anticipate their future progress.

Furthermore, in both our research and numerous other studies in the domain, it is been shown that leveraging symmetries can enhance sample efficiency. However, it's important to note that while symmetries are prevalent in the physical world, they may not always manifest in the observation space. For instance, translational and rotational symmetries in 3D spaces aren't necessarily maintained in 2D observations. Even within a 2D context, the symmetries of objects can be obscured due to occlusion. Moreover, while physical systems operate following conservation laws, our measurements are noisy so the exact conservation laws cannot hold. In addition, designing neural networks that preserve specific symmetries can be challenging. Practitioners might also be

unaware of certain symmetries present in their data. The discussion above highlights the importance of being able to automatically discover symmetries, bringing together learning and symmetries. Learning algorithms could discover symmetries through meta-learning [Zhou et al., 2020], and then utilize these symmetries for subsequent learning tasks. Recent findings even indicate that, in larger models, symmetries might spontaneously arise [Gruver et al., 2023].

Finally, a significant barrier to "learning to learn" for some applications is the lack of tasks in these domains. A potential solution lies in the use of synthetic data. For instance, in our experiments with geological data, we utilize a simulator that can generate an endless stream of examples. Within the realm of NPs, it is feasible to sample from existing SPs such as GPs and use this data to train NPs when real-world data is insufficient. Better usage of synthetic data might be a key ingredient in building data-efficient deep learning models suitable for a broader spectrum of problems.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Akshay Agrawal, Brandon Amos, Shane T. Barratt, Stephen P. Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In *Neural Information Processing Systems*, 2019. URL https://api.semanticscholar.org/CorpusID:202786139.

Brandon Amos. Tutorial on amortized optimization. *Found. Trends Mach. Learn.*, 16: 592–732, 2022. URL https://api.semanticscholar.org/CorpusID:258298859.

Brandon Amos, Samuel Cohen, Giulia Luise, and Ievgen Redko. Meta optimal transport. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:249605440.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Matthias Bauer, Mateo Rojas-Carulla, Jakub Bartłomiej Świątkowski, Bernhard Schölkopf, and Richard E Turner. Discriminative k-shot learning using probabilistic models. *arXiv preprint arXiv:1706.00326*, 2017.

Erik J Bekkers. B-spline CNNs on Lie groups. In *ICLR*, 2020.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

Dimitri P. Bertsekas and Steven E. Shreve. Stochastic optimal control : the discrete time case. 2007.

Benjamin Bloem-Reddy and Yee Whye Teh. Probabilistic symmetries and invariant neural networks. *Journal of Machine Learning Research*, 21(90):1–61, 2020. URL http://jmlr.org/papers/v21/19-322.html.

Paul C. Bressloff. Stochastic processes in cell biology. *Interdisciplinary Applied Mathematics*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020a. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon

Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020b. URL https://api.semanticscholar.org/CorpusID:218971783.

Wessel Bruinsma, James Requeima, Andrew YK Foong, Jonathan Gordon, and Richard E Turner. The gaussian neural process. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2021.

Tu Dinh Bui, Dinh Tran, and Richard E Turner. Student-t processes as alternatives to gaussian processes. In *International Conference on Machine Learning*, pages 1232–1241, 2016.

Yuri Burda, Roger B Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *ICLR (Poster)*, 2016.

Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

Anadi Chaman and Ivan Dokmanić. Truly shift-invariant convolutional neural networks. *arXiv preprint arXiv:2011.14214*, 2020.

Shing Chan and Ahmed H Elsheikh. Parametric generation of conditional geological realizations using generative neural networks. *Computational Geosciences*, 23(5): 925–952, 2019.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018a.

Tian Qi Chen, Xuechen Li, Roger Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *International Conference on Learning Representations*, 2018b.

Muyang Chung and Richard E Turner. Non-gaussian process regression with student-t processes. In *International Conference on Machine Learning*, pages 4607–4616, 2019.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999, 2016.

Taco S Cohen and Max Welling. Steerable cnns. In *International Conference on Learning Representations*, 2017.

Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *ICLR*, 2018a.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *International Conference on Learning Representations*, 2018b. URL https://openreview.net/forum?id=Hkbd5xZRb.

Taco S Cohen, Mario Geiger, and Maurice Weiler. Intertwiners between induced representations (with applications to the theory of equivariant neural networks). *arXiv preprint arXiv:1803.10743*, 2018c.

Taco S Cohen, Mario Geiger, and Maurice Weiler. A general theory of equivariant cnns on homogeneous spaces. *Advances in neural information processing systems*, 32:9145–9156, 2019.

Rob Cornish, Anthony Caterini, George Deligiannidis, and Arnaud Doucet. Relaxing bijectivity constraints with continuously indexed normalising flows. In *International conference on machine learning*, pages 2133–2143. PMLR, 2020.

Noel Cressie. The origins of kriging. *Mathematical geology*, 22(3):239–252, 1990.

Andreas Damianou and Neil D Lawrence. Deep gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR, 2013.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Guneet Singh Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. In *International Conference on Learning Representations*, 2019.

Sander Dieleman, Kyle W Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2015.

Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. In *International Conference on Machine Learning*, pages 1889–1898, 2016.

Emilien Dupont, Tuanfeng Zhang, Peter Tilke, Lin Liang, and William Bailey. Generating realistic geology conditioned on physical measurements with generative adversarial networks. *arXiv preprint arXiv:1802.03065*, 2018.

Emilien Dupont, Yee Whye Teh, and A. Doucet. Generative models as distributions of functions. In *AISTATS*, 2022.

Conor Durkan, George Papamakarios, Iain Murray, and Jascha Sohl-Dickstein. Neural spline flows. In *International Conference on Learning Representations*, 2019.

Vincent Dutordoir, Alan Saul, Zoubin Ghahramani, and Fergus Simpson. Neural diffusion processes. *arXiv preprint arXiv:2206.03992*, 2022.

Harrison Edwards and Amos Storkey. Towards a neural statistician. In *International Conference on Learning Representations*, 2016.

Martin Engelcke, Adam R. Kosiorek, Oiwi Parker Jones, and Ingmar Posner. GENESIS: Generative Scene Inference and Sampling of Object-Centric Latent Representations. *International Conference on Learning Representations (ICLR)*, 2020.

Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) equivariant representations with spherical CNNs. In *ECCV*, 2018.

Carlos Esteves, Ameesh Makadia, and Kostas Daniilidis. Spin-weighted spherical CNNs. In *NeurIPS*, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017a.

Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368, 2017b.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to Lie groups on arbitrary continuous data. In *ICML*, 2020.

Andrew Foong, Wessel Bruinsma, Jonathan Gordon, Yann Dubois, James Requeima, and Richard Turner. Meta-learning stationary stochastic process prediction with convolutional neural processes. *Advances in Neural Information Processing Systems*, 33:8284–8295, 2020.

Fred N. Fritsch and Judy Butland. A method for constructing local monotone piecewise cubic interpolants. *Siam Journal on Scientific and Statistical Computing*, 5:300–304, 1984.

Fabian B Fuchs, Daniel E Worrall, Volker Fischer, and Max Welling. SE(3)-Transformers: 3D roto-translation equivariant attention networks. In *NeurIPS*, 2020.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1690–1699, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo Jimenez Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes. *ArXiv*, abs/1807.01622, 2018c.

Robert Gens and Pedro M Domingos. Deep symmetry networks. *Advances in neural information processing systems*, 27:2537–2545, 2014.

Zoubin Ghahramani. A tutorial on gaussian processes. *Summer School on Machine Learning*, 6:67–80, 1999.

Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard Turner. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2018.

Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2018.

Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pages 2424–2433. PMLR, 2019.

Nate Gruver, Marc Anton Finzi, Micah Goldblum, and Andrew Gordon Wilson. The lie derivative for measuring learned equivariance. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=JL7Va5Vy15J.

Y. Guo, P. Bartlett, A. Smola, and R. C. Williamson. Norm-based regularization of boosting. *Submitted to Journal of Machine Learning Research*, 2001.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 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.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

Emiel Hoogeboom, Jorn WT Peters, Taco S Cohen, and Max Welling. HexaConv. In *ICLR*, 2018.

Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. In *International Conference on Learning Representations*, 2018.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Michael Hutchinson, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim. Lietransformer: Equivariant self-attention for lie groups. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

Harry Joe. Multivariate models and dependence concepts. *Chapman & Hall/CRC*, 1997.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Rishabh Kabra, Chris Burgess, Loic Matthey, Raphael Lopez Kaufman, Klaus Greff, Malcolm Reynolds, and Alexander Lerchner. Multi-object datasets. https://github.com/deepmind/multi-object-datasets/, 2019.

Angjoo Kanazawa, Abhishek Sharma, and David Jacobs. Locally scale-invariant convolutional neural networks. *arXiv preprint arXiv:1412.5104*, 2014.

Makoto Kawano, Wataru Kumagai, Akiyoshi Sannai, Yusuke Iwasawa, and Yutaka Matsuo. Group equivariant conditional neural processes. In *International Conference on Learning Representations*, 2020.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In *International Conference on Learning Representations*, 2018.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2018.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Gregory Koch. Siamese neural networks for one-shot image recognition. *Master's thesis, University of Toronto*, 2015.

Risi Kondor, Zhen Lin, and Shubhendu Trivedi. Clebsch–Gordan nets: a fully Fourier space spherical convolutional neural network. In *NeurIPS*, 2018.

Iryna Korshunova, Jonas Degrave, Ferenc Huszár, Yarin Gal, Arthur Gretton, and Joni Dambre. Bruno: A deep recurrent model for exchangeable data. *Advances in Neural Information Processing Systems*, 31, 2018.

Iryna Korshunova, Yarin Gal, Arthur Gretton, and Joni Dambre. Conditional bruno: A neural process for exchangeable labelled data. *Neurocomputing*, 416:305–309, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.11.108. URL https://www.sciencedirect.com/science/article/pii/S0925231220304987.

Carlo R. Laing and Gabriel J. Lord. Stochastic methods in neuroscience. 2009.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019a.

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019b.

Xuechen Li, Ricky Tian Qi Chen, Ting-Kam Leonard Wong, and David Duvenaud. Scalable gradients for stochastic differential equations. In *Artificial Intelligence and Statistics*, 2020.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. In *International Conference on Learning Representations*, 2019.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4114–4124. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/locatello19a.html.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020.

Diego Marcos, Michele Volpi, and Devis Tuia. Learning rotation invariant convolutional filters for texture classification. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2012–2017. IEEE, 2016.

Joe Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference. In *International Conference on Machine Learning*, pages 3403–3412, 2018.

Juan Maroñas, Oliver Hamelijnck, Jeremias Knoblauch, and Theodoros Damoulas. Transforming gaussian processes with normalizing flows. In *International Conference on Artificial Intelligence and Statistics*, pages 1081–1089. PMLR, 2021.

Jonathan Masci, Ueli Meier, D. Ciresan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *ICANN*, 2011.

Llew Mason, Jonathan Baxter, Peter L Bartlett, Marcus Frean, et al. Functional gradient techniques for combining hypotheses. *Advances in Large Margin Classifiers. MIT Press*, 1999.

Emile Mathieu, Adam Foster, and Yee Whye Teh. On contrastive representations of stochastic processes. In *NeurIPS*, 2021.

Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *ArXiv*, abs/2202.12837, 2022. URL https://api.semanticscholar.org/CorpusID:247155069.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21(132):1–62, 2020.

Tsendsuren Munkhdalai, Xingdi Yuan, Soroush Mehri, and Adam Trischler. Rapid adaptation with conditionally shifted neurons. In *International Conference on Machine Learning*, pages 3661–3670, 2018.

Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019.

Roger B Nelsen. *An introduction to copulas*. Springer Science & Business Media, 2007.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 721–731, 2018.

George Papamakarios, Iain Murray, Matthew Gorham, and Jascha Sohl-Dickstein. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019a. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019b.

Wolfgang Paul and Jörg Baschnagel. Stochastic processes: From physics to finance. 2000.

Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7229–7238, 2018.

Tian Qin, Alex Beatson, Deniz Oktay, Nick McGreivy, and Ryan P. Adams. Meta-pde: Learning to solve pdes quickly without a mesh. *ArXiv*, abs/2211.01604, 2022. URL https://api.semanticscholar.org/CorpusID:253265115.

Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 873–880, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553486. URL https://doi.org/10.1145/1553374.1553486.

Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian processes for machine learning. In *Adaptive computation and machine learning*, 2009.

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*. MIT press, 2006.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2016.

Scott Reed, Yutian Chen, Thomas Paine, Aäron van den Oord, SM Ali Eslami, Danilo Rezende, Oriol Vinyals, and Nando de Freitas. Few-shot autoregressive density estimation: Towards learning to learn distributions. In *International Conference on Learning Representations*, 2018.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. In *International Conference on Learning Representations*, 2018.

James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E Turner. Fast and flexible multi-task classification using conditional neural adaptive processes. *Advances in Neural Information Processing Systems*, 32, 2019.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018.

David W Romero and Mark Hoogendoorn. Co-attentive equivariant neural networks: Focusing equivariance on transformations co-occurring in data. In *ICLR*, 2020.

David W Romero, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Attentive group equivariant convolutional networks. In *ICML*, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, pages 92–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

Amar Shah, Andrew Wilson, and Zoubin Ghahramani. Student-t Processes as Alternatives to Gaussian Processes. In Samuel Kaski and Jukka Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 877–885, Reykjavik, Iceland, 22–25 Apr 2014. PMLR.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

E.P. Simoncelli, W.T. Freeman, E.H. Adelson, and D.J. Heeger. Shiftable multiscale transforms. *IEEE Transactions on Information Theory*, 38(2):587–607, 1992. doi: 10.1109/18.119725.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

Sebastien Strebelle. Conditional simulation of complex geological structures using multiple-point statistics. *Mathematical geology*, 34(1):1–21, 2002.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

Zoltán Sylvester, Paul Durkin, and Jacob A Covault. High curvatures drive river meandering. *Geology*, 47(3):263–266, 2019.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

Yee Teh, Michael Jordan, Matthew Beal, and David Blei. Sharing clusters among related groups: Hierarchical dirichlet processes. *Advances in neural information processing systems*, 17, 2004.

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

Nico G. van Kampen and William P. Reinhardt. Stochastic processes in physics and chemistry. 1981.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the limitations of representing functions on sets. *arXiv preprint arXiv:1901.09006*, 2019.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi: 10.21105/joss.03021. URL https://doi.org/10.21105/joss.03021.

Nicholas Watters, Loic Matthey, Christopher P Burgess, and Alexander Lerchner. Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. *arXiv preprint arXiv:1901.07017*, 2019.

Maurice Weiler and Gabriele Cesa. General E(2)-Equivariant Steerable CNNs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019a.

Maurice Weiler and Gabriele Cesa. General e (2)-equivariant steerable cnns. In *Advances in Neural Information Processing Systems*, pages 14334–14345, 2019b.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems*, pages 10381–10392, 2018a.

Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2018b.

Andrew G Wilson and Zoubin Ghahramani. Copula processes. *Advances in Neural Information Processing Systems*, 23, 2010.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037, 2017.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *ArXiv*, abs/2111.02080, 2021. URL https://api.semanticscholar.org/CorpusID:241035330.

Jin Xu, Jean-Francois Ton, Hyunjik Kim, Adam Kosiorek, and Yee Whye Teh. Metafun: Meta-learning with iterative functional updates. In *International Conference on Machine Learning*, pages 10617–10627. PMLR, 2020.

Jin Xu, Hyunjik Kim, Tom Rainforth, and Yee Whye Teh. Group equivariant subsampling. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Jin Xu, Emilien Dupont, Kaspar Märtens, Tom Rainforth, and Yee Whye Teh. Deep stochastic processes via functional markov transition operators. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

Richard Zhang. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*, pages 7324–7334, 2019.

Tuan-Feng Zhang, Peter Tilke, Emilien Dupont, Ling-Chen Zhu, Lin Liang, and William Bailey. Generating geologically realistic 3d reservoir facies models using deep learning of sedimentary architecture with generative adversarial networks. *Petroleum Science*, 16(3):541–549, 2019.

Tuanfeng Zhang, Paul Switzer, and Andre Journel. Filter-based classification of training image patterns for spatial simulation. *Mathematical Geology*, 38(1):63–80, 2006.

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Information maximizing variational autoencoders. *arXiv preprint arXiv:1706.02262*, 2017.

Allan Zhou, Tom Knowles, and Chelsea Finn. Meta-learning symmetries by reparameterization. In *International Conference on Learning Representations*, 2020.