Incorporating Inductive Biases into Machine Learning Algorithms

Ning Miao

Wolfson College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy in Statistics

Trinity 2024

Acknowledgements

Firstly, I would love to express my sincerest gratitude to my supervisors Tom Rainforth and Yee Whye Teh for their kindness, patience, and invaluable guidance. I still remember when I just started my study in Oxford, Tom told me, 'You are not here to help me, I'm here to help you.' And this is exactly what he did. Over the past four years, we had countless discussions about my projects, from which I learned a lot. In particular, he would edit my drafts word by word, which took him a lot of time, but helped me a lot in improving my writing skills. I owe Yee Whye a huge debt of gratitude for his strong support, both academically and emotionally. Whenever my research reaches a deadlock, he is always able to give me hope and help me find a way forward. With the support from Tom and Yee Whye, I have grown from a shy, fresh DPhil student into a confident researcher who is proud of his work. They make me feel safe in academia, which allows me to do my research freely.

I would love to say 'thank you' to all my coauthors, including Emile Mathieu, N. Siddharth, Hyunjik Kim, Adam Foster, and Yann Dubois, as well as all members of RainML and Yee Whye group, especially Freddie Bickford Smith, Tim Reichelt, Andrew Campbell, Desi R. Ivanova, Jannik Kossen, and Jiazhan Feng. I would like to express a special thanks to Jin Xu for his help in every stage of my DPhil study. I would also like to thank my other friends in Oxford, especially Xi Lin, Yutong Lu, Chao Zhang, Zhixiao Zhu, Hanwen Xing, Yifan Yu, Linying Yang, Yanzhao Yang, Jun Yang, Zhongyi Hu, Jessie Jiang, and Alex Buna, as well as my old friends at home, especially Yuhao Jia, Dean Cai, Cheng Gao, Zhuofan Hao, Yu Gao, Chenxing Li, Shurun Wang, Hao Zhou, Wenxian Shi, and Hao Xu. Their companionship has made my life full of sunshine. I am also very grateful to my former supervisor Lei Li for his continuous support and guidance.

I appreciate the funding from Tencent, China Scholarship Council, and ELISE European Network of AI Excellence Centres, which allowed me to focus on my research without worrying about money.

I am deeply thankful to my family, especially my parents, for their unconditional love. Without their encouragement and financial support, it would have been impossible for me to even start my study at Oxford. Born in a small town in northern China, it is not easy to get the opportunity to pursue one's academic dreams. I feel incredibly fortunate to be their child.

Finally, I would like to express my deepest gratitude to my partner Yao Li. She came to Oxford to accompany me since the start of my DPhil study. We have shared happiness and weathered difficult times together. She has always been there for me, and I cannot imagine the rest of my life without her.

Abstract

Recently, significant advances in artificial intelligence (AI) have surpassed what was imaginable even five years ago. Today, we can instruct diffusion-based models to generate high-quality videos from human descriptions or prompt large language models (LLMs) to assist with writing, translation, and even mathematical reasoning. These remarkable abilities arise from training massive deep-learning models on huge amounts of data. However, we do not always have enough data. In some tasks, such as mathematical reasoning or molecule generation, available data are very limited. Furthermore, despite current LLMs utilizing nearly all available data on the Internet, they remain imperfect. Thus, it is a critical question how to enhance the performance of AI systems when it is difficult to increase the amount of training data.

In this thesis, we address this challenge from the perspective of inductive biases. Specifically, we investigate how to effectively use human knowledge about data or tasks to optimize the behavior of a machine learning algorithm, without requiring extra data. We will first give a brief review of research on inductive biases, and then we will show how to incorporate inductive biases during structure designing, training, and inference of a machine learning model, respectively. We also performed extensive experiments demonstrating that incorporating appropriate inductive biases can greatly boost model performance on a variety of tasks without the need for additional data.

Contents

Chapter 1

Introduction

Since the invention of the backpropagation method [Amari, 1967, Linnainmaa, 1970], the paradigm for using a neural network remains largely unchanged. Specifically, we first use data to train a neural network and then use the trained network to do inference. For a discriminative task, most machine learning algorithms can be seen as mappings from a labeled training set to a function from data space to label space. Similarly, for a generative task, they map an unlabelled training set to a distribution on the data space. Nevertheless, from multilayer perceptrons (MLPs, Rosenblatt [1958]) to convolutional neural networks (CNNs, Fukushima [1980], LeCun et al. [1989]), recurrent neural networks (RNNs, Hochreiter and Schmidhuber [1997], Cho et al. [2014]), and then transformers [Vaswani et al., 2017], different machine learning algorithms have very different behaviors and performance even on the same training data. This is because different algorithms have different **inductive biases**. That is, given the same training data, they have different preferences on the functions and distributions that are consistent with training data. For example, a CNN would prefer a function with translational invariance and hierarchical structure compared with an MLP.

Inductive biases are critical to the generalization performances of machine learning algorithms, especially when we do not have enough data. In most cases, training data cannot cover the whole data space. Even for enormous corpora used to train large language models (LLMs), we can easily ask a question during test time that does not have an exact match in the training corpus. Consequently, machine learning algorithms need to use their inductive biases to decide how to generalize outside of training data. As a simple example, the k-means algorithm uses the inductive bias that

the label of a test sample is related to its neighbors in the training set to generalize to the whole data space. For smaller datasets, for example, in math, chemistry, and life science, inductive biases are more critical. As we have limited training data on these domains, most test samples would be far away from training data. Hence, more carefully constructed inductive biases are needed to guide the model to reasonably generalize to a test sample that may be very different from training samples.

In this thesis, we try to understand the inductive biases of current machine learning models and, more importantly, figure out how to translate human knowledge or preferences into the desired inductive biases of a machine learning algorithm. Specifically, we introduce three novel methods to add inductive biases into the model structure, training method, and inference method of a machine learning algorithm, respectively.

Incorporating inductive biases in model structures. Models are the most basic components of machine learning algorithms. For discriminative tasks, model structures, such as the architectures of neural networks, determine key characteristics of learned mappings such as the hierarchy of the information flow and their overall complexity. Other more elaborate inductive biases such as invariances and long-term dependencies can also be integrated into model structures. For generative tasks, besides sample-level inductive biases, distribution-level inductive biases can also be incorporated. In other words, the structural differences of generative models can lead to different preferences on the characteristics of distributions, such as sparsity, multimodality, and other topological properties. In Chapter 3, we show how to precisely control the distribution-level inductive biases of variational autoencoders (VAEs), which are critical to their generation performance and feature quality.

Incorporating inductive biases during training. Different training methods can lead to different local minima in the parameter space of a machine learning model, which, in turn, influences its generalization performance. For example, different optimizers can lead to local minima of different generalization performance. Specifically, as shown in Zhou et al. [2020b], models learned by Stochastic Gradient Descent (SGD) usually have better generalization ability than ADAM-alike optimizers [Kingma and Ba, 2015]. Data augmentation is another critical component in model training, which is widely used to boost the generalization performance of machine learning models. By carefully augmenting a training sample to a distribution of related new samples, we

implicitly introduce the knowledge of invariance to the learned models. In Chapter 4, we focus on finding a better way to incorporate invariance using learnable data augmentation, which achieves better classification performance compared with fixed augmentation algorithms.

Incorporating inductive biases during inference. For some tasks and models, we can directly use the output of trained models as our final answer. For example, we can use the logits from a CNN classifier to predict the class of an input image. However, for others, how to do inference with the trained model remains a challenging question. For general-purpose LLMs, their in-context learning [Brown et al., 2020] abililty means that we can change their behavior with a simple instruction or a few examples during inference. As a result, we need to pay special attention to the way we prompt them for a specific task. For example, adding Chain-of-Thought (CoT, Wei et al. [2022]) instructions or examples introduces the inductive biases that LLMs should reason step-by-step instead of directly jumping to a final answer, greatly boosting the reasoning ability of LLMs. In Chapter 5, we show how to further improve LLMs' reasoning performance using the knowledge that verification is critical to the reliability of a reasoning process. We find that by introducing the inductive biases of verification, we can significantly increase the performance of LLMs on reasoning tasks.

1.1 Thesis outlines

The following of this thesis will discuss inductive biases of machine learning algorithms from the above three perspectives. We will start with a more general discussion on inductive biases and then delve into three specific examples of incorporating inductive biases into machine learning algorithms.

In Chapter 2, we first give our definition of inductive bias and introduce different kinds of inductive biases. Then we discuss the inductive biases in different parts of machine learning algorithms, including model structures, training algorithms, and inference algorithms. We will also use a few examples to show previous efforts in incorporating inductive biases into machine learning algorithms.

In Chapter 3, we focus on introducing inductive biases in model structures. Specifically, we aim to use our knowledge of the real data distribution to build a more data-efficient

variational autoencoder (VAE, Goodfellow et al. [2014]).

We first explain why directly changing the prior can be a surprisingly ineffective mechanism for incorporating inductive biases into VAEs, and then introduce a simple and effective alternative approach: Intermediary Latent Space VAEs (InteL-VAEs). InteL-VAEs use an intermediary set of latent variables to control the stochasticity of the encoding process, before mapping these in turn to the latent representation using a parametric function that encapsulates our desired inductive bias(es). This allows us to impose properties like sparsity or clustering on learned representations, and incorporate human knowledge into the generative model. We show that these advantages, in turn, lead to both better generative models and better representations being learned.

In Chapter 4, we focus on introducing inductive biases during the training of a machine learning model. We introduce a method to automatically learn input-specific augmentations from data, which incorporate a more precise invariance into machine learning models.

Compared with previous augmentation methods, which use the same augmentation for all inputs, we instead introduce a learnable invariance module that maps from inputs to tailored transformation parameters, allowing instance-specific invariances to be captured. This can be simultaneously trained alongside the downstream model in a fully end-to-end manner or separately learned for a pre-trained model. We demonstrate that the method learns meaningful input-dependent augmentations for a wide range of transformation classes, which in turn provides better performance on both supervised and self-supervised tasks.

In Chapter 5, we focus on the inductive biases in the inference stage. Specifically, we tackle the problem of hallucination of chain-of-thoughts (CoT, Wei et al. [2022]) reasoning in large language models by self-checking. We introduce a general-purpose zero-shot verification schema for detecting reasoning errors step by step. We then use the results of these checks to improve question-answering performance by conducting weighted voting on multiple solutions to the question. We test the method on math- and logic-based datasets and find that it successfully recognizes errors and, in turn, increases final answer accuracies.

1.2 Included papers

Chapter 3 contains:

> **Ning Miao**, Emile Mathieu, N. Siddharth, Yee Whye Teh, and Tom Rainforth. On Incorporating Inductive Biases into VAEs. International Conference on Learning Representations (ICLR), 2022 [Miao et al., 2021]

Chapter 4 contains:

> **Ning Miao**, Tom Rainforth, Emile Mathieu, Yann Dubois, Yee Whye Teh, Adam Foster, and Hyunjik Kim. Learning instance–specific augmentations by capturing local invariances. International Conference on Machine Learning (ICML), 2023 [Miao et al., 2023a]

Chapter 5 contains:

> **Ning Miao**, Yee Whye Teh, and Tom Rainforth. SelfCheck: Using LLMs to Zero-Shot Check Their Own Step-by-Step Reasoning. International Conference on Learning Representations (ICLR), 2024 [Miao et al., 2023b]

Chapter 2

Literature Review and Discussion on Inductive Biases

In this chapter, we will first give a definition of inductive biases, and present different taxonomies of inductive biases to facilitate discussion. Then we will discuss the inductive biases of different machine learning design choices in model structures, training algorithms, and inference algorithms, before introducing recent endeavors in incorporating various inductive biases into machine learning models. Our goal is to inspire readers to think about what inductive biases are, why they are important, and how they can be used to build better machine learning models. However, we do not aim to cover all related works on inductive biases.

2.1 Definition and taxonomy of inductive biases

Assume $\mathcal{D}_{\text{train}}$ is a training set containing data pairs $(\mathrm{x}_i, \mathrm{y}_i)$ for discriminative tasks or unlabeled data points x_i for generative tasks. A machine learning algorithm ϕ can be viewed as a mapping from the space of training sets to the function space $\phi : \mathbb{D} \to \mathcal{F}$. Here, for discriminative models, $f = \phi(\mathcal{D}_{\text{train}})$ is a function from input space $\mathcal{X}$ to label space $\mathcal{Y}$, like an image classifier. For generative models, f is a probability over the data space $\mathcal{X}$, which is the learned distribution from training data. Using this notation, **inductive biases** are the characteristics of ϕ. For example, the inductive bias of rotational invariance means that ϕ tends to select functions f whose outputs are unaffected when the input images x are rotated. Different inductive biases focus

on different aspects of the machine learning algorithm, and they can be combined to reflect different human knowledge.

There are many perspectives from which to observe a machine learning algorithm ϕ. For ease of thinking and discussion, we need to categorize inductive biases. For example, inductive biases can be divided into restriction biases and preference biases.

- **Restriction biases** constrain the learned functions f to a specific set. For example, linear regression only considers linear relationships between x and y and excludes all non-linear functions from its hypothesis class. From a stochastic perspective, restriction biases compulsorily assign zero probability to regions outside of a certain subset in the function space $\mathcal{F}$.
- **Preference biases** are softer inductive biases that prefer certain types of functions over others without completely excluding them. Examples include adding regularizers (see Section 2.3.2) and data augmentation (see Section 2.3.3) during training. By doing so, the algorithm would prefer neural networks with smaller weights but is still willing to increase their weights to better fit training data. For a Bayesian perspective, a preference bias equals choosing an informative prior in the function space.

Essentially, restriction biases can be seen as special cases of preference biases, where the prior probabilities for all functions outside of the preferred area are set to zero. However, there is a fundamental difference between them. With a preference bias, if there are enough data to demonstrate that the preference is inappropriate, the inductive bias can typically be overridden, allowing the model to better learn the true distribution. This flexibility is not present with restriction biases. For example, even if we have infinite data pairs for the true function $y = x^2$, we will still end up getting a linear function when using linear regression.

We can also categorize inductive biases based on the knowledge they reflect. For example, some inductive biases focus on individual data points, while others may capture relations between samples.

- **Individual inductive biases** embody our preference of model behavior on each individual sample. For example, when designing an encoder to map an input x

to a feature z, we may hope z to be a sparse vector to facilitate interpretation and control.

- **Relational inductive biases** reflect knowledge involving multiple related samples. For example, if x is an image and x' is a slightly rotated version of x. We may prefer ϕ to select a classifier f, such that $f(x) = f(x')$ most of the time, which is known as rotational invariance. Other concepts such as equivariance and methods like k-nearest neighbors also reflect relational inductive biases.

- **Global inductive biases** are another kind of inductive biases that can not be categorized as individual or relational, as they provide a more holistic view of functions' behavior. For discriminative tasks, linear models assume a linear relationship between all x and y. Another basic inductive bias in neural networks is that function f should be continuous. For generative tasks, global inductive biases manifest as preferences for certain types of learned distributions. For instance, when learning from the MNIST dataset [Deng, 2012], we expect the learned distribution to be multi-modal rather than single-modal to reflect the knowledge that the dataset consists of images of ten distinct digits.

In the rest of the chapter, we will analyze the inductive biases inherent in mainstream machine learning algorithms. Through reverse engineering these algorithms, we aim to provide readers with insights into the relationship between inductive biases and the design of machine learning algorithms. Additionally, we will introduce recent endeavors in directly incorporating inductive biases into machine learning algorithms.

2.2 Inductive biases of model structures

In this section, we analyze the inductive biases of mainstream machine learning models from linear regression to multi-layer perceptrons, CNNs, and transformers. Then we compare the inductive biases of different generative models before delving into the example of incorporating invariance and equivariance into model structures.

2.2.1 Inductive biases of neural networks

Neural networks are the backbones of most modern machine learning algorithms. We will start from the simplest 'neural network', linear regression, and proceed to discuss and compare the inductive biases of different neural networks from multiple perspectives.

Linear regression Linear regression [Galton, 1886, Pearson, 1901] seeks to find a linear relationship between an input vector x and a target vector y, which can be seen as a single layer perceptron without activation. Due to its simplicity, the weight matrix W for the linear mapping can usually be directly calculated using the equation

$$W = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{Y}, \tag{2.1}$$

where $\mathrm{X} = [(x)_1, ..., (x)_N]^T$, $\mathrm{Y} = [(y)_1, ..., (y)_N]^T$ are the matrices of training inputs and targets, respectively. The linearity inductive bias in linear regression is restrictive and sometimes overly strong, given that many real-world relationships are non-linear. Consequently, various methods have been proposed to relax this inductive bias [Edwards and Parry, 1993, Wahba, 1990]. The most significant approach to allow non-linearity involves stacking multiple linear layers (with activations) to form a multi-layer perceptron, which serves as the foundation of modern deep learning.

Multi-layer perceptrons (MLPs) MLPs, also known as fully connected neural networks (FCNs), are the simplest neural networks in the true sense. According to the universal approximation theorem [Hornik et al., 1989], any continuous function can be approximated arbitrarily well by an MLP with sufficient depth and width. At first glance, it may appear that MLPs have no inductive biases except for continuity. However, MLPs do exhibit preferences for certain functions over others due to their structures and parameterization methods. We will now examine the inductive biases of MLPs. Given that MLPs serve as the foundational blocks for most modern neural networks, most of the arguments apply to more complicated neural networks as well.

We know that the expressiveness of a neural network increases as it becomes wider (has more neurons in each layer) or deeper (stacks more layers). However, a very wide but shallow MLP often does not work well in practice, as a result of lacking the **hierachical inductive bias** inherent in deeper neural networks, which encourages

learning a hierarchical feature structure. With multiple layers, MLPs can extract features in different levels of abstraction. Roughly speaking, deep MLPs decompose the complexity into a series of simpler feature abstraction tasks, which makes them more robust when dealing with unseen inputs during training.

Another significant inductive bias of MLPs is the **globality of weights**. In other words, perturbing a single weight in an MLP can affect its outputs for many or all possible inputs, depending on the type of activations used. As a result, when we continue to train a neural network with limited data in a new domain, the weights of the neural network undergo significant changes, leading to catastrophic forgetting for continual learning. The globality of weights also poses other challenges for neural networks, such as their incapability to model high-frequency functions effectively. To address this issue, Tancik et al. [2020] pass the input vector x through a Fourier feature mapping before feeding it to the neural network. More recently, Liu et al. [2024] introduce Kolmogorov-Arnold networks (KAN). KANs replace the global non-linear activations after addition with piecewise spline functions for each link between neighboring layers. By doing so, only a small proportion of weights are associated with a specific region in the data space, resulting in better performance on tasks such as continual learning and high-frequency signal fitting.

Convolutional neural networks (CNNs) CNNs can be seen as a restricted version of MLPs, which are specially designed for vision tasks. Specifically, for each layer, CNNs remove links between neurons if they are geometrically far away from each other. Additionally, CNNs usually share weights of convolution kernels at the same layer. The two modifications on MLPs equip CNNs with desired inductive biases on the image space, making them historically extremely competitive in vision tasks such as image classification and generation.

By only keeping links between neighboring neurons, CNNs reflect the belief that pixels closer to each other are more likely to work jointly to form higher-level features. For example, an eyeball and an eyebrow can indicate the more abstract concept, the eye, while an eyebrow and a table corner usually do not have any direct connections.

By sharing the weights of convolution kernels, CNNs further reduce the number of learnable parameters at each layer, which in turn allows for deeper CNNs. This design is based on another belief that neural networks should be translational invariant on

image data. In other words, any patch of an image should have the same high-level representation regardless of its position.

By incorporating these beliefs into the structure of neural networks, CNNs retain the same level of expressiveness and greatly reduce the number of parameters at the same time, which in turn reduces the probability of overfitting to spurious relationships. Experiments show that CNNs achieve significant performance gains compared with FCNs. However, these inductive biases are specifically designed for vision tasks, so they might not be suitable for tasks with different characteristics. For example, Chen et al. [2015] show that for text-dependent speaker verification, weight sharing is not helpful to model performance.

Transformers Transformers were proposed as a method of machine translation to better model long-range dependency and speed up training, compared with recurrent neural networks (RNNs). Different from RNNs, which compress information autoregressively into low-dimensional vectors, transformers use a multi-head attention mechanism to directly link related tokens. Since their invention, transformers have replaced RNNs and CNNs as the default models for both natural language understanding (NLU) and natural language generation (NLG). Recent works have found that by treating patches in images similar to tokens in sentences, transformers work well on computer vision (CV) tasks, overtaking carefully designed CNNs when trained on a large amount of data [Dosovitskiy et al., 2020, Liu et al., 2023].

There are a few recent works on certain aspects of the inductive biases of transformers such as Lavie et al. [2024]. A very common belief is that compared with RNNs and CNNs, transformers are more flexible, which means they have less restrictive inductive biases. This allows transformers to learn relationships more accurately, without being restricted by the very coarse inductive biases in RNNs and CNNs. A downside of more flexible models is that they need more training data, which makes transformers extremely data-hungry.

However, because FCNs are also thought of as very flexible models without many restrictive inductive biases, one may wonder what the differences are between them and transformers that lead to the significant performance improvement of transformers. One hypothesis is that the performance gap between FCNs and transformers is because we usually pre-train transformers on large-scale datasets, but use FCNs without

pre-training. To check this hypothesis, Bachmann et al. [2024] pre-trained large FCNs on ImageNet21k [Deng et al., 2009]. Although they observed increased accuracies compared with FCNs without pre-training, FCNs still performed significantly worse than ResNet-18 [He et al., 2016a], let alone more powerful vision transformers [Dosovitskiy et al., 2020]. So pre-training is at least not the only fundamental difference between FCNs and transformers.

After taking a closer look at the structure of transformers, we notice that there is a critical difference in inductive biases between the attention layer in a transformer and a fully connected layer in an FCN, which might explain their different behavior. The difference is that attention layers only allow the extraction of one-to-one relationships between input tokens, while fully connected layers are responsive to multi-item relationships, which makes the search space exponentially larger. Concretely, at each time step, the attention mechanism of the transformer will first generate a query vector, which is then compared with the key vectors of all tokens to extract one-to-one relationships. In comparison, fully connected layers can directly extract relationships among multiple tokens. Because every multi-item relationship can be represented by nesting multiple two-item relationships, transformers with multiple attention layers are as expressive as FCNs. At the same time, transformers encourage the learning of simpler relationships, i.e., relationships that involve fewer tokens. As a result, they have stronger generalization ability than FCNs, which suffer from complicated spurious relationships.

Another key design innovation of transformers is multi-head attention, which allows transformers to jointly attend to information from different perspectives at different positions. However, Liu et al. [2021] show that stacking multiple single-head attention layers can achieve a similar effect if carefully trained. As a result, it may not be a critical inductive bias of transformers.

2.2.2 Inductive biases of generative models

The discussion of the inductive biases of generative models has many distinctive factors compared with the inductive biases of neural networks. For image generation models, such as Generative adversarial networks (GANs, Goodfellow et al. [2014]), Variational autoencoders (VAEs, Goodfellow et al. [2014]) and Diffusion models [Ho et al., 2020], even though they are using similar neural networks (such as CNNs or

transformers) as backbones, they have very different behaviors. As most practical text generation models are based on the same autoregressive structure, we will focus on image generation.

GANs, VAEs, and Diffusion models all have very different structures, which leads to significantly different inductive biases. A GAN is formed of two neural networks, a generator that maps Gaussian noise to an image, and a discriminator that distinguishes between real and generated images. The discriminator is trained to recognize generated images, while the generator learns to fool the discriminator. At equilibrium, the discriminator cannot figure out the difference between generated and real images, which means the generator can generate images that are indistinguishable from real samples. As a result, the first priority of GANs is to generate high-quality rather than diverse images.

VAEs also consist of two components, an encoder and a decoder. The encoder is a probabilistic mapping from an image to a distribution of vectors on a usually low-dimensional feature space and the decoder learns to reverse this process. Because VAEs are trained to recover every single image, their generated images are usually quite diverse. However, the bottleneck of the low dimensional feature space and the pixel-wise training target lead to more blurry images.

A diffusion model is a sequence of image-to-image mappings that recovers clear images from noise. Each recovering mapping is a minor modification to the noisy input, which is not difficult to learn. However, because a diffusion model cascades many such mappings, it can be very powerful. As a result, high-fidelity images can be generated through the step-by-step editing process. Meanwhile, like VAEs, diffusion models are also trained to recover all input images in the training set, so their generation is very diverse.

2.2.3 Incorporating invariance and equivariance into model structures

We now use the example of invariance and equivariance to show how to incorporate certain inductive biases into a model. When we say a model f is invariant to certain transformation t on data space, we mean $f(x) = f(t(x))$ for all input x. For example, when f is an image classifier, we usually hope it to be invariant to small rotations.

The definition of equivariance is $t(f(x)) = f(t(x))$, where t can also be seen as a transformation on the output space. For example, when f predicts the position of human faces in an image, we expect the model's outputs would change accordingly if we move the image in a certain direction. By incorporating invariance and/or equivariance into a model, we can usually reduce the demand for data and achieve better performance.

In the past few years, researchers have explored different structures to introduce invariance or equivariance. Jaderberg et al. [2015] designed a learnable module, which is called the spatial transformer to learn a transformation (including scaling, cropping, and rotations) of the input image. By doing so, the module selects the most relevant and informative region of the input and transforms the cropped region into a canonical pose. For example, in an image recognition task, the transformer can focus on the main object of the image and rotate it to an upright pose. Though effective in improving the performance of downstream tasks, there is no guarantee on what invariance the spatial transformation would learn. Dieleman et al. [2016] are one of the earliest to incorporate strict rotational invariances. To do so, they simply rotate the original image by ($0^{\cdot}$, $90^{\cdot}$, $180^{\cdot}$, $270^{\cdot}$), and feed the 4 copies of the original image into a neural network before averaging the output features. Because of the commutativity of addition, the resulting network is strictly invariant to right-angle rotations. Cohen and Welling [2016] find that it is more efficient to rotate the convolution kernels than input images, and generalize the method from rotation to all transformations that form a finite group. All of these methods insert special layers or add a wrapper around the original neural networks to incorporate invariance or equivariance. Recently, researchers have found that invariance can be explicitly learned without requiring special invariance-related labels Benton et al. [2020]. In Chapter 4, we show how to learn instance-specific invariances, taking into consideration the fact that invariances are very different at different places in the space of images.

2.3 Inductive biases of training algorithms

Once we have a machine learning model, the next thing is to train it on raw or augmented data. In this section, we discuss the inductive biases of different optimizers, regularizers, and data augmentation schemes, and show how they influence the performance of resulting trained models.

2.3.1 Inductive biases of optimizers

When we have a machine learning model, such as a neural network, we usually need to randomly initialize its weights and train it on a set of training data. To do so, we need an optimizer to find a local update of the weights, in order to reduce a pre-defined loss, which is usually a function of neural networks' outputs and the true labels or values. However, because of the non-linear and non-convex nature of most machine learning models, there are usually a lot of local minima. Different optimizers would prefer different local minima, and different minima usually lead to very different generalization performances of the trained model.

Nowadays, the default choices for optimizers are mostly stochastic gradient methods, no matter if we are training a small FCN or a huge LLM with billions of parameters. The success of stochastic gradient (SG) based methods is believed to be a result of some of their critical inductive biases. Firstly, they usually prefer simpler functions to complicated ones. For example, Rahaman et al. [2019] find that SG methods tend to learn low-frequency features first before minimizing the residue of loss by extracting more complicated features. As a result, an SG optimizer would prioritize simple relationships over complicated ones, which would, in turn, lead to better out-of-domain generalization ability. Secondly, researchers have found that the local minima reached by SG methods are more likely to be broad minima [Bradley et al., 2022]. Broad local minima are minima whose loss Hessians have small eigenvalues, which means they are less sensitive to the differences between training and testing distributions. An explanation for why SG optimizers tend to find broad minima is that different mini-batches have different data distributions, but once the weights converge to some minima, the minima must generalize to different mini-batches. Please refer to Gurnee [2022] for a more extensive discussion.

Even different SG optimizers can have different inductive biases. A well-known phenomenon is that though vanilla stochastic gradient optimizers have slower convergence speed than adaptive gradient optimizers (AG) such as ADAM [Kingma and Ba, 2015], they usually generalize better. Zhou et al. [2020b] argue that this phenomenon is a result of the instability of SGD. Concretely, the geometry adaptation in ADAM via adaptively scaling each gradient coordinate makes it insensitive to random noises that frequently change directions. However, these random noises are potentially critical in escaping from local basins and searching for flat minima [Neelakantan et al., 2015].

2.3.2 Inductive biases of regularizer

Adding regularizers, such as l_2, l_1 regularizers [Drucker and Le Cun, 1992, Hinton, 2012, Tibshirani, 1996] and dropout [Srivastava et al., 2014], is an effective way to introduce global-level inductive biases on the training process and the trained models. In this part, we analyze and compare the inductive biases of some commonly used regularizers.

l_2 and l_1 regularizers Both l_2 and l_1 regularizations are based on the belief that smaller weights lead to smoother and simpler models, which tend to generalize better, so should be preferred when fitting an over-parameterized neural network to limited training data. They can both be applied by adding an extra term to the original loss functions(, or equivalently by weight decay). The difference is that l_2 regularizers shrink weights proportionally to their value, while l_1 regularizers deduct the same value to the absolute value of each weight. Consequently, l_1 regularizers can push unused weights to zero very rapidly, effectively cutting the links between neuron pairs in neighboring layers. Differently, the l_2 regularizer minimizes the squared Frobenius norm of each weight matrix, which equates to minimizing the sum of squared singular values of each weight matrix. By applying the l_2 regularizer, the model tends to learn smoother layers with small Lipschitz constants. However, l_2 regularization rarely pushes weights to zero. The different characteristics of l_1 and l_2 regularizations make them suitable for different kinds of tasks. l_1 regularization is commonly used to select features, while l_2 regularization is more effective in improving the robustness and generalization ability of a machine learning model.

Dropout Dropout is another regularizer to introduce the inductive bias of redundancy into neural networks. Specifically, dropout randomly deactivates some input and intermediate nodes as well as their connections with other nodes during training to force neural networks to learn more redundant features and prevent them from overly relying on specific features. Dropout also reduces co-adaption between nodes. For example, to learn a linear function $y = 2x$, dropout discourages neural networks such as $h_1 = 22x$, $h_2 = -20x$ and $y = h_1 + h_2$, which would lead to inefficient use of neural network capacity. Another inductive bias brought by dropout is the learned model is more likely to be a hierarchical ensemble of several related sub-models. During training, we are essentially learning sub-models created by randomly removing some

neurons. During inference, all neurons are turned on, which can be seen as an ensemble of sub-models learned during training. There are other works on the inductive biases of dropout from other perspectives, such as Helmbold and Long [2015, 2016, 2018]. Interested readers can refer to their papers for more theoretical discussions.

Other regularizers There are many other kinds of regularization methods. For example, early stopping [Zhang and Yu, 2005] directly avoids the problem of overfitting by constantly evaluating model performance on leave-out data. Also, when training a VAE, the regularization term prevents overfitting by encouraging a smooth conditional distribution of the latent variable. Otherwise, it would degenerate into a vanilla autoencoder (AE, Rumelhart et al. [1986]), which often simply memorizes the training data. Researchers also design their own regularizers for different purposes. For example, in Section 3.6.3, we use a sparsity regularizer to encourage sparse latent variables in a VAE. In Section 4.3.2, we show how to design a regularizer on the entropy of transformation distributions to learn augmentations as diverse as possible.

2.3.3 Incorporating inductive biases by data augmentation

Besides directly encoding invariance into model structures, like in Section 2.2.3, we can also use data augmentation to instill invariance during training. Assume we are doing image classification and we have a training set consisting of multiple image-label pairs $(\mathrm{x}_i, \mathrm{y}_i)$. To enlarge the training set, as well as instill the knowledge that labels should be invariant to certain transformation t (such as a rotation or flipping) on x, we simply augment the training set with new data in the form of $(t(\mathrm{x}_i), \mathrm{y}_i)$. In practice, we sample t from different sets of transformations $\mathcal{T}$ to introduce different kinds of invariances. For example, for images, $\mathcal{T}$ can contain transformations including flipping, translation, rotation, zooming-in, and zooming-out to instill geometric invariance. Similarly, we can use color jittering to introduce invariance in the color space.

Data augmentation is also widely used in other domains such as speech recognition and natural language processing. However, because of the differences in invariances in different domains, data augmentations also differ a lot across domains. In speech recognition, augmentations are mainly adding Gaussian noises or changing the speed of the signal. Because of the discrete nature of natural languages, data augmentation on text space is more challenging than on image or audio spaces. Namely, it is not

easy to add a 'noise' to a sentence that changes it literally but keeps its labels, such as sentiment, topic, or meaning. We can randomly change some words in a sentence, but that would easily lead to non-fluent sentences or changed labels. Though we cannot use human-designed transformation to augment a dataset on text, we can train two translation models to do back translation [Sennrich et al., 2016] or a large language model to directly paraphrase a sentence.

A problem with traditional data augmentation methods is that augmentations need to be pre-defined by humans using prior knowledge, which makes them inconvenient to deal with datasets and tasks that have different invariances. For example, the classification of animals is nearly fully invariant to color jittering, but the classification of rocks highly depends on color information, so we should not apply large color jittering to it. To solve this problem, a lot of efforts have been made to automatically learn suitable augmentations for different datasets and tasks [Cubuk et al., 2018, 2020, Lim et al., 2019, Ho et al., 2019, Hataya et al., 2020, Li et al., 2020, Zheng et al., 2022, Benton et al., 2020]. In essence, they try to learn the parameters of augmentations, such as the maximum angles for rotations or maximum changes of brightness, aiming to directly learn the invariances inherent in datasets and tasks.

The previous augmentation learning methods learn different invariances and augmentations for different datasets and tasks, but even in the same datasets, desired invariances can be very different for different samples. For example, changing the color of a leaf from yellow to green results in another leaf, but the same transformation would change a lemon to nearly a lime. In Chapter 4, we discuss this problem and propose an effective method to incorporate local invariance by learning instance-specific augmentation.

2.4 Inductive biases of inference algorithms

Even for a fixed model, inference algorithms can largely influence its inductive biases. For a simple example, when doing text generation, beam search usually results in higher-quality outputs than token-by-token sampling by filtering out low-confidence samples. In this section, we introduce several representative methods to introduce inductive biases during inference time. We will start with two general examples and then discuss the very special case of LLM inference.

2.4.1 Introducing inductive biases by test-time augmentation

As described in Section 2.3.3 data augmentation is an effective way to incorporate invariances into a machine learning model during training. Interestingly, data augmentation can also be applied to incorporate the same invariances during inference. Assume we have an image classification model f and a set of transformations $\mathcal{T}$ that we expect the model to be invariant to. For an input image x, instead of directly using $f(x)$ as the predicted label for x, we first augment x into a set of transformed images $t(x)$, where $t \in \mathcal{T}$. Then $\mathbb{E}_{t \in \mathcal{T}} f(t(x))$ forms the invariant prediction of x's label. If $\mathcal{T}$ forms an invariance group, two inputs x and x' share the same real label y if $x = t(x')$ for some transformation $t \in \mathcal{T}$. In practice, when $\mathcal{T}$ is an infinite set, such as the 2D rotation group, we need to randomly sample a finite $\hat{\mathcal{T}} \subset \mathcal{T}$ to represent the whole invariance group. According to the law of large numbers, the predicted label for any input x in the same invariance group will converge to the same label for large enough $\hat{T}$. In other words, test-time augmentation will asymptotically incorporate the invariance into the model. Meanwhile, by averaging the model outputs for multiple views of an input, we can effectively reduce the variances of the model, which are caused by random mistakes at certain inputs.

2.4.2 Introducing inductive biases by another model

If we want to change the generation distribution of a language model, the most direct idea is to finetune it on corresponding datasets. For example, if we want to generate texts with a certain sentiment, we can finetune the language model on sentences with the sentiment. However, finetuning may be impossible when the language model is very large or we do not have enough data for finetuning. So the question is whether we can change the behavior of a language model without finetuning it.

Miao et al. [2019] use an external classifier to introduce desired inductive biases into a fixed language model. Specifically, they use the classifier to bias the output distribution of the language models towards the expected direction, for example, increasing the probability of sentences with a certain sentiment. Then they apply efficient sampling algorithms to sample from the biased distribution. Because there are various off-the-shelf classifiers for different desired inductive biases, this method is very convenient to use in practice. Even though we need to train it from scratch, training a classifier usually requires far fewer samples than training a generative model.

As a result, the method is very effective in introducing specific inductive biases while maintaining the generation quality of the language model.

2.4.3 Inductive biases in large-language-model inference

Large language models [Brown et al., 2020, Chowdhery et al., 2022, Touvron et al., 2023] have nearly the same structure as small transformer-based language models, the only differences are (1) they are much wider and deeper, with billions to hundreds of billions of parameters and (2) they are trained on a huge amount of unlabeled text data, which contain trillions of tokens or more. As language models, they have lower perplexities and generate high-quality texts. Meanwhile, some very interesting abilities have emerged only for large language models, one of which is in-context learning. Specifically, when we want to teach an LLM to tackle a problem, we no longer always need to finetune it on a labeled dataset. We can simply give the LLM several examples or even a natural language instruction, which are called prompts, and the LLM follows the examples or the instruction to solve the problem. In-context learning has greatly changed the paradigm of using a machine learning model. Now we can instill our knowledge by simply designing examples or instructions.

A widely used prompting method is Chain-of-Thought (CoT), which greatly improves LLMs' reasoning ability. Based on the knowledge that complex reasoning is a combination of simpler reasoning steps, CoT uses a few examples or an instruction to teach the model to think step by step, and generate intermediate reasoning steps before reaching a conclusion. Experiments have shown that the instilled inductive bias can greatly improve LLMs' reasoning performance, especially on math problems.

Although effective in comparably simple reasoning problems, LLMs are found to frequently make mistakes in complex reasoning. The reason is two-fold. Firstly, LLMs are good at learning superficial relationships, without fully understanding deep logic relationships. Consequently, they tend to use their experience in a lexically similar problem to solve the current problem and ignore the subtle differences. Secondly, when an LLM does not know something, it would usually make up an answer, instead of saying 'I don't know.'. A lot of efforts have been devoted to solving these problems, such as Ji et al. [2023], Lin et al. [2022]. In Chapter 5, we alternatively improve LLMs' reasoning accuracies by checking their own outputs. Concretely, by incorporating the inductive bias that checking should also be step-by-step and each checking step should

be made as simple as possible, we prompt LLMs to do self-checking using only simple instructions. By filtering out reasonings labeled as incorrect, we can achieve higher accuracies on a variety of problems.

2.5 Connections with other machine learning concepts

In this section, we discuss a few machine-learning concepts closely related to inductive bias. However, in the following chapters, we will focus on our original definition of inductive biases, that is models' explicit preference over some functions over others, without delving into Bayesian methods or learnable inductive biases.

Bayesian machine learning Bayesian machine learning (BML) is an umbrella containing all machine learning methods that are based on Bayes' theorem

$$p(\theta|\mathcal{D}_{\text{train}}) = \frac{p(\theta) \cdot p(\mathcal{D}_{\text{train}}|\theta)}{p(\mathcal{D}_{\text{train}})}, \tag{2.2}$$

where θ is the model parameter with prior distribution $p(\theta)$, $\mathcal{D}_{\text{train}}$ is training data, and $p(\mathcal{D}_{\text{train}}|\theta)$ is the likelihood of $\mathcal{D}_{\text{train}}$ under the model with parameter θ. Unlike regular machine learning models, which aim to generate a single-point maximum likelihood estimation (MLE) of θ, BML methods take the prior $p(\theta)$ into consideration. Sometimes we still want a single point estimation of θ, but we instead learn a maximum a posteriori (MAP) estimation, which is often equivalent to performing MLE with an additional regularization term. More frequently, we are not satisfied with a single-point estimation, but we instead estimate the posterior distribution of θ given its prior $p(\theta)$ and data $\mathcal{D}_{\text{train}}$.

BML can be seen as a direct way to introduce inductive biases. Specifically, the prior distribution $p(\theta)$ can reflect human preferences over different θ and consequently different models. For example, we can use a Gaussian prior with zero mean and small variance to reflect our belief that a model should have small parameters.

As an example, Bayesian neural networks (BNNs, MacKay [1995], Lampinen and Vehtari [2001]) combine the merits of BML and neural networks by assigning a distribution on the weights of neural networks. Because of their ability to estimate the

uncertainty of model predictions, BNNs are widely used for analytical purposes [Kendall and Gal, 2017, Kwon et al., 2018].

Because BML is a large but relatively independent area, with its own theory and methodology, we will leave its discussion for future work.

Meta-learning Each traditional supervised learning algorithm can be seen as a mapping ϕ from the training set $\mathcal{D}_{\text{train}}$, which is formed of (x, y) pairs, to the learned function f from the input x to the label y. Differently, meta-learning learns the mapping ϕ from data, which is especially useful when we have limited training data on the target task, but there is plenty of data on related tasks. There are many different kinds of meta-learning methods. For example, Koch et al. [2015] use the idea of metric-based meta-learning to build an image classification model using only a few example images for each new class. Specifically, they train a similarity metric using data from old classes. During test time, they use the similarity metric to match test images with examples from each class. As an example of model-based meta-learning, Santoro et al. [2016] learn a mapping from $(\mathcal{D}_{\text{train}}, x)$ to y on related tasks using a memory augmentation RNN. Differently, Ravi and Larochelle [2016] directly predict the parameters θ of a trained neural network using a few examples on the target domain, which is an optimization-based meta-learning algorithm.

All these methods can be seen as learning 'inductive biases' from related domains and applying them to the target domain. Although they are effective in reducing data requirements and improving performance on the target domain, it is hard to explain or manually manipulate the 'inductive biases'. As a result, we will not delve into the discussion of meta-learning methods in this work.

Transfer learning Based on the assumption that knowledge learned for a task could be useful for another related task, transfer learning simply reuses all or a part of the neural network trained for the original task as a starting point to train neural networks for new tasks. The purpose of doing so is to save the computation and reduce the data requirement to learn low-level features again, which are largely shared across related tasks. Transfer learning is widely used in CV and natural language processing (NLP). For example, Kolesnikov et al. [2020] pre-train a large visual transformer on huge image classification datasets such as JFT [Sun et al., 2017] or ImageNet-21k [Deng et al., 2009] and finetune the model on smaller domains, which significantly improves

classification accuracies. BERT [Kenton and Toutanova, 2019, He et al., 2020] and LLMs pre-train transformers to predict missing tokens in the middle or the end of a sequence and can be easily fine-tuned to do text classification, part-of-speech tagging, and even generative tasks, such as abstraction and question-answering.

Similar to meta-learning, 'inductive biases' in transfer learning methods are difficult for humans to analyze and manipulate, so we will limit our discussion on interpretable inductive biases to the following chapters.

Chapter 3

On Incorporating Inductive Biases into VAEs

3.1 Introduction

Variational auto-encoders (VAEs) provide a rich class of deep generative models (DGMs) with many variants [Kingma and Welling, 2014, Rezende and Mohamed, 2015, Burda et al., 2016, Gulrajani et al., 2016, Vahdat and Kautz, 2020]. Based on an encoder-decoder structure, VAEs encode datapoints into latent embeddings before decoding them back to data space. By parameterizing the encoder and decoder using expressive neural networks, VAEs provide a powerful basis for learning both generative models and representations.

The standard VAE framework assumes an isotropic Gaussian prior. However, this can cause issues, such as when one desires the learned representations to exhibit some properties of interest, for example sparsity [Tonolini et al., 2020] or clustering [Dilokthanakul et al., 2016], or when the data distribution has very different topological properties from a Gaussian, for example multi-modality [Shi et al., 2020] or group structure [Falorsi et al., 2018]. Therefore, a variety of recent works have looked to use non-Gaussian priors [van den Oord et al., 2017, Tomczak and Welling, 2018, Casale et al., 2018, Razavi et al., 2019, Bauer and Mnih, 2019], often with the motivation of adding inductive biases into the model [Davidson et al., 2018b, Mathieu et al., 2019b, Nagano et al., 2019, Skopek et al., 2019].

In this work, we argue that this approach of using non-Gaussian priors can be a

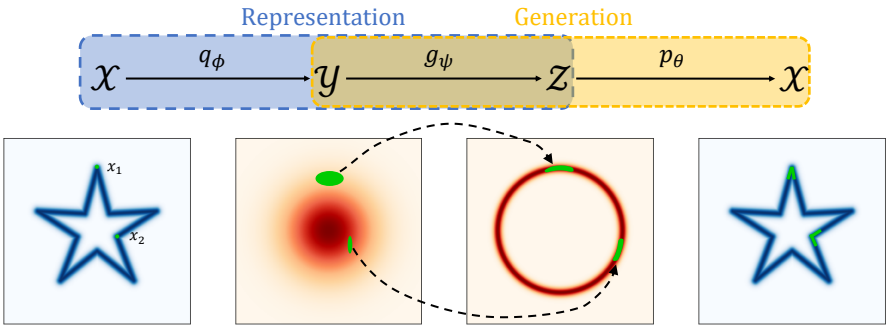


Figure 3.1: Example InteL-VAE with star-like data. We consider the auto-encoding for two example datapoints (x_1 and x_2, shown in green), which are first stochastically mapped to $\mathcal{Y}$ using a Gaussian encoder. This embedding is then pushed forward to $\mathcal{Z}$ using the *non-stochastic* mapping g_ψ, which is a radial mapping to enforce a spherical distribution. Decoding is then done in the standard way from $\mathcal{Z}$, with the complexity of the decoder mapping simplified by the induced structural properties of $\mathcal{Z}$.

problematic, and even ineffective, mechanism for adding *inductive biases* into VAEs. Firstly, non-Gaussian priors will often necessitate complex encoder models to maintain consistency with the prior's shape and dependency structure [Webb et al., 2018], which typically no longer permit simple parameterization. Secondly, the latent encodings are still not guaranteed to follow the desired structure because the 'prior' only appears in the training objective as a regularizer on the encoder. Indeed, Mathieu et al. [2019b] find that changing the prior is typically insufficient in practice to learn the desired representations at a *population level*, with mismatches occurring between the data distribution and learned model.

To provide an alternative, more effective, approach that does not suffer from these pathologies, we introduce *Intermediary Latent Space VAEs* (InteL-VAEs), an extension to the standard VAE framework that allows a wide range of powerful inductive biases to be incorporated while maintaining an isotropic Gaussian prior. This is achieved by introducing an *intermediary* set of latent variables that deal with the stochasticity of the encoding process *before* incorporating the desired inductive biases via a parametric function that maps these intermediary latents to the latent representation itself, with the decoder taking this final representation as input. See Figure 5.2.1 for an example.

The InteL-VAE framework provides a variety of advantages over directly replacing the prior. Firstly, it directly enforces our inductive biases on the representations, rather than relying on the regularizing effect of the prior to encourage this implicitly. Secondly, it provides a natural congruence between the generative and representational models via sharing of the mapping function, side-stepping the issues that non-Gaussian

priors can cause for the inference model. Finally, it allows for more general and more flexible inductive biases to be incorporated, by removing the need to express them with an explicit density function and allowing for parts of the mapping to be learned during training.

To further introduce a number of novel specific realizations of the InteL-VAE framework, showing how they can be used to incorporate various inductive biases, enforcing latent representations that are, for example, multiply connected, multi-modal, sparse, or hierarchical. Experimental results show their superiority compared with baseline methods in both generation and feature quality, most notably providing state-of-the-art performance for learning sparse representations in the VAE framework.

To summarize, we a) highlight the need for inductive biases in VAEs and explain why directly changing the prior is a suboptimal means for incorporating them; b) propose InteL-VAEs as a simple but effective general framework to introduce inductive biases; and c) introduce specific InteL-VAE variants which can learn improved generative models and representations over existing baselines on a number of tasks.

3.2 The need for inductive biases in VAEs

Variational auto-encoders (VAEs) are deep stochastic auto-encoders that can be used for learning both deep generative models and low-dimensional representations of complex data. Their key components are an encoder, $q_\phi(z|x)$, which probabilistically maps from data $x \in \mathcal{X}$ to latents $z \in \mathcal{Z}$; a decoder, $p_\theta(x|z)$, which probabilistically maps from latents to data; and a prior, $p(z)$, that completes the generative model, $p(z)p_\theta(x|z)$, and regularizes the encoder during training. The encoder and decoder are parameterized by deep neural networks and are simultaneously trained using a dataset $\{x_1, x_2, ..., x_N\}$ and a variational lower bound on the log-likelihood, most commonly,

$$\mathcal{L}(x, \theta, \phi) := \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p_\theta(x|z)\right] - D_{\mathrm{KL}} \left(q_\phi(z|x) \parallel p(z)\right). \tag{3.1}$$

Namely, we optimize $\mathcal{L}(\theta, \phi) := \mathbb{E}_{x\sim \mathrm{p_{data}(x)}} \left[\mathcal{L}(x, \theta, \phi)\right]$, where $\mathrm{p_{data}(x)}$ represents the empirical data distribution. Here the prior is typically fixed to a standard Gaussian, i.e. $p(z) = \mathcal{N}(z; 0, I)$.

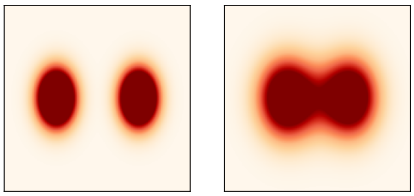
While it is well documented that this standard VAE setup with a 'Gaussian' latent space can be suboptimal [Davidson et al., 2018a, Mathieu et al., 2019b, Tomczak

and Welling, 2018, Bauer and Mnih, 2019, Tonolini et al., 2020], there is perhaps less of a unified high-level view on exactly when, why, and how one should change it to incorporate inductive biases. Note here that the prior does not play the same role as in a Bayesian model: because the latents themselves are somewhat arbitrary and the model is learned from data, it does not encapsulate our initial beliefs in the way one might expect.

We argue that there are two core reasons why inductive biases can be important for VAEs: (a) standard VAEs can fail to encourage, and even prohibit, desired structure in the *representations* we learn; and (b) standard VAEs do not allow one to impart prior information or desired topological characteristic into the *generative model*.

Considering the former, one often has some a priori desired characteristics, or constraints, on the representations learned [Bengio et al., 2013]. For example, sparse features can be desirable because they can improve data efficiency [Yip and Sussman, 1997], and provide robustness to noise [Wright et al., 2009, Ahmad and Scheinkman, 2019] and attacks [Gopalakrishnan et al., 2018]. In other settings one might desire clustered [Jiang et al., 2017], disentangled [Ansari and Soh, 2019, Kim and Mnih, 2018, Higgins et al., 2018] or hierarchical representations [Song and Li, 2013, Sønderby et al., 2016, Zhao et al., 2017]. The KL-divergence term in Equation (3.1) regularizes the encoding distribution towards the prior and, as a standard Gaussian distribution typically does not exhibit our desired characteristics, this regularization can significantly hinder our ability to learn representations with the desired properties.

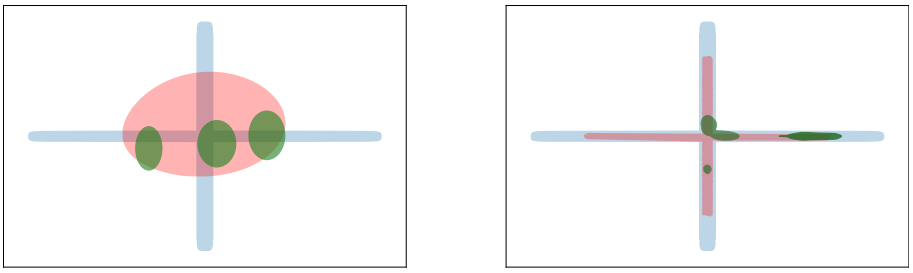
Not only can this be problematic at an individual sample level, it can cause even more pronounced issues at the *population level*: desired structural characteristics of our representations often relate to the pushforward distribution of the data in the latent space, $q_\phi(z) := \mathbb{E}_{\mathrm{p_{data}(x)}}[q_\phi(z|x)]$, which is both difficult to control and only implicitly regularized to the prior [Hoffman and Johnson, 2016].



(a) Data (b) VAE

Figure 3.2: VAE learned generative distribution $\mathbb{E}_{p(z)}[p_\theta(x|z)]$ for mixture data.

Inductive biases can also be essential to the generation quality of VAEs: because the generation process of standard VAEs is essentially pushing-forward the Gaussian prior on $\mathcal{Z}$ to data space $\mathcal{X}$ by a 'smooth' decoder, there is an underlying inductive bias



(a) Directly replacing $p(z)$ (b) InteL-VAE

Figure 3.3: Prior-encoder mismatch. We train (a) a VAE with a sparse prior and (b) an InteL-VAE with a sparse inductive bias on 2 dimensional sparse data. Figure shows target latent distribution $p(z)$ (blue), learned variational embeddings $q_\phi(z|x)$ of exemplar data (green), and data pushforward $q_\phi(z)$ (red shadow) for each method. Simply replacing the prior does not help the VAE match prior structure on either a per-sample or population level, whereas InteL-VAE produces an effective match.

that standard VAEs prefer sample distributions with similar topology structures to Gaussians. As a result, VAEs can perform poorly when the data manifold exhibits certain different topological properties [Caterini et al., 2020]. For example, they can struggle when data is clustered into unconnected components as shown in Figure 3.2, or when data is not simply-connected. This renders learning effective mappings using finite datasets and conventional architectures (potentially prohibitively) difficult. In particular, it can necessitate large Lipschitz constants in the decoder, causing knock-on issues like unstable training and brittle models [Scaman and Virmaux, 2018], as well as posterior collapse [van den Oord et al., 2017, Alemi et al., 2018]. In short, the Gaussian prior of a standard VAE can induce fundamental topological differences to the true data distribution [Falorsi et al., 2018, Shi et al., 2020].

3.3 Shortfalls of VAEs with non-Gaussian priors

Though directly replacing the Gaussian prior with a different prior sounds like a simple solution, effectively introducing inductive biases can, unfortunately, be more complicated.

Firstly, the only influence of the prior during training is as a regularizer on the encoder through the $D_{\mathrm{KL}}\left(q_\phi(z|x) \parallel p(z)\right)$ term. This regularization is always competing with the need for effective reconstructions and only has an indirect influence on $q_\phi(z)$. As such, simply replacing the prior can be an ineffective way of inducing desired structure at the population level [Mathieu et al., 2019b], particularly if $p(z)$ is a complex

distribution that it is difficult to fit (see, e.g., Figure 3.3a). Mismatches between $q_\phi(z)$ and $p(z)$ can also have further deleterious effects on the learned generative model: the former represents the distribution of the data in latent space during training, while the latter is what is used by the learned generative model, leading to unrepresentative generations if there is mismatch.

Secondly, it can be extremely difficult to construct appropriate encoder mappings and distributions for non-Gaussian priors. While the typical choice of a mean-field Gaussian for the encoder distribution is simple, easy to train, and often effective for Gaussian priors, it is often inappropriate for other choices of prior. For example, in Figure 3.3, we consider replacement with a sparse prior. A VAE with a Gaussian encoder struggles to encode points in a manner that even remotely matches the prior. One might suggest replacing the encoder distribution as well, but this has its own issues, most notably that other distributions can be hard to effectively parameterize or train. In particular, the form of the required encoding noise might become heavily spatially variant; in our sparse example, the noise must be elongated in a particular direction depending on where the mean embedding is. If the prior has constraints or topological properties distinct from the data, it can even be difficult to learn a mean encoder mapping that respects these, due to the continuous nature of neural networks.

3.4 The InteL–VAE framework

To solve the issues highlighted in the previous section, and provide a principled and effective method for adding inductive biases to VAEs, we propose *Intermediary Latent Space VAEs* (InteL-VAEs). The key idea behind InteL-VAEs is to introduce an *intermediary* set of latent variables $y \in \mathcal{Y}$, used as a stepping stone in the construction of the *representation* $z \in \mathcal{Z}$. Data is initially encoded in $\mathcal{Y}$ using a conventional VAE encoder (e.g. a mean-field Gaussian) before being passed through a *non-stochastic* mapping $g_\psi : \mathcal{Y} \mapsto \mathcal{Z}$ that incorporates our desired inductive biases and which can be trained, if needed, through its parameters ψ. The prior is defined on $\mathcal{Y}$ and taken to be a standard Gaussian, $p(y) = \mathcal{N}(y; 0, I)$, while our representations, $z = g_\psi(y)$, correspond to a pushforward of y. By first encoding datapoints to y, rather than z directly, we can deal with all the encoder and prior stochasticity in this first, well-behaved, latent space, while maintaining z as our representation and using it

for the decoder $p_\theta(x|z)$. In principle, g_ψ can be any arbitrary parametric (or fixed) mapping, including non-differentiable or even discontinuous functions. However, to allow for reparameterized gradient estimators [Kingma and Welling, 2014, Rezende and Mohamed, 2015], we will restrict ourselves to g_ψ that are sub-differentiable (and thus continuous) with respect to both their inputs and parameters. Note that setting g_ψ to the identity mapping recovers a conventional VAE.

As shown in Figure 5.2.1, the auto-encoding process is now $\mathcal{X} \xrightarrow{q_\phi} \mathcal{Y} \xrightarrow{g_\psi} \mathcal{Z} \xrightarrow{p_\theta} \mathcal{X}$. This three-step process no longer unambiguously fits into the encoder-decoder terminology of the standard VAE and permits a variety of interpretations; for now we take the convention of calling $q_\phi(y|x)$ the encoder and $p_\theta(x|z)$ the decoder, but also discuss some alternative interpretations below. We emphasize here that these no longer respectively match up with our representation model—which corresponds to passing an input into the encoder and then mapping the resulting encoding using g_ψ—and our generative model—which corresponds to $\mathcal{N}(y; 0, I)p_\theta(x|z = g_\psi(y))$, such that we sample a y from the prior and then pass this through through g_ψ and the decoder in turn.

The mapping g_ψ introduces inductive biases into *both* the generative model and our representations by imposing a particular form on z, such as the spherical structure enforced in Figure 5.2.1 (see also Section 3.6). It can be viewed as a *shared module* between them, ensuring congruence between the two. This congruence allows us to more directly introduce inductive biases through careful construction of g_ψ, without complicating the process of learning an effective inference network. In particular, because $\mathcal{Y}$ is treated as our latent space for the purposes of training, we sidestep the inference issues that non-Gaussian priors usually cause. Moreover, because all samples must explicitly pass through g_ψ during both training and generation, we can more directly ensure the desired structure is enforced without causing a mismatch in the latent distribution between training and deployment.

Training As with standard VAEs, training of an InteL-VAE is done by maximizing a variational lower bound (ELBO) on the log evidence, which we denote $\mathcal{L}_\mathcal{Y}$. Most simply, we have

$$\begin{aligned}\log p_{\theta,\psi}(x) &:= \log\left(\mathbb{E}_{p(y)}\left[p_\theta(x|g_\psi(y))\right]\right) = \log\left(\mathbb{E}_{q_\phi(y|x)}\left[\frac{p_\theta(x|g_\psi(y))\mathcal{N}(y;0,I)}{q_\phi(y|x)}\right]\right) \\ &\geq \mathbb{E}_{q_\phi(y|x)}[\log p_\theta(x|g_\psi(y))] - D_{\mathrm{KL}}\left(q_\phi(y|x) \parallel \mathcal{N}(y;0,I)\right) =: \mathcal{L}_\mathcal{Y}(x,\theta,\phi,\psi).\end{aligned} \tag{3.2}$$

Note that the regularization is on y, but our representation corresponds to $z = g_\psi(y)$. Training corresponds to the optimization $\arg\max_{\theta,\phi,\psi} \mathbb{E}_{x\sim \mathrm{p_{data}(x)}} [\mathcal{L}_\mathcal{Y}(x, \theta, \phi, \psi)]$, which can be performed using stochastic gradient ascent with reparameterized gradients in the standard manner. Although inductive biases are introduced, the calculation, and optimization, of $\mathcal{L}_\mathcal{Y}$ is thus equivalent to the standard ELBO. In particular, parameterizing $q_\phi(y|x)$ with a Gaussian distribution still yields an analytical $D_{\mathrm{KL}}\left(q_\phi(y|x) \parallel \mathcal{N}(y; 0, I)\right)$ term.

Alternative Interpretations It is interesting to note that our representation, $g_\psi(y)$, only appears in the context of the decoder in this training objective. As such, we see that an important alternative interpretation of InteL-VAEs is to consider g_ψ as being a customized first layer in the decoder, and our test–time representations as partial decodings of the latents y. This viewpoint allows it to be applied with more general bounds and VAE variants (e.g. Burda et al. [2016], Le et al. [2018], Maddison et al. [2017], Naesseth et al. [2018], Zhao et al. [2019]), as it requires only a carefully customized decoder architecture during training and an adjusted mechanism for constructing representations at test–time.

Yet another interpretation is to think about InteL-VAEs as implicitly defining a conventional VAE with latents z, but where both the non-Gaussian prior, $p_\psi(z)$, and our encoder distribution, $q_{\phi,\psi}(z|x)$, are themselves defined implicitly as pushforwards along g_ψ, which acts as a shared module that instills a natural compatibility between the two. Formally we have the following theorem.

Theorem 3.4.1. *Let $p_\psi(z)$ and $q_{\phi,\psi}(z|x)$ represent the respective pushforward distributions of $\mathcal{N}(0, I)$ and $q_\phi(y|x)$ induced by the mapping $g_\psi : \mathcal{Y} \mapsto \mathcal{Z}$. The following holds for all measurable g_ψ:*

$$D_{KL}\left(q_{\phi,\psi}(z|x) \parallel p_\psi(z)\right) \leq D_{KL}\left(q_\phi(y|x) \parallel \mathcal{N}(y; 0, I)\right). \tag{3.3}$$

If g_ψ is also an invertible function then the above becomes an equality and $\mathcal{L}_\mathcal{Y}$ equals the standard ELBO on the space of $\mathcal{Z}$ as follows

$$\mathcal{L}_\mathcal{Y}(x, \theta, \phi, \psi) = \mathbb{E}_{q_{\phi,\psi}(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_{\phi,\psi}(z|x) \parallel p_\psi(z)\right). \tag{3.4}$$

The proof is given in Section 3.A. Here, (3.3) shows that the divergence in our representation space $\mathcal{Z}$ is never more than that in $\mathcal{Y}$, or equivalently that the implied ELBO on the space of $\mathcal{Z}$ is always at least as tight as that on $\mathcal{Y}$; (3.4) shows they are

exactly equal if g_ψ is invertible. As the magnitude of $D_{\mathrm{KL}}\left(q_\phi(y|x) \parallel \mathcal{N}(y; 0, I)\right)$ in an InteL-VAE will remain comparable to the KL divergence in a standard Gaussian prior VAE setup, this, in turn, ensures that $D_{\mathrm{KL}}\left(q_{\phi,\psi}(z|x) \parallel p_\psi(z)\right)$ does not become overly large. This is in stark contrast to the conventional non-Gaussian prior setup, where it can be difficult to avoid $D_{\mathrm{KL}}\left(q_\phi(z|x) \parallel p_\psi(z)\right)$ exploding without undermining reconstruction [Mathieu et al., 2019b]. The intuition here is that having the stochasticity in the encoder *before* it is passed through g_ψ ensures that the form of the noise in the embedding is inherently appropriate for the space: the same mapping is used to warp this noise as to define the generative model in the first place. For example, when g_ψ is a sparse mapping, the Gaussian noise in $q_\phi(y|x)$ will be compressed to a sparse subspace by g_ψ, leading to a sparse variational posterior $q_{\phi,\psi}(z|x)$ as shown in Figure 3.3b. In particular, $q_\phi(y|x)$ does not need to learn any complex spatial variations that result from properties of $\mathcal{Z}$. In turn, InteL-VAEs further alleviate issues of mismatch between $p_\psi(z)$ and $q_{\phi,\psi}(z)$.

Further Benefits A key benefit of InteL-VAEs is that the extracted features are *guaranteed* to have the desired structure, such as sparsity or multiply-connectivity. Take the spherical case for example, all extracted features $g_\psi(\mu_\phi(x))$ lie within a small neighborhood of the unit sphere. By comparison, methods based on training loss modifications, e.g. Mathieu et al. [2019b], often fail to generate features with the targeted properties.

A more subtle advantage is that we do not need to explicitly specify $p_\psi(z)$. This can be extremely helpful when we want to specify complex inductive biases: designing a non-stochastic mapping is typically much easier than a density function, particularly for complex spaces. Further, this can make it much easier to parameterize and learn aspects of $p_\psi(z)$ in a data-driven manner (see e.g. Sec. 3.6.3).

3.5 Related work

Inductive biases There is much prior work on introducing human knowledge to deep learning models by structural design, such as CNNs [LeCun et al., 1989], RNNs [Hochreiter and Schmidhuber, 1997] and transformers [Vaswani et al., 2017]. However, most of these designs are on the *sample* level, utilizing low–level information such as transformation invariances or internal correlations in each sample. By contrast,

InteL-VAEs provide a convenient way to incorporate *population* level knowledge—information about the global properties of data distributions can be effectively utilized.

Non-Gaussian priors There is an abundance of prior work utilizing non-Gaussian priors to improve the fit and generation capabilities of VAEs, including MoG priors [Dilokthanakul et al., 2016, Shi et al., 2020], sparse priors [Mathieu et al., 2019b, Tonolini et al., 2020, Barello et al., 2018], Gaussian-process priors [Casale et al., 2018] and autoregressive priors [Razavi et al., 2019, van den Oord et al., 2017]. However, these methods often require specialized algorithms to train and are primarily applicable only to specific kinds of data. Moreover, as we have explained, changing the prior alone often provides insufficient pressure on its own to induce the desired characteristics. Others have proposed non-Gaussian priors to reduce the prior-posterior gap, such as Vamp-VAE [Tomczak and Welling, 2018] and LARS [Bauer and Mnih, 2019], but these are tangential to our inductive bias aims.

Non-Euclidean latents A related line of work has focused on non-Euclidean latent spaces. For instance Davidson et al. [2018a] leveraged a von Mises-Fisher distribution on a hyperspherical latent space, Falorsi et al. [2018] endowed the latent space with a SO(3) group structure, and Mathieu et al. [2019a], Ovinnikov [2019], Nagano et al. [2019] with hyperbolic geometry. Other spaces like product of constant curvature spaces [Skopek et al., 2019] and embedded manifolds [Rey et al., 2019] have also been considered. However, these works generally require careful design and training.

Normalizing flows Our use of a non-stochastic mapping shares some interesting links to normalizing flows (NFs) [Rezende and Mohamed, 2015, Papamakarios et al., 2019, Grathwohl et al., 2018, Dinh et al., 2017, Huang et al., 2018, Papamakarios et al., 2018]. Indeed a NF would be a valid choice for g_ψ, albeit an unlikely one due to their architectural constraints. However, unlike previous use of NFs in VAEs, our g_ψ is crucially *shared* between the generative and representational models, rather than just being used in the encoder, while the KL divergence in our framework is taken before, not after, the mapping. Moreover, the underlying motivation, and type of mapping typically used, differs substantially: our mapping is used to introduce inductive biases, not purely to improve inference. Our mapping is also more general than a NF (e.g. it need not be invertible) and does not introduce additional constraints or computational issues.

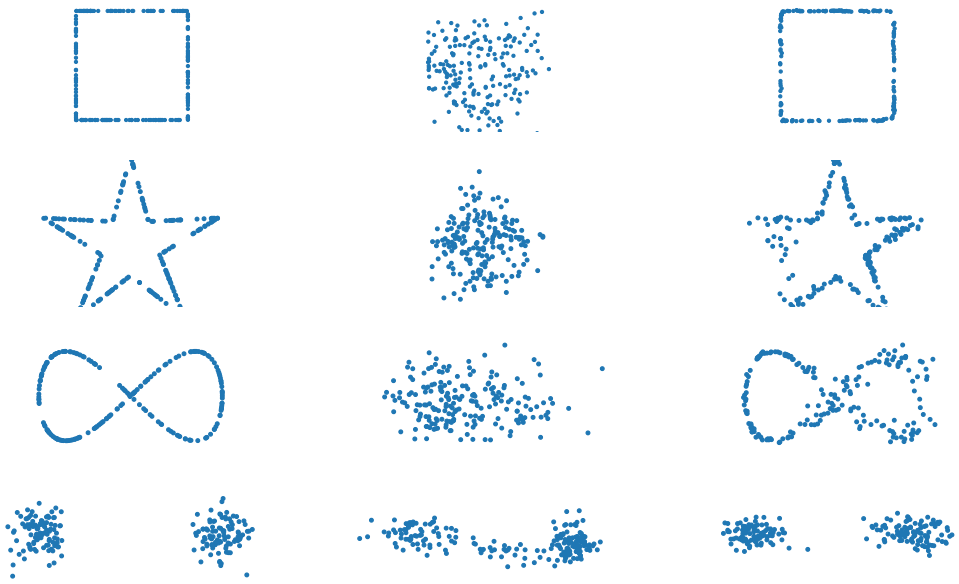
3.6 Specific realizations of the InteL-VAE framework

We now present several novel example InteL-VAEs, introducing various inductive biases through different choices of g_ψ. We will start with artificial, but surprisingly challenging, examples where some precise topological properties of the target distributions are known, incorporating them directly through a fixed g_ψ. We will then move onto experiments where we impose a fixed clustering inductive bias when training on image data, allowing us to learn InteL-VAEs that account effectively for multi-modality in the data distribution. Finally, we consider the example of learning sparse representations of high–dimensional data. Here we will see that it is imperative to exploit the ability of InteL-VAEs to learn aspects of g_ψ during training, providing a flexible inductive bias framework, rather than a pre-fixed mapping. By comparing InteL-VAEs with strong baselines, we show that InteL-VAEs are effective in introducing these desired inductive biases, and consequently both improve generation quality and learn better data representations for downstream tasks. One note of particular importance is that we find that InteL-VAEs provide state-of-the-art performance for learning sparse VAE representations. A further example of using InteL-VAEs to learn hierarchical representations is presented in Section 3.B, while full details on the various examples are given in Section 3.C.

3.6.1 Multiple–connectivity

Data is often most naturally described on non-Euclidean spaces such as circles, e.g. wind directions [Mardia and Jupp, 2000], and other multiply-connected shapes, e.g. holes in disease databases [Liu et al., 1997]. For reasons previously explained in Section 4.2, standard VAEs cannot practically model such topologies, which prevents them from learning generative models which match even the simplest data distributions with non-trivial topological structures, as shown in Figure 3.4b.

Luckily, by designing g_ψ to map the Gaussian prior to a simple representative distribution in a topological class, we can easily equip InteL-VAEs with the knowledge to approximate any data distributions with similar topological properties. Specifically, by defining g_ψ as the orthogonal projection to $\mathbb{S}^1$, $g_\psi(z) = z/(||z||_2 + \epsilon)$, we map the Gaussian prior approximately to a uniform distribution to $\mathbb{S}^1$, where ϵ is a small positive constant to ensure the continuity of g_ψ near the origin. From Rows 1 and 2



(a) Data (b) VAE (c) InteL-VAE

Figure 3.4: Training data and samples from learned generative models of vanilla-VAE and InteL-VAE for multiply-connected and clustered distributions. InteL-VAE uses [Rows 1,2] circular prior with one hole, [Row 3] multiply-connected prior with two holes, and [Row 4] clustered prior. Vamp-VAE behaves similarly to a vanilla VAE; its results are presented in Figure 3.4.

of Figure 3.4, we find that this inductive bias gives InteL-VAEs the ability to learn various distributions with a hole. We can add further holes by simply 'gluing' point pairs. For example, for two holes we can use

$$g_2(y) = \text{Concat}\left(g_1(y)_{[:,1]},\ g_1(y)_{[:,2]}\sqrt{(4/3 - (1 - |g_1(y)_{[:,1]}|)^2)} - 1/\sqrt{3}\right), \qquad (3.5)$$

which first map y to approximately S^1, and then glues $(0,1)$ and $(0,-1)$ together to create new holes (see Figure 3.C.1 for an illustration). Furthermore, we can continue to glue points together to achieve a higher number of holes h, and thus more complex connectivity. Row 3 of Figure 3.4 gives an example of learning an infinity sign by introducing a 'two-hole' inductive bias.

Compared with vanilla-VAE and Vamp-VAE, which try to find a convex hull for real data distributions, InteL-VAEs can deal with distributions with highly non-convex and very non-smooth supports (see Figure 3.4 and Section 3.C.1). We emphasize here that our inductive bias does not contain the information about the precise shape of the data, only the number of holes. We thus see that InteL-VAEs can provide substantial improvements in performance by incorporating only basic prior information about the

Method	Data	VAE	GM-VAE	MoG-VAE	Vamp-VAE	Flow	InteL-VAE
Unc.(%)	0.2 ± 0.1	2.5 ± 0.4	3.5 ± 1.8	4.5 ± 0.8	2.4 ± 0.3	16.2 ± 2.1	**0.9 ± 0.8**
'1' prop.(%)	50.0 ± 0.2	48.8 ± 0.2	48.1 ± 0.3	47.7 ± 0.4	48.8 ± 0.1	42.5 ± 1.0	**49.5 ± 0.4**

Table 3.1: Quantitative results on **MNIST-01**. Unc(ertainty) is the proportion of images whose labels are 'indistinguishable' to the pre-trained classifier, defined as having prediction confidence $< 80\%$. '1' prop(ortion) is the proportion of images classified as '1'.

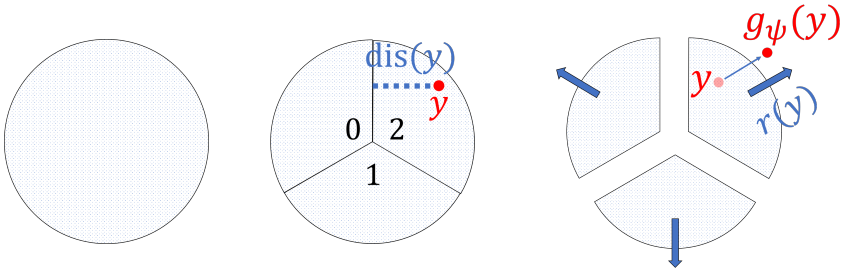


Figure 3.5: Illustration of clustered mapping where $K = 3$. The circle represents a density isoline of a Gaussian. Note that not all points in the sector are moved equally: points close to the boundaries between sectors are moved less, with points on the boundary themselves not moved at all.

topological properties of the data, which point out a way to approximate distributions on more complex structures, such as linear groups [Gupta and Mishra, 2018].

3.6.2 Multi–modality

Many real-world datasets exhibit multi-modality. For example, data with distinct classes are often naturally clustered into (nearly) disconnected components representing each class. However, vanilla VAEs generally fail to fit multi-modal data due to the topological issues explained in Section 4.2. Previous work [Johnson et al., 2017, Mathieu et al., 2019b] has thus proposed the use of a multi-modal prior, such as a mixture of Gaussian (MoG) distribution, so as to capture all components of the data. Nonetheless, VAEs with such priors often still struggle to model multi-modal data because of mismatch between $q_\phi(z)$ and $p(z)$ or training instability issues.

We tackle this problem by using a mapping g_ψ which contains a clustering inductive bias. The high-level idea is to design a mapping g_ψ with a localized high Lipschitz constant that 'splits' the continuous Gaussian distribution into K disconnected parts and then pushes them away from each other. In particular, we split $\mathcal{Y}$ it into K

equally sized sectors using its first two dimensions (noting it is not needed to split on all dimensions to form clusters), as shown in Figure 3.5. For any point y, we can easily get the center direction $\mathrm{r}(y)$ of the sector that y belongs to and the distance $\mathrm{dis}(y)$ between y and the sector boundary. Then we define $g_\psi(y)$ as:

$$g_\psi(y) = y + c_1 \mathrm{dis}(y)^{c_2} \mathrm{r}(y), \tag{3.6}$$

where c_1 and c_2 are empirical constants. We can see that although g_ψ has very different function on different sectors, it is still continuous on the whole plane with $g_\psi(y) = y$ on sector boundaries, which is desirable for gradient-based training. See Section 3.C.2 for more details.

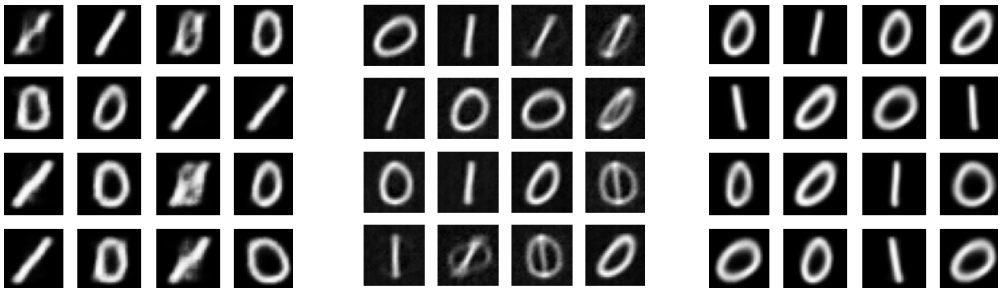
To assess the performance of our approach, we first consider a simple 2-component MoG synthetic dataset in the last row of Figure 3.4. We see that the vanilla VAE fails to learn a clustered distribution that fits the data, while the InteL-VAE sorts this issue and fits the data well.

To provide a more real-world example, we train an InteL-VAE and a variety of baselines on the **MNIST** dataset, comparing the generation quality of the learned models using the FID score [Heusel et al., 2017] in Table 3.2. We find that the GM-VAE [Dilokthanakul et al., 2016] and MoG-VAE (VAE with a fixed MoG prior) achieve performance gains by using non-Gaussian priors. The Vamp-VAE [Tomczak and Welling, 2018] and a VAE with a Sylvester Normalizing Flow [Berg et al., 2018] encoder provide further gains by making the prior and encoder distributions more flexible respectively. However, the InteL-VAE comfortably outperforms all of them.

Method	FID Score (↓)
VAE	42.0 ± 1.1
GM-VAE	41.0 ± 4.7
MoG-VAE	41.2 ± 3.3
Vamp-VAE	38.8 ± 2.4
VAE with Sylvester NF	35.0 ± 0.9
InteL-VAE	32.2 ± 1.5

Table 3.2: Generation quality on MNIST. Shown is mean FID score (lower better) ± standard deviation over 10 runs.

To gain insight into how InteL-VAEs achieve superior generation quality, we perform analysis on a simplified setting where we select only the '0' and '1' digits from the **MNIST** dataset to form a strongly clustered dataset, **MNIST-01**. We further decrease the latent dimension to 1 to make the problem more challenging. Fig. 3.6 shows that here the vanilla VAE generates some samples which look like interpolations between '0' and '1', meaning that it still tries to learn a connected distribution containing '0' and '1'. Further, the general generation quality is poor, with blurred



(a) VAE (b) MoG-VAE (c) InteL-VAE

Figure 3.6: Generated samples for **MNIST-01**.

images and a lack of diversity in generated samples (e.g. all the '1's have the same slant). Despite using a clustered prior, the MoG-VAE still produces unwanted interpolations between the classes. By contrast, InteL-VAE generates digits that are unambiguous and crisper.

To quantify these results, we further train a logistic classifier on **MNIST-01** and use it to classify images generated by each method. For each method, we calculate the proportion of samples produced by the generative model that are assigned to each class by this pre-trained classifier, as well as the proportion of samples for which the classifier is uncertain. From Table 3.1 we see that InteL-VAE significantly outperforms its competitors in the ability to generate balanced and unambiguous digits. To extend this example further, and show the ability of InteL-VAEs to learn aspects of g_ψ during training, we further consider parameterizing and then learning the relative size of the clusters. Table 3.3 shows that this can be successfully learned by InteL-VAEs on **MNIST-01**.

True Prop.	Learned Prop.
0.5	0.47 ± 0.01
0.4	0.36 ± 0.10
0.25	0.25 ± 0.08
0.2	0.16 ± 0.11
0	0.02 ± 0.01

Table 3.3: Learned proportions of '0's on **MNIST-01** for different ground truths. Error bars are std. dev. from 10 runs.

3.6.3 Sparsity

With only few activated dimensions, sparse features are often well-suited to data efficiency on downstream tasks [Huang and Aviyente, 2006], in addition to being naturally easier to visualize and manipulate than dense features [Ng et al., 2011]. However, existing VAE models for sparse representations trade off generation quality

to achieve this sparsity [Mathieu et al., 2019b, Tonolini et al., 2020, Barello et al., 2018]. Here, we show that InteL-VAEs can instead *simultaneously* increase feature sparsity and generation quality. Moreover, they are able to achieve state-of-the-art scores on sparsity metrics.

Compared with our previous examples, the g_ψ here needs to be more flexible so that it can learn to map points in a data-specific way and induce sparsity without unduly harming reconstruction. To achieve this, we use the simple form for the mapping: $g_\psi(y) = y \odot \text{DS}_\psi(y)$, where $\odot$ is pointwise multiplication, and DS is a 'dimension selector' network that selects dimensions to deactivate given y. DS outputs values between $[0, 1]$ for each dimension, with 0 being fully deactivated and 1 fully activated; the more dimensions we deactivate, the sparser the representation. By learning DS during training, this setup allows us to learn a sparse representation in a data-driven manner. To control the degree of sparsity, we add a sparsity regularizer, $\mathcal{L}_{sp}$, to the ELBO with weighting parameter γ (higher γ corresponds to more sparsity). Namely, we optimize $\mathcal{L}_{\mathcal{Y}}(\theta, \phi, \psi) + \gamma\, \mathcal{L}_{sp}(\phi, \psi)$, where

$$\mathcal{L}_{sp}(\phi, \psi) := \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^{M}(H\left(DS(y_i)\right)) - H\left(\frac{1}{M}\sum_{i=1}^{M} DS(y_i)\right)\right], \tag{3.7}$$

$H(v) = -\sum_i (v_i/\|v\|_1) \log (v_i/\|v\|_1)$ is the normalized entropy of an positive vector v, and the expectation is over drawing a minibatch of samples $x_1, \ldots, x_M$ and then sampling each corresponding $y_i \sim q_\phi(\cdot|x = x_i)$. $\mathcal{L}_{sp}$ encourages DS to deactivate more dimensions, while also encouraging diversity in which dimensions are activated for different data points, improving utilization of the latent space. Please see Section 3.C.3 for more details and intuitions. Initial qualitative results are shown in Figure 3.8, where we see that our InteL-VAE is able to learn sparse and intuitive representations.

To quantitatively assess the ability of our approach to yield sparse representations and good quality generations, we compare against vanilla VAEs, the specially customized sparse-VAE of Tonolini et al. [2020], and the sparse version of Mathieu et al. [2019b] (DD) on **Fashion-MNIST** [Xiao et al., 2017] and **MNIST**. As shown in Fig. 3.7 (*left*), we find that InteL-VAEs increase sparsity of the representations—measured by the **Hoyer metric** [Hurley and Rickard, 2009]—while increasing generative sample quality at the same time. Indeed, the FID score obtained by InteL-VAE outperforms the vanilla VAE when $\gamma < 3.0$, while the sparsity score substantially increases with γ, reaching extremely high levels. By comparison, DD significantly degrades generation quality and only provides a more modest increase in sparsity,

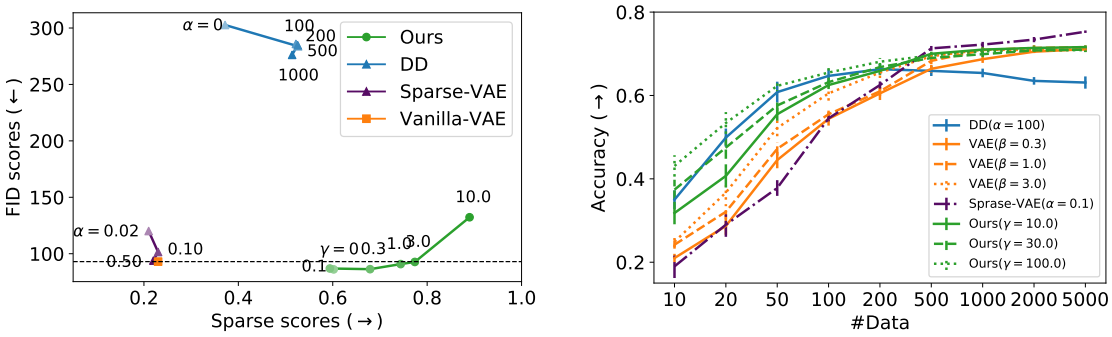


Figure 3.7: Results on **Fashion-MNIST**. The *left* figure shows FID and sparsity scores. Lower FID scores (↓) represent better sample quality while higher sparse scores (→) indicate sparser features. The *right* figure shows the performance of sparse features from InteL-VAE on downstream classification tasks. See Section 3.C.3 for details and results for **MNIST**.

Method	FID (↓)	Sparsity (↑)
VAE	68.6±1.1	0.22±0.01
Vamp-VAE	67.5±1.1	0.22±0.01
VAE with Sylvester NF	66.3±0.4	0.22±0.01
Sparse-VAE ($\alpha = 0.01$)	328±10.1	0.25±0.01
Sparse-VAE ($\alpha = 0.2$)	337±8.1	0.28±0.01
InteL-VAE ($\gamma = 30$)	64.9±0.4	0.25±0.01
InteL-VAE ($\gamma = 70$)	68.0±0.6	0.46±0.02

Table 3.4: Generation results on CelebA.

while its sparsity also drops if the regularization coefficient is set too high. The level of sparsity achieved by sparse-VAEs was substantially less than both DD and InteL-VAEs.

To further evaluate the quality of the learned features for downstream tasks, we trained a classifier to predict class labels from the latent representations. For this, we choose a random forest [Breiman, 2001] with maximum depth 4 as it is well-suited for sparse features. We vary the size of training data given to the classifier to measure the data efficiency of each model. Figure 3.7 (*right*) shows that InteL-VAE typically outperforms other the models, especially in few-shot scenarios.

Finally, to verify InteL-VAE's effectiveness on larger and higher-resolution datasets, we also make comparisons on **CelebA** [Liu et al., 2015]. From Table 3.4, we can see that InteL-VAE increase sparse scores to 0.46 without sacrificing generation quality. By

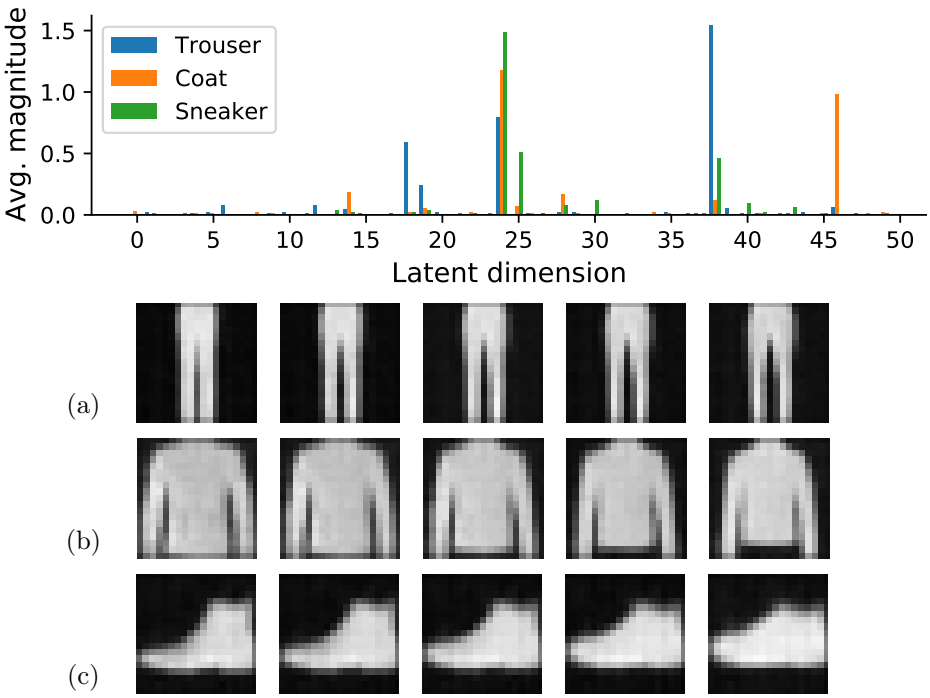


Figure 3.8: Qualitative evaluation of sparsity. [Top] Average magnitude of each latent dimension for three example classes in **Fashion-MNIST**; less than 10% dimensions are activated for each class. [Bottom] Activated dimensions are different between classes: (a-c) show the results of separately manipulating an activated dimension for each class. (a) Trouser separation (Dim 18). (b) Coat length (Dim 46). (c) Shoe style (formal/sport, Dim 25).

comparison, the maximal sparse score that sparse-VAE gets is 0.30, with unacceptable sample quality. Interestingly, InteL-VAEs with low regulation γ achieved particularly good generative sample quality, outperforming even the Vamp-VAE and a VAE with a Sylvester NF encoder.

Conclusions In this paper, we proposed InteL-VAEs, a general schema for incorporating inductive biases into VAEs. Experiments show that InteL-VAEs can both provide representations with desired properties and improve generation quality, outperforming a variety of baselines such as directly changing the prior. This is achieved while maintaining the simplicity and stability of standard VAEs.

Appendix

3.A Proofs

Theorem 3.4.1. *Let $p_\psi(z)$ and $q_{\phi,\psi}(z|x)$ represent the respective pushforward distributions of $\mathcal{N}(0, I)$ and $q_\phi(y|x)$ induced by the mapping $g_\psi : \mathcal{Y} \mapsto \mathcal{Z}$. The following holds for all measurable g_ψ:*

$$D_{KL}\left(q_{\phi,\psi}(z|x) \parallel p_\psi(z)\right) \leq D_{KL}\left(q_\phi(y|x) \parallel \mathcal{N}(y; 0, I)\right). \tag{3.3}$$

If g_ψ is also an invertible function then the above becomes an equality and $\mathcal{L}_\mathcal{Y}$ equals the standard ELBO on the space of $\mathcal{Z}$ as follows

$$\mathcal{L}_\mathcal{Y}(x, \theta, \phi, \psi) = \mathbb{E}_{q_{\phi,\psi}(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_{\phi,\psi}(z|x) \parallel p_\psi(z)\right). \tag{3.4}$$

Proof. We first prove the inequality from Equation (3.3), then we show that Equation (3.3) is actually an equality when g_ψ is invertible, and finally we prove that the reconstruction term is unchanged by g_ψ.

Let us denote by $\mathcal{F}$ and $\mathcal{G}$ the sigma-algebras of respectively $\mathcal{Y}$ and $\mathcal{Z}$, and we have by construction a measurable map $g_\psi : (\mathcal{Y}, \mathcal{F}) \to (\mathcal{Z}, \mathcal{G})$. We can actually define the measurable space $(\mathcal{Z}, \mathcal{G})$ as the image of $(\mathcal{Y}, \mathcal{F})$ by g_ψ, then g_ψ is automatically both surjective and measurable.[1] We also assume that there exists a measure on $\mathcal{Y}$, which we denote ξ, and denote with ν the corresponding pushforward measure by g_ψ on $\mathcal{Z}$. We further have $\nu(A) = \xi(g_\psi^{-1}(A))$ for any $A \in \mathcal{G}$.[2]

We start by proving Equation (3.3), where the Kullback-Leibler (KL) divergence between the two pushforward measures[3] $q_{\phi,\psi} \triangleq q_\phi \circ g_\psi^{-1}$ and $p_\psi \triangleq p \circ g_\psi^{-1}$ is upper

[1] We recall that g_ψ is said to be measurable if and only if for any $A \in \mathcal{G}$, $g_\psi^{-1}(A) \in \mathcal{F}$.

[2] The notation $g_\psi^{-1}(A)$ does not imply that g_ψ is invertible, but denotes the preimage of A which is defined as $g_\psi^{-1}(A) = \{y \in \mathcal{Y} \mid g_\psi(y) \in A\}$.

[3] We denote the pushforward of a probability measure χ along a map g by $\chi \circ g^{-1}$.

bounded by $D_{\mathrm{KL}}\left(q_\phi(y|x) \parallel p(y)\right)$, where here we have $p(y) = \mathcal{N}(y; 0, I)$ but we will use p as a convenient shorthand. At a high-level, we essentially have that Equation (3.3) follows directly the data processing inequality [Sason, 2019] with a deterministic kernel $z = g_\psi(y)$. Nonetheless, we develop in what follows a proof which additionally gives sufficient conditions for when this inequality becomes non-strict. We can assume that $D_{\mathrm{KL}}\left(q_\phi(y|x) \parallel \mathcal{N}(y; 0, I)\right)$ is finite, as otherwise the result is trivially true, which in turn implies $q_\phi \ll p$.[4] For any $A \in \mathcal{G}$, we have that if $p_\psi(A) = p \circ g_\psi^{-1}(A) = p(g_\psi^{-1}(A)) = 0$ then this implies $q_\phi(g_\psi^{-1}(A)) = q_\phi \circ g_\psi^{-1}(A) = q_{\phi,\psi}(A) = 0$. As such, we have that $q_{\phi,\psi} \ll p_\psi$ and so the $D_{\mathrm{KL}}\left(q_{\phi,\psi}(z|x) \parallel p_\psi(z)\right)$ is also defined.

Our next significant step is to show that

$$\mathbb{E}_{p(y)}\left[\frac{q_\phi}{p} \,\middle|\, \sigma(g_\psi)\right] = \frac{q_\phi \circ g_\psi^{-1}}{p \circ g_\psi^{-1}} \circ g_\psi, \tag{3.8}$$

where $\sigma(g_\psi)$ denotes the sigma-algebra generated by the function g_ψ. To do this, let $h : (\mathcal{Z}, \mathcal{G}) \to (\mathbb{R}_+, \mathcal{B}(\mathbb{R}_+))$ be a measurable function s.t. $\mathbb{E}_{p(y)}\left[\frac{q_\phi}{p} \,\middle|\, \sigma(g_\psi)\right] = h \circ g_\psi$. To show this, we will demonstrate that they lead to equivalent measures when integrated over any arbitrary set $A \in \mathcal{G}$:

$$\begin{aligned}
\int_{\mathcal{Z}} \mathbb{1}_A \, \frac{q_\phi \circ g_\psi^{-1}}{p \circ g_\psi^{-1}} \, p \circ g_\psi^{-1} \, d\nu &= \int_{\mathcal{Z}} \mathbb{1}_A \, q_\phi \circ g_\psi^{-1} \, d\nu = \int_{\mathcal{Z}} \mathbb{1}_A \, d(q_\phi \circ g_\psi^{-1}) \\
&\overset{(a)}{=} \int_{\mathcal{Y}} (\mathbb{1}_A \circ g_\psi) \, dq_\phi = \int_{\mathcal{Y}} (\mathbb{1}_A \circ g_\psi) \, q_\phi \, d\xi \\
&\overset{(b)}{=} \int_{\mathcal{Y}} (\mathbb{1}_A \circ g_\psi) \, \frac{q_\phi}{p} \, p \, d\xi \\
&\overset{(c)}{=} \int_{\mathcal{Y}} (\mathbb{1}_A \circ g_\psi) \, \mathbb{E}_{p(y)}\left[\frac{q_\phi}{p} \,\middle|\, \sigma(g_\psi)\right] \, p \, d\xi \\
&\overset{(d)}{=} \int_{\mathcal{Y}} (\mathbb{1}_A \circ g_\psi) \, (h \circ g_\psi) \, p \, d\xi = \int_{\mathcal{Y}} (\mathbb{1}_A \circ g_\psi) \, (h \circ g_\psi) \, dp \\
&\overset{(e)}{=} \int_{\mathcal{Z}} \mathbb{1}_A \, h \, d(p \circ g_\psi^{-1}) = \int_{\mathcal{Z}} \mathbb{1}_A \, h \, (p \circ g_\psi^{-1}) \, d\nu,
\end{aligned}$$

where we have leveraged the definition of pushforward measures in (a & e); the absolute continuity of q_ϕ w.r.t. p in (b); the conditional expectation definition in (c); and the definition of h in (d). By equating terms, we have that $q_\phi \circ g_\psi^{-1} / p \circ g_\psi^{-1} = h$, almost-surely with respect to $q_\phi \circ g_\psi^{-1}$ and thus that Equation (3.8) is verified.

[4] We denote the absolute continuity of measures with $\ll$, where μ is said to be absolutely continuous w.r.t. ν, i.e. $\mu \ll \nu$, if for any measurable set A, $\nu(A) = 0$ implies $\mu(A) = 0$.

Let us define $f : x \mapsto x \log(x)$, which is strictly convex on $[0, \infty)$ (as it can be prolonged with $f(0) = 0$). We have the following

$$
\begin{aligned}
D_{\mathrm{KL}}\left(q_{\phi,\psi}(z|x) \parallel p_\psi(z)\right) &\overset{(a)}{=} \int_{\mathcal{Z}} \log\left(\frac{q_{\phi,\psi}}{p_\psi}\right) q_{\phi,\psi}\, d\nu \\
&\overset{(b)}{=} \int_{\mathcal{Z}} \log\left(\frac{q_{\phi,\psi}}{p_\psi}\right) \frac{q_{\phi,\psi}}{p_\psi}\, p_\psi\, d\nu \\
&\overset{(c)}{=} \int_{\mathcal{Z}} f\left(\frac{q_{\phi,\psi}}{p_\psi}\right) p_\psi\, d\nu = \int_{\mathcal{Z}} f\left(\frac{q_{\phi,\psi}}{p_\psi}\right) d(p \circ g_\psi^{-1}) \\
&\overset{(d)}{=} \int_{\mathcal{Y}} f\left(\frac{q_{\phi,\psi}}{p_\psi} \circ g_\psi\right) dp = \int_{\mathcal{Y}} f\left(\frac{q_\phi \circ g_\psi^{-1}}{p \circ g_\psi^{-1}} \circ g_\psi\right) p\, d\xi \\
&\overset{(e)}{=} \int_{\mathcal{Y}} f\left(\mathbb{E}_{p(y)}\left[\frac{q_\phi}{p} \,\middle|\, \sigma(g_\psi)\right]\right) p\, d\xi \\
&\overset{(f)}{\leq} \int_{\mathcal{Y}} \mathbb{E}_{p(y)}\left[f\left(\frac{q_\phi}{p}\right) \,\middle|\, \sigma(g_\psi)\right] p\, d\xi \\
&\overset{(g)}{=} \int_{\mathcal{Y}} f\left(\frac{q_\phi}{p}\right) p\, d\xi \\
&\overset{(h)}{=} \int_{\mathcal{Y}} \log\left(\frac{q_\phi}{p}\right) \frac{q_\phi}{p}\, p\, d\xi \\
&\overset{(i)}{=} \mathbb{E}_{q_\phi(y|x)}\left[\log\left(\frac{q_\phi(y|x)}{p(y)}\right)\right] \\
&\overset{(j)}{=} D_{\mathrm{KL}}\left(q_\phi(y|x) \parallel p(y)\right),
\end{aligned}
$$

where we leveraged the definition of the KL divergence in (a & j); the absolute continuity of q_ϕ w.r.t. p in (b & i); the definition of f in (c & h); the definition of the pushforward measure in (d); Equation (3.8) in (e); the conditional Jensen inequality in (f) and the law of total expectation in (g). Note that this proof not only holds for the KL divergence, but for any f-divergences as they are defined as in (b) with f convex.

To prove Equation (3.4), we now need to show that line (f) above becomes an equality when g_ψ is invertible. As f is strictly convex, this happens if and only if $\frac{q_\phi}{p} = \mathbb{E}_{p(y)}\left[\frac{q_\phi}{p} \,\middle|\, \sigma(g_\psi)\right]$. A sufficient condition for this to be true is for $\frac{q_\phi}{p}$ to be measurable w.r.t. $\sigma(g_\psi)$ which is satisfied when $g_\psi : \mathcal{Y} \mapsto \mathcal{Z}$ is invertible as $\sigma(g_\psi) \supseteq \mathcal{F}$, as required. We have thus shown that the KL divergences are equal when using an invertible g_ψ.

For the reconstruction term, we instead have

$$\begin{aligned}\mathbb{E}_{q_\phi(y|x)}[\log p_\theta(x|g_\psi(y))] &= \int_{\mathcal{Y}} \log p_\theta(x|g_\psi(y)) q_\phi(y|x) d\xi \\ &= \int_{\mathcal{Z}} \log p_\theta(x|z) q_{\phi,\psi}(z|x) d\nu \\ &= \mathbb{E}_{q_{\phi,\psi}(z|x)}[\log p_\theta(x|z)].\end{aligned}$$

Equation (3.4) now follows from the fact that both the reconstruction and KL terms are equal.

□

3.B Hierarchical representations

The isotropic Gaussian prior in standard VAEs assumes that representations are independent across dimensions [Kumar et al., 2018]. However, this assumption is often unrealistic [Belghazi et al., 2018, Mathieu et al., 2019b]. For example, in Fashion-MNIST, high-level features such as object category, may affect low-level features such as shape or height. Separately extracting such global and local information can be beneficial for visualization and data manipulation [Zhao et al., 2017]. To try and capture this, we introduce an inductive bias that is tailored to model and learn hierarchical features. We note here that our aim is not to try and provide a state-of-the-art hierarchical VAE approach, as a wide variety of highly–customized and powerful approaches are already well–established, but to show how easily the InteL-VAE framework can be used to induce hierarchical representations in a simple, lightweight, manner.

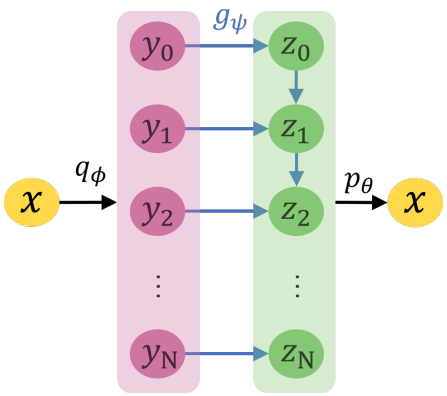


Figure 3.B.1: Graphical model for hierarchical InteL-VAE

Mapping design Following existing ideas from hierarchical VAEs [Sønderby et al., 2016, Zhao et al., 2017], we propose a hierarchical mapping g_ψ. As shown in Figure 3.B.1, the intermediary Gaussian variable y is first split into a set of N layers The mapping $z = g_\psi(y)$ is then recursively defined as $z_i = \text{NN}_i(z_{i-1}, y_i)$, where NN_i

is a neural network combining information from higher-level feature z_{i-1} and new information from y_i. As a result, we get a hierarchical encoding $z = [z_0, z_1, ..., z_N]$, where high-level features influence low-level ones but not vice-versa. This g_ψ thus endows InteL-VAEs with hierarchical representations.

Experiments While conventional hierarchical VAEs, e.g. [Sønderby et al., 2016, Zhao et al., 2017, Vahdat and Kautz, 2020], use hierarchies to try and improve generation quality, our usage is explicitly from the representation perspective, with our experiments set up accordingly. Fig. 3.B.2 shows some hierarchical features learned by InteL-VAE on **Fashion-MNIST**. We observe that high-level information such as categories have indeed been learned in the top-level features, while low-level features control more detailed aspects.

Figure 3.B.2: Manipulating representations of a hierarchical InteL-VAE. The features are split into 5 levels, with each of (a) [highest] to (e) [lowest] corresponding to an example feature from each. We see that high-level features control more complex properties, such as class label or topological structure, while low-level features control simpler details, (e.g. (d) controls collar shape).

To provide more quantitative investigation, we also consider the **CelebA** dataset [Liu et al., 2015] and investigate performance on downstream tasks, comparing to vanilla-VAEs with different latent dimensions. For this, we train a linear classifier to predict all 40 binary labels from the learned features for each method. In order to eliminate the effect of latent dimensions, we compare InteL-VAE (with fixed latent dimension 128) and vanilla VAE with different latent dimensions $(1, 2, 4, 8, 16, 32, 64, 128)$. We show experiment results on some labels as well as the average accuracy on all labels in Table 3.B.1 and Figure 3.B.3. We first find that the optimal latent dimension increases with the number of data points for the vanilla-VAEs, but is always worse than the InteL-VAE. Notably, the accuracy with InteL-VAE is quite robust, even as the number of data points gets dramatically low, indicating high data efficiency. To the best of our knowledge, this is the first result showing that a hierarchical inductive bias in VAE is beneficial to feature quality.

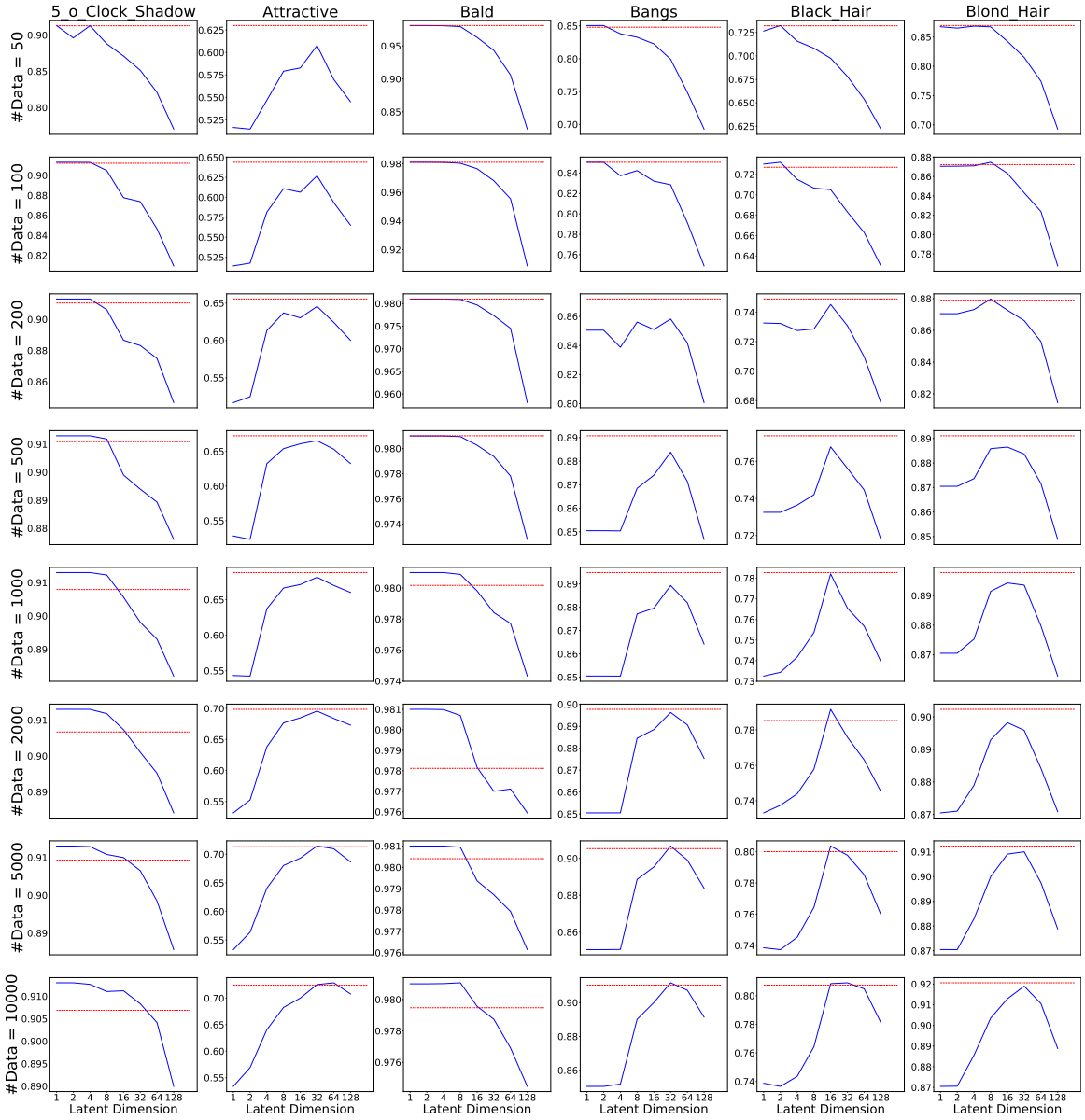


Figure 3.B.3: InteL-VAE's performance of attribute prediction on **CelebA** dataset. Each column shows results on the same feature with different data sizes and each column shows results on different features. In each graph, test accuracy of vanilla-VAE with different latent dimensions are shown in blue line. And results of InteL-VAE with hierarchical prior are shown in red. We find that our method (red line) achieves comparable or even better results compared with vanilla-VAE with all latent dimensions.

Related work Hierarchical VAEs [Vahdat and Kautz, 2020, Ranganath et al., 2016, Sønderby et al., 2016, Klushyn et al., Zhao et al., 2017] seek to improve the fit and generation quality of VAEs by recursively correcting the generative distributions. However, they require careful design of neural layers, and the hierarchical KL divergence

Model	Latent dim	Data size					
		50	100	500	1000	5000	10000
VAE	8	<u>0.791</u>	0.799	0.814	0.815	0.819	0.819
	16	0.788	<u>0.801</u>	0.820	0.824	0.829	0.831
	32	0.769	0.795	0.825	<u>0.832</u>	0.842	0.846
	64	0.767	0.794	<u>0.826</u>	<u>0.832</u>	<u>0.849</u>	<u>0.855</u>
	128	0.722	0.765	0.817	0.825	0.830	0.852
InteL-VAE	64	**0.817**	**0.824**	**0.841**	**0.846**	**0.854**	**0.857**

Table 3.B.1: Average accuracy in predicting all 40 binary labels of **CelebA**. Overall best accuracy is shown in bold and best results of vanilla-VAEs are underlined for comparison. Each experiment is repeated 10 times and differences are significant at the 5% level for data size ≤ 1000.

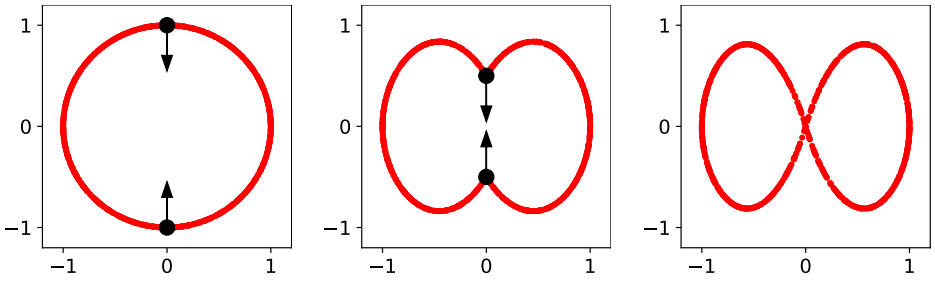
makes training deep hierarchical VAEs unstable [Vahdat and Kautz, 2020]. In comparison, InteL-VAE with hierarchical mappings is extremely easy to implement without causing any computational instabilities, while its aims also differ noticeably: our approach successfully learns hierarchical *representations*—something that is rarely mentioned in prior works.

3.C Full method and experiment details

In this section, we first provide complete details of the mapping designs used for our different InteL-VAE realizations along with some additional experiments. We then provide other general information about datasets, network structures, and experiment settings to facilitate results reproduction.

3.C.1 Multiple-connectivity

Mapping design Full details for this mapping were given in the main paper. Figure 3.C.1 provides a further illustration of the gluing process. Additional resulting including the Vamp-VAE are given in Figure 3.4.



(a) Circular prior with $h = 1$ (b) Glue point pair (c) Implied prior with $h = 2$

Figure 3.C.1: An illustration of the glue function in multiply-connected mappings.

3.C.2 Multi-modality

Mapping design In Section 3.6.2, we see the general idea of designing clustered mappings. In this part, we delve into the details of mapping design as well as extending it to 1 dimensional and high-dimensional cases. For simplicity's sake let us temporarily assume that the dimension of $\mathcal{Y}$ is 2. Our approach is based on splitting the original space into K equally sized sectors, where K is the number of clusters we wish to create, as shown in Figure 3.5b. For any point y, we can get its component (sector) index $\mathrm{ci}(y)$ as well as its distance from the sector boundary $\mathrm{dis}(y)$. By further defining the radius direction for the k-th sector (cf Figure 3.5c) as

$$\Delta(k) = \left(\cos\left(\frac{2\pi}{K}\left(k+\frac{1}{2}\right)\right), \sin\left(\frac{2\pi}{K}\left(k+\frac{1}{2}\right)\right)\right) \quad \forall k \in \{1,\ldots,K\},$$

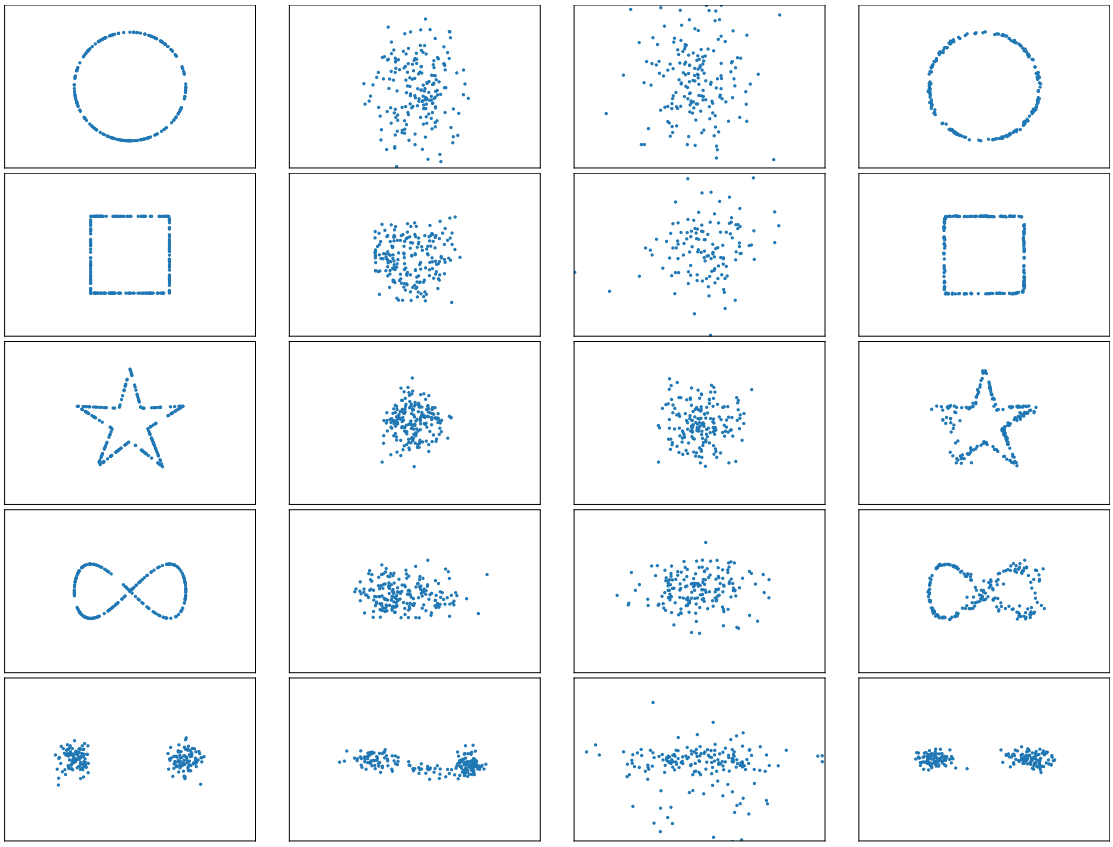
we can in turn define $g(y)$ as:

$$\mathrm{r}(y) = \Delta(\mathrm{ci}(y)), \tag{3.9}$$

$$g(y) = y + c_1 \mathrm{dis}(y)^{c_2} \mathrm{r}(y), \tag{3.10}$$

where c_1 and c_2 are constants, which are set to 5 and 0.2 in our experiments. we make sure g still continuous by keeping $g(y) = y$ on boundaries.

When dimension of $\mathcal{Y}$ is greater than 2, we have more diverse choice for g. When K is decomposable, i.e., $K = \prod_i K_i$, we can separately cut the plane expanded by $\mathcal{Y}_{2i}$ and $\mathcal{Y}_{2i+1}$ into K_i sectors by the Equation (3.9). As a result, $\mathcal{Y}$ is split into $K = \prod_i k_i$ clusters. When $K = 2$, we find that g only changes the 1-st dimension of $\mathcal{Y}$, so it can be applied to cases where latent dimension is 1.



(a) Real distribution (b) VAE (c) Vamp-VAE (d) InteL-VAE

Figure 3.C.2: Extension of Figure 3.4 showing Vamp-VAE baseline and additional circular target distribution (top row, uses the same single hole g_ψ as the second and third rows).

Learnable proportions We can also make the mapping more flexible by learning rather than assigning the cluster proportions. To do so, we keep a learnable value u_i for each cluster and set the angle of the i-th sector as $2\pi\text{Softmax}(u)_i$. Things are simpler for the 1-dimensional case where we can uniformly translate y by a learnable bias b before splitting the space from the origin.

3.C.3 Sparsity

Relationship to soft attention We note that our setup for the sparsity mapping shares some similarities with a soft attention layer [Bahdanau et al., 2014]. However, there are also some important points of difference. Firstly, soft attention aims to find the weights to blend features from different time steps (for sequential data) or different

positions (for image data). In contrast, the dimension selector (DS) selects which dimensions to activate or deactivate for the same latent vector. Secondly, the weights of features are usually calculated by inner products of features for soft attention, while DS relies on a network to directly output the logits.

Sparsity regularizer Our sparsity regularizer term, $\mathcal{L}_{sp}$, is used to encourage our dimensionality selector network (DS) to produce sparse mappings. It is defined using a mini-batch of samples $\{y_i\}_{i=1}^{M}$ drawn during training as per (3.7). During training, the first term of $\mathcal{L}_{sp}$ decreases the number of activated dimensions for each sample, while the second term prevents the samples from all using the same set of activated dimensions, which would cause the model to degenerate to a vanilla VAE with a lower latent dimensionality.

We note that $\mathcal{L}_{sp}$ alone is not expected to induce sparsity without also using the carefully constructed g_ψ of the suggested InteL-VAE. We confirm this empirically by performing an ablation study on **MNIST** where we apply this regularization directly to a vanilla VAE. We find that even when using very large values of $\gamma > 30.0$ we can only slightly increase the sparsity score ($0.230 \rightarrow 0.235$). Moreover, unlikely for the InteL-VAE, this substantially deteriorates generation quality, with the FID score raising to more than 80.0 at the same time.

Sparse metric We use the Hoyer extrinsic metric [Hurley and Rickard, 2009] to measure the sparsity of representations. For a representation $z \in \mathbb{R}^D$,

$$\text{Hoyer}(z) = \frac{\sqrt{D} - ||\hat{z}||_1/||\hat{z}||_2}{\sqrt{D} - 1}. \tag{3.11}$$

Here, following Mathieu et al. [2019b], we crucially first normalized each dimension d of z to have standard deviation 1, $\hat{z}_d = z_d/\sigma_d$, to ensure that we only measure sparsity that varies between data points (as is desired), rather than any tendency to uniformly 'switch off' certain latent dimensions (which is tangential to our aims). In other words, this normalization is necessary to avoid giving high scores to representations whose length scales vary between dimensions, but which are not really sparse.

By averaging Hoyer(z) over all representations, we can get the sparse score of a method. For the sparsest case, where each representation has a single activated dimension, the sparse score is 1. And when the representations get denser, $||\hat{z}||_2$ get smaller compared with $||\hat{z}||_1$, leading to smaller sparse scores.

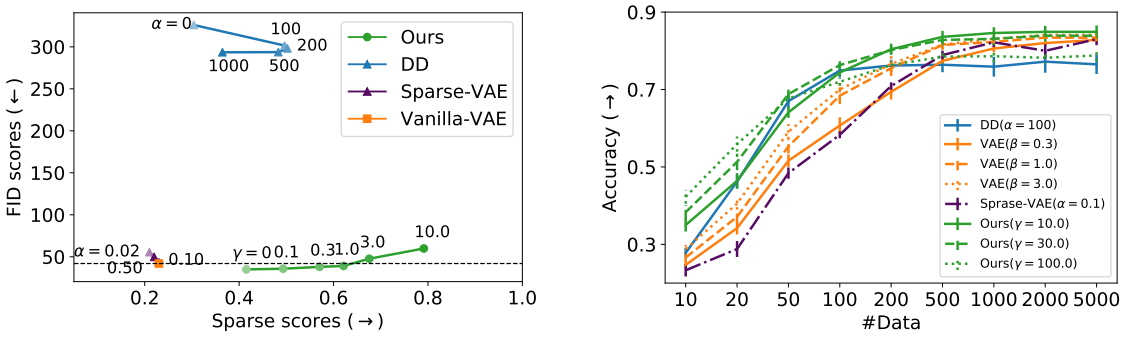


Figure 3.C.3: Results on **MNIST**. The *left* figure shows FID and sparsity scores. Lower FID scores (↓) represent better sample quality while higher sparse scores (→) indicate sparser features. The *right* figure shows the performance of sparse features from InteL-VAE on downstream classification tasks. See Section 3.6.3 for details and results for **MNIST**.

Reproduction of Sparse-VAE We tried two different code bases for Sparse-VAE [Tonolini et al., 2020]. The official code base[5] gives higher sparse scores for MNIST and FashionMNIST (though still lower than InteL-VAE), but is very unstable during training, with runs regularly failing after diverging and producing NaNs. This issue gets even more severe on CelebA which occurs after only a few training steps, undermining our ability to train anything meaningful at all. To account for this, we switched to the codebase[6] from De la Fuente and Aduviri [2019] that looked to replicate the results of the original paper. We report the results from this code base because it solves the instability issue and achieves reasonable results on CelebA. Interestingly, though its generation quality is good on MNIST and Fashion-MNIST, it fails to achieve a sparse score significantly higher than vanilla-VAE. As the original paper does not provide any quantitative evaluation of the achieved sparsity, it is difficult to know if this behavior is expected. We note though that the qualitative results shown in the paper appear to be substantially less sparse than those we show for the InteL-VAE, cf their Figure 5 compared to the top row of our Figure 3.8. In particular, their representation seems to mostly 'switch off' some latents entirely, rather than having diversity between datapoints that is needed to score well under the Hoyer metric.

[5] https://github.com/ftonolini45/Variational_Sparse_Coding
[6] https://github.com/Alfo5123/Variational-Sparse-Coding

Parameters	Synthetic	MNIST	Fashion-MNIST	MNIST-01	CelebA
Dataset sizes	Unlimited	55k/5k/10k	55k/5k/10k	10k/1k/2k	163k/20k/20k
Input space	$\mathbf{R}^2$	Binary 28x28	Binary 28x28	Binary 28x28	RGB 64x64x3
Encoder net	MLP	CNN	CNN	CNN	CNN
Decoder net	MLP	CNN	CNN	CNN	CNN
Latent dimension	2-10	50	50	1-10	1-128
Batch size	10-500	100	100	100	100
Optimizer	Adam	Adam	Adam	Adam	Adam
Learning rate	1e-3	1e-3	1e-3	1e-3	1e-3

Table 3.C.1: Hyperparameters used for different experiments.

Encoder
Input 64 x 64 x 3
4x4 conv. 64 stride 2 & BN & LReLU
4x4 conv. 128 stride 2 & BN & LReLU
4x4 conv. 256 stride 2 & BN & LReLU
Dense (*dim*)

Decoder
Input *dim*
Dense (8x8x256) & BN & ReLU
4x4 upconv. 256 stride 2 & BN & ReLU
4x4 upconv. 128 stride 2 & BN & ReLU
4x4 upconv. 3 stride 2

Table 3.C.2: Encoder and Decoder structures for CelebA, where *dim* is the latent dimension.

3.C.4 Additional experiment details

Datasets Both synthetic and real datasets are used in this paper. All synthetic datasets (sphere, square, star, and mixture of Gaussian) are generated by generators provided in our codes. For real datasets, We load MNIST, Fashion-MNIST, and CelebA directly from Tensorflow [Abadi et al., 2015], and we resize images from CelebA to 64x64 following Hou et al. [2017]. For experiments with a specified number of training samples, we randomly select a subset of the training data. We use the same random seed for each model in the same experiment and different random seeds when repeating experiments.

Model structure For low-dimensional data, the encoder and decoder are both simple multilayer perceptrons with 3 hidden layers (10-10-10) and ReLU [Glorot et al., 2011] activation. For MNIST and Fashion-MNIST, we use the same encoder and decoder as Mathieu et al. [2019b]. For CelebA, the structure of convolutional networks are shown in Table 3.C.2.

Experiment settings Other hyperparameters are shown in Table 3.C.1. All experiments are run on a GTX-1080-Ti GPU.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	On Incorporating Inductive Biases into VAEs.
Publication Status	☑Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Ning Miao, Emile Mathieu, N. Siddharth, Yee Whye Teh and Tom Rainforth. On Incorporating Inductive Biases into VAEs. International Conference on Learning Representations (ICLR), 2022

Student Confirmation

Student Name:	NING MIAO		
Contribution to the Paper	Ning is the lead author of this paper. He did all the experiments and wrote the manuscript.		
Signature	Ning Miao	Date	16/06/2024

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Tom Rainforth, Senior Researcher in Machine Learning		
Supervisor comments Ning led the paper, making a substantial contribution as per the description above.		
Signature	Date	20/06/2024

This completed form should be included in the thesis, at the end of the relevant chapter.

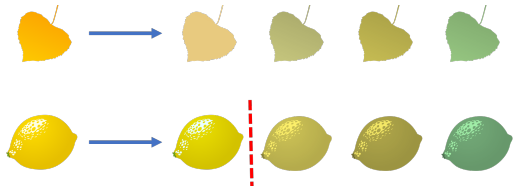
Chapter 4

Learning Instance–Specific Augmentations by Capturing Local Invariances

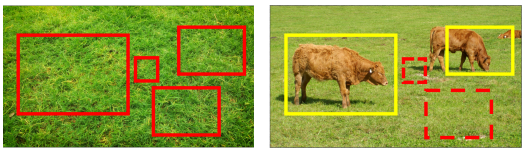
4.1 Introduction

Data augmentation is an important tool in deep learning [Shorten and Khoshgoftaar, 2019]. It allows one to incorporate inductive biases and invariances into models [Chen et al., 2019, Lyle et al., 2020], providing an effective regularization technique that aids generalization [Goodfellow et al., 2016]. It has proved particularly successful for computer vision tasks, forming an essential component of many modern supervised [Krizhevsky et al., 2012, Perez and Wang, 2017, Mikołajczyk and Grochowski, 2018, Cubuk et al., 2020] and self-supervised [Bachman et al., 2019, Chen et al., 2020, Tian et al., 2020, Foster et al., 2021] approaches.

Algorithmically, data augmentations apply a *random transformation* $\tau : \mathcal{X} \to \mathcal{X}$, $\tau \sim p(\tau)$, to each input data point $\mathbf{x} \in \mathcal{X}$, before feeding this *augmented* data into the downstream model. These transformations are resampled each time the data point is used (e.g. at each training epoch), effectively populating the training set with additional samples. Augmentation is also sometimes used at test time by ensembling predictions from multiple transformations of the input. A particular augmentation is defined by the choice of the *transformation distribution* $p(\tau)$, whose construction forms the key design choice. Good transformation distributions induce substantial and wide-ranging changes to the input, while preserving the information needed for prediction.



(a) Color jittering (red line indicates class boundary)



(b) Cropping

Figure 4.1.1: Different inputs require different augmentations. In (a), a leaf is invariant to color change from yellow to green, but the same transformation changes a lemon to a lime. In (b), the same effect is shown for cropping. Solid rectangles represent the patches that preserve the labels of the original images ([left] grass, [right] cattle), while dashed rectangles represent patches with different labels.

To try and ensure a good augmentation scheme, previous work has looked to *learn* this transformation distribution from data [Cubuk et al., 2018, Lim et al., 2019, Benton et al., 2020]. However, existing approaches typically assume independence between the input $\mathbf{x}$ and the transformation distribution $p(\tau)$. As such, they are only able to learn *global invariances*, severely limiting their flexibility and potential impact. For example, when using color jittering, changing the color of a leaf from yellow to green would preserve its label, but the same transformation would change a lemon to a lime (see Figure 4.1.1a). This transformation cannot, therefore, be usefully applied as a global augmentation, even though it is a useful invariance for the *specific input instance* of a leaf. Similar examples regularly occur for other transformations, such as cropping (see Figure 4.1.1b).

Another line of recent work [Zhou et al., 2020a, Cheung and Yeung, 2022] has instead looked to utilize instance-aware augmentations by defining a small predefined set of allowable transformations, then introducing a policy that assigns probabilities (and magnitudes) to elements of this set as a function of the input. While these approaches allow some of the shortfalls of global augmentations to be overcome, they do not have the flexibility to learn fine-grained transformation distributions, or uncover underlying invariances.

To address these shortfalls, we introduce InstaAug, a new approach to learn *instance-specific* augmentations by capturing *local* invariances that are specific to the region of the provided input. InstaAug is based on using a transformation distribution of the form $p(\tau; \phi(\mathbf{x}))$, where ϕ is a deep neural network that maps inputs to transformation distribution parameters. We refer to ϕ as an *invariance module*. It can be trained simultaneously with the downstream model in a fully end-to-end manner, or

individually with a fixed pre-trained model. Both cases only require access to training data and optimize a single objective function that minimizes the training error while maintaining transformation diversity. As such, *InstaAug* allows one to directly learn powerful and general augmentations, without requiring access to additional data or annotations.

We evaluate InstaAug in both supervised and self-supervised settings, focusing on image classification and contrastive learning respectively. Our experimental results show that InstaAug is able to uncover meaningful invariances that are consistent with human cognition, and improve model performance for various tasks compared with baseline models. While we primarily focus on the case where the invariance module is trained alongside the downstream model (to allow augmentation during training), we find that InstaAug can also provide substantial performance gains when used to learn test-time augmentations for large pre-trained models. Accompanying code is provided at https://github.com/NingMiao/InstaAug.

4.2 Background

Data augmentation methods operate as a wrapper algorithm around some downstream model, f, randomly transforming the inputs $\mathbf{x} \in \mathcal{X}$ before they are passed to the model. The outputs of the augmented model are given by $f(\tau(\mathbf{x}))$, where $\tau : \mathcal{X} \mapsto \mathcal{X}$ represents the transformation, sampled from some transformation distribution $p(\tau)$. The aim of this augmentation is to instill inductive biases into the learned model, leading to improved generalization by capturing invariances of the problem. It can be used both during training to provide additional synthetic training data, and/or at test-time, where ensembling the predictions from multiple transformations can provide a useful regularization that often improves performance [Shanmugam et al., 2021].

Some approaches look to learn aspects of the augmentation [Cubuk et al., 2018, 2020, Lim et al., 2019, Ho et al., 2019, Hataya et al., 2020, Li et al., 2020, Zheng et al., 2022]. These approaches can be viewed as learning parameters of $p(\tau)$, helping to automate its construction and tuning. Of particular relevance, Augerino [Benton et al., 2020] provides a mechanism for learning augmentations using a simple end-to-end training scheme, where the parameters of the downstream model and transformation

distribution are learned simultaneously using the (empirical) risk minimization

$$\min_{f,\theta} \mathbb{E}_{\mathbf{x},y\sim p_{\text{data}}} \left[\mathbb{E}_{\tau\sim p_\theta(\tau)} \left[\mathcal{L}(f(\tau(\mathbf{x})), y)\right]\right] + \lambda\mathbb{R}(\theta), \tag{4.1}$$

where $\mathcal{L}$ is a loss function and $\lambda\mathbb{R}(\theta)$ is a regularization term that encourages large transformations.

All of these approaches can be thought of as *global* augmentation schemes, in that transformations are sampled independently to the input. For an unrestricted, universal, class of transformations, this assumption can be justified through the noise outsourcing lemma [Kallenberg and Kallenberg, 1997]: any conditional distribution $Y|X = \mathbf{x}$ can be expressed as a deterministic function $g : \mathcal{X} \times \mathbb{R} \to \mathcal{Y}$ of the input and some independent noise $\varepsilon \sim \mathcal{N}(0, I)$. Thus, using reparameterization, the dependency on $\mathbf{x}$ can, in principle, be entirely dealt with by the transformation itself. However, in practice, the transformation class must be restricted to provide the desired inductive biases, meaning this result no longer holds and so the independence assumption can cause severe restrictions. For example, sampling color jitterings independently to the input is equivalent to the unrealistic assumption that the labels of all images $\mathbf{x}$ are invariant to the same group of changes (*cf.* Figure 4.1.1a).

4.3 InstaAug: capturing local invariances

In order to remedy the problems of global augmentations, we propose InstaAug. InstaAug learns an *input dependent* distribution $p(\tau; \phi(\mathbf{x}))$ of information-preserving transformations that actively makes use of the input $\mathbf{x}$ via the *invariance module* ϕ, as opposed to learning a global transformation distribution $p_\theta(\tau)$. This generalizes the hypothesis class of transformation distributions, and significantly increases the flexibility and expressivity of the resulting augmentation, without undermining our ability to carefully control the inductive biases that are imparted. It can also informally be viewed as a mechanism for learning invariances that are local to the specific input.

We argue that a good augmentation strategy needs to fulfill two properties. First, the transformations should preserve the information in $\mathbf{x}$ that is necessary for the task at hand. For example, transformations must preserve information about the label for supervised tasks. Second, the set of transformations needs to have sufficient 'diversity' to effectively augment the data; we quantify this as the entropy of the transformation

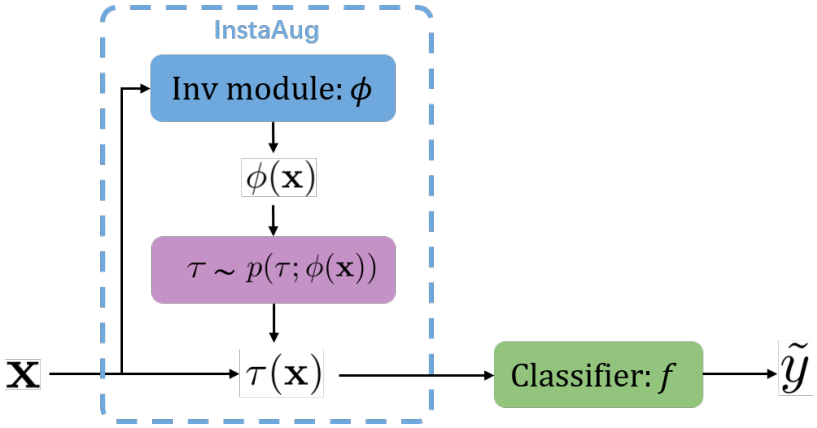


Figure 4.3.1: Summary of InstaAug.

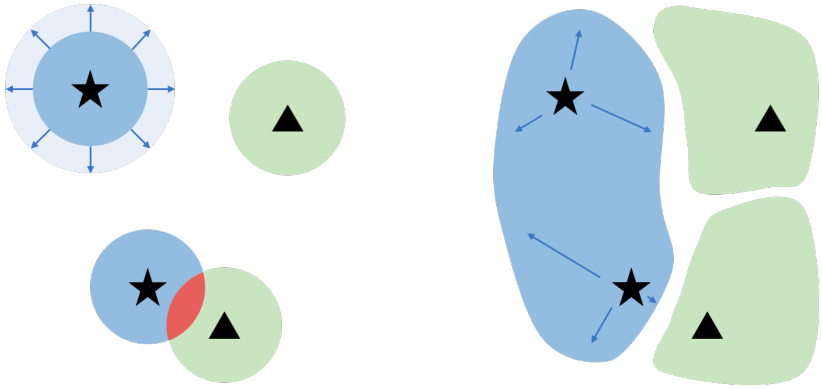
distribution $p(\tau; \phi(\mathbf{x}))$. In addition to their intuitive nature, in Section 4.A we provide theoretical analysis that shows these requirements naturally originate from a decomposition of the generalization error between the true risk and augmented empirical risk of f. For simplicity, we describe InstaAug for the specific case where f is a classifier in the remainder of this section.

4.3.1 Model structure

InstaAug is based around using a simple plug-in invariance module, ϕ, between the input $\mathbf{x}$ and the classifier f, as shown in Figure 5.2.1. We assume a parametric family of distributions $p(\tau; \cdot)$ over some transformation space, then use ϕ, which is a trainable neural network, to predict its parameters for a given input. During training, we *sample* a transformation $\tau \sim p(\tau; \phi(\mathbf{x}))$, which is applied to $\mathbf{x}$ to generate an augmented sample $\tau(\mathbf{x})$, before feeding this into the classifier f.

4.3.2 Training

Good augmentations should induce substantial changes to the input $\mathbf{x}$ while preserving all necessary information about the task at hand, thereby capturing the maximum possible invariance. Figure 4.3.2a illustrates the tension between these two objectives experienced by global augmentation schemes. Wider-ranging transformations are generally beneficial for generalization, but 'excessive' transformations will generate samples that will be incorrectly classified. In Figure 4.3.2a we see this in the red area,



(a) Global augmentation (b) InstaAug

Figure 4.3.2: InstaAug learns more diverse augmentations that also preserve labels compared to global augmentations. ★ and ▲ are samples from two different classes. Blue and green shades represent label-preserving augmentations for each class. In (a), the upper ★ would benefit from being further augmented, but some of the augmented samples for the lower ★ are already over-augmented and indistinguishable from another class (see the red intersection). InstaAug solves this problem by learning a different augmentation for each instance, as shown in (b).

where the augmentations for a pair of data points have started to overlap, creating ambiguity and inevitably misclassifications. Using instance-specific augmentations (Figure 4.3.2b) allows for a better trade-off of these needs. However, to achieve this we need our objective to encourage diversity in augmentations, not just low training error. It should also let the level of diversity vary between inputs, as some points will be able to support larger transformations than others.

Based on these needs, training is done by simultaneously minimizing a conventional expected loss with respect to both ϕ and f (or just ϕ if f is a fixed pre-trained classifier as per Section 4.5.3), while regularizing the average entropy of the transformations, $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\mathbb{H}[p(\tau;\phi(\mathbf{x}))]\right]$. The core motivation for this setup is that minimizing the expected loss will naturally encourage the information needed for prediction to be preserved, but the regularization on entropy is needed to enforce diversity. Further motivation is provided by the theoretical analysis of Section 4.A.

By appropriately parameterizing $p(\tau;\phi(\mathbf{x}))$ (see Section 4.3.3), we can write down its entropy in closed form. We can then formulate the problem as minimizing the following w.r.t. f and ϕ:

$$\mathbb{E}_{\mathbf{x},y\sim p_{\text{data}},\tau\sim p(\tau;\phi(\mathbf{x}))}\left[\mathcal{L}(f(\tau(\mathbf{x})),y)-\lambda\mathbb{H}[p(\tau;\phi(\mathbf{x}))]\right], \tag{4.2}$$

where $\mathcal{L}$ is the loss of the downstream task, for which we will generally use the cross-entropy. Unlike in the Augerino objective of Equation (4.1), λ here is an automatically-tuned weight of the entropy term that enables precise control over transformation diversity. Specifically, we initialize λ with a small positive value, then increase it when the average entropy drops below a lower bound $\mathrm{H}_{\min}$ and decrease it when it exceeds an upper bound $\mathrm{H}_{\max}$) during training. Here $\mathrm{H}_{\min}$ and $\mathrm{H}_{\max}$ are hyperparameters, through which we can directly control the diversity level of learned transformations during the whole training process. As described in Table 4.E.1, they can easily be tuned.

This dynamic λ is necessary because the requirement for λ is different at different stages of training. In the beginning, when the classifier is weak, we need a small λ to avoid the transformations becoming overly diverse, which results in different classes overlapping with each other. As the classifier gets more powerful during training, larger λ is needed to compete with the cross-entropy term $\mathcal{L}$.

Using this approach, the invariance module and downstream model can be trained simultaneously using end-to-end gradient descent, utilizing the reparameterization trick to deal with the stochasticity of τ when possible [Kingma and Welling, 2014], and the REINFORCE estimator [Williams, 1992] otherwise. The approach can also be extended to regression or self-supervised learning by substituting the loss function $\mathcal{L}$ (see Section 4.C).

4.3.3 Parameterization of augmentations

The parameterization method is another critical factor in the quality of learned invariances. A good parameterization should be flexible enough to reflect the complexity of real data while not creating obstacles to gradient-based learning. Here we focus on parameterizing transformations that are frequently used in computer vision, though our framework can easily be extended to other domains. Due to the varied characteristics of different image transformations, we design two different parameterization methods for $p(\tau; \phi(\mathbf{x}))$.

Uniform parameterization. For simpler transformations, such as rotation and color jittering, we find that a uniform distribution is enough for parameterizing $p(\tau; \phi(\mathbf{x}))$, such that $\phi(\mathbf{x})$ returns a pair $(\theta_{\min}, \theta_{\max})$ representing extrema of the

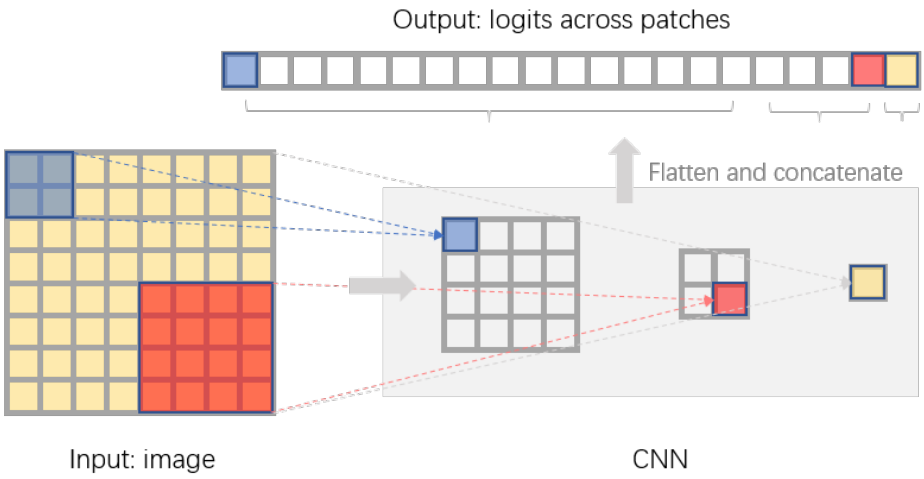


Figure 4.3.3: Location-related parameterization of crops by a CNN. The shaded area (bottom right) shows a simplified 3-layer CNN, and squares represent units at different convolutional layers. Each unit defines a patch in the input image (shown in the same color) through its receptive field. The activation value of the unit then gives the corresponding unnormalized log probability for that patch.

possible transformations. For example, for rotations, these represent the maximum and minimum rotation angles, such that $\tau(\mathbf{x}) = R(\theta)\mathbf{x}$, where $\theta \sim \mathcal{U}(\theta_{\min}, \theta_{\max})$ and $R(\theta)$ is the rotation operator. To compose multiple transformations, we simply sample them independently, such that $p(\tau_1, \ldots, \tau_K; \phi(\mathbf{x})) = \prod_{k=1}^{K} p(\tau_k; \phi_k(\mathbf{x}))$. This provides a similar parameterization to [Benton et al., 2020], but where $(\theta_{\min}, \theta_{\max})$ now critically varies with the input $\mathbf{x}$ and there is no symmetry assumption on this range.

Location-related parameterization Using this uniform parameterization is unfortunately not appropriate for more complex transformations like cropping. Firstly, the distribution of crop centers may be multi-modal, since important information may exist in different parts of an image. Secondly, the desired crop size and center are often highly correlated so cannot be sampled independently. Finally, we encountered significant practical training issues when using the uniform parameterization for cropping, with ϕ becoming trapped in local optima with little transformation diversity.

We, therefore, propose an alternative location-related parameterization (LRP) for cropping, which is based on defining a large set of representative crops, then constructing ϕ to map from inputs to a vector of probabilities over this set. As shown in Figure 4.3.3, this is achieved using a CNN where each hidden unit corresponds to a possible crop defined by its receptive field. In order to select crops with different sizes, units from different layers are utilized, with those of earlier/latter layers representing

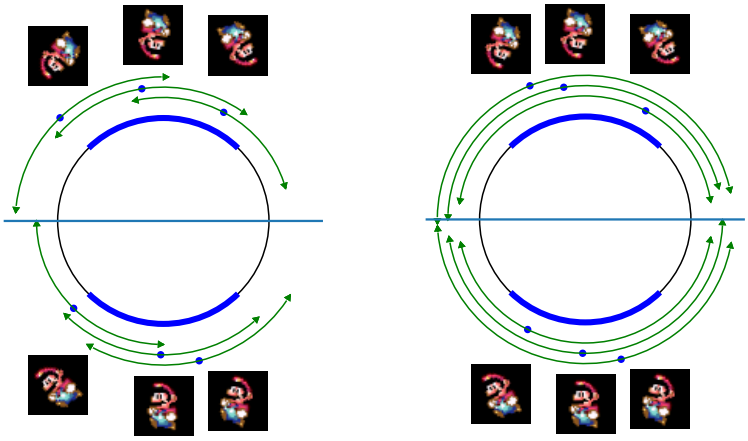
smaller/larger crops. This parameterization proved more effective than simply outputting the probabilities from a conventional network, due to the greater parameter sharing between related crops. We note that it can also be directly extended to other transformations, such as masking, local blurring, pixel-wise perturbation, and local color jittering.

4.4 Related work

Hard-coded invariance. Many recent works have looked to hard-code global invariance in neural networks. For example, various architectures have been designed to be invariant to translation [Chaman and Dokmanic, 2021, Zhang, 2019], rotation [Worrall et al., 2017, Zhou et al., 2017, Marcos et al., 2017], scaling [Worrall and Welling, 2019, Sosnovik et al., 2019] or other group actions [Cohen and Welling, 2016, Xu et al., 2021]. Unfortunately, they require the set of invariant transformations to be closed under composition, leaving out many practical transformations that do not form a group.

Learning augmentations. There have been numerous prior works that automatically learn *global* augmentations and invariance from data. As discussed in Section 4.2, Augerino [Benton et al., 2020] is perhaps the most closely linked such approach to InstaAug as it also relies on end-to-end training (see Section 4.B for further discussion on its similarities and differences to InstaAug). AutoAugment [Cubuk et al., 2018] instead uses reinforcement learning to find augmentation strategies that increase accuracy on a separate validation set. Various follow-up works have improved its efficiency and/or performance [Lim et al., 2019, Ho et al., 2019, Tang et al., 2019, Hataya et al., 2020, Li et al., 2020, Cubuk et al., 2020, Zheng et al., 2022].

Augmentation policies. A couple of recent works have further looked to learn augmentation *policies* that allow a degree of dependency on the input or class label, namely AdaAug [Cheung and Yeung, 2022] and MetaAugment [Zhou et al., 2021]. These policies assign probabilities and magnitudes to a fixed finite list of possible transformation operations. Though they can depend on the input, they only make discrete choices and cannot learn a fine-grained transformation distribution in the way that InstaAug does; they are thus not suitable for capturing local invariances. For example, using InstaAug with cropping learns a joint distribution over patch positions and sizes, whereas these methods only learn a probability for whether to apply cropping



(a) Augerino (b) InstaAug (Ours)

Figure 4.4.1: Learned invariances for the Mario and Iggy dataset. The blue arcs show the training data range, while the green arcs show example learned transformation distributions.

or not, and a scalar magnitude to use if it is applied. Further, both AdaAug and MetaAugment require a separate validation set, only consider augmentation during training (so cannot be used for pre-trained classifiers), and cannot be applied in unsupervised settings. AdaAug also has the additional restriction that its policy is based on a linear mapping from the penultimate layer of the classification model, so its input dependence is inherently limited. Meanwhile, although MetaAugment learns a sample-level policy network, in practice it averages this policy among training samples to form a global policy applied to all samples.

Other related work. The spatial transformer [Jaderberg et al., 2015] aims to learn instance-specific transformations, but only applies a single transformation to each input rather than a distribution of transformations, making it distinct from data augmentation. Luo et al. [2020] and Kim et al. [2020a] both also learn instance-specific augmentations. However, the latter consider only test-time augmentation, while the former introduces an approach that is highly specialized to test recognition and cannot be applied in the more general settings we consider. Schwöbel et al. [2022] adopt the Bayesian paradigm to learn invariances by maximizing an approximation of the marginal likelihood. Limited by the ability of approximation methods, it only works on simple datasets such as MNIST. Tian et al. [2021] use a continuous parameterization for all common augmentations and optimize it by stochastic gradient Langevin dynamics [Welling and Teh, 2011]. However, they require training on a separate validation set. Tamkin et al. [2020] and Chen et al. [2021] both utilize

adversarial augmentations to increase robustness. Zhou et al. [2020a] learn symmetries shared across several datasets through a meta-learning scheme. Mixup methods [Zhang et al., 2018, Yun et al., 2019, Ramé et al., 2021] can also be thought of as a specific type of data augmentation. Some of them [Kim et al., 2020b,c, Park et al., 2022], allow for input dependence through gradient-based saliency [Simonyan et al., 2013]. However, they use a fixed augmentation strategy rather than learning a transformation distribution.

4.5 Supervised learning experiments

In this section, we show the effectiveness of InstaAug in learning rotation, cropping, and color jittering. We use location-related parameterization (LRP) for cropping and uniform parameterization for rotation and color jittering.

4.5.1 Rotated 2D images

We first consider a simple synthetic dataset proposed in Benton et al. [2020]. The dataset contains four categories, (1) upright Mario; (2) upside-down Mario; (3) upright Iggy; and (4) upside-down Iggy. Each of the four base images is randomly rotated in the interval of $[-\pi/4, \pi/4]$ to form the training dataset. The task is to predict the correct character (Mario vs Iggy) and the orientation (up vs down). We assess whether InstaAug is able to learn the 'best' rotation range for each sample—i.e. the maximum range that avoids 'up' and 'down' classes from overlapping.

Figure 4.4.1 shows that InstaAug effectively recovers the broadest range of rotations for each image while preserving labels, while Augerino only learns a subset of these ranges. This can be most easily seen by the fact that the transformation distributions (shown in green) always extend to very close to the true class boundary for InstaAug, but not for Augerino. These gains are because Augerino learns a *single* global augmentation distribution shared across all images (note the shared transformation distribution arcs), which are inevitably limited for any given input.

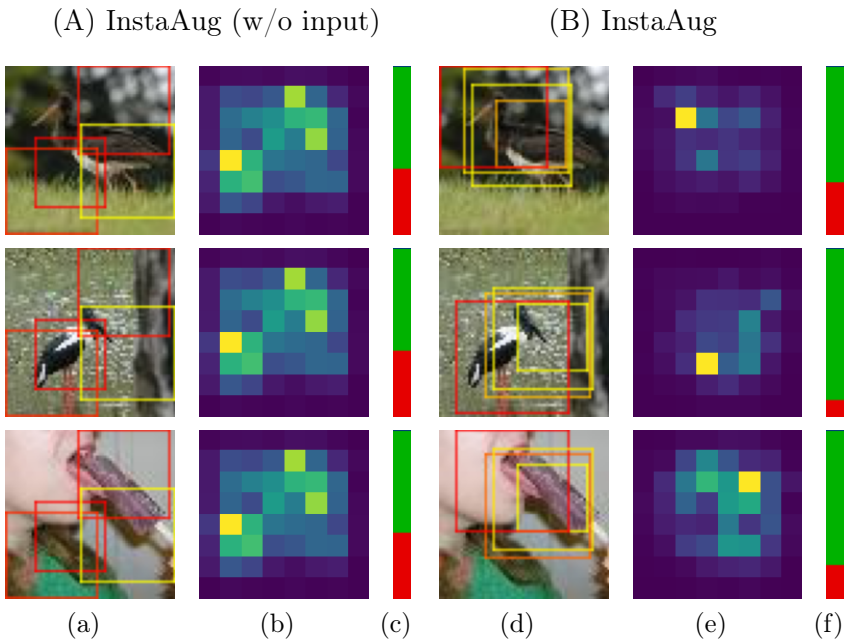


Figure 4.5.1: InstaAug (B) learns more sensible crops compared to random and learned global (A) augmentations. Columns (a, d) show examples of sampled crops, with red edges indicating higher probability. Columns (b, e) show density maps for the crop centers, with brighter color meaning higher probability. Columns (c, f) give the proportion of crops (red) above a particular size threshold, showing that InstaAug produces fewer large crops.

4.5.2 Cropping

We now move to more realistic images and to the most common and effective form of image augmentation: cropping. We first evaluate the performance of jointly training InstaAug and the classifier on Tiny-Imagenet (TinyIN, 64×64), as it inherits the image complexity of ImageNet whilst being within our computational budget. TinyIN is a standard testbed for data augmentations. Full experiment details are given in Section 4.D.1.

We benchmark InstaAug alongside several augmentation baselines, including random crop, Augerino, and AdaAug. For random crop, we use a uniform distribution on patch size and position, tuning the bounds on the former through a comprehensive hyperparameter sweep to ensure appropriate scaling. We further compare to other prior works that have obtained competitive results on TinyIN [Zhang et al., 2018, Yun et al., 2019, Ramé et al., 2021, Kim et al., 2020b,c, Park et al., 2022], directly taking their reported results.

Method	Instance	Accuracy (%)
No augmentation	✗	$55.06_{\pm 0.10}$
Random crop	✗	$64.49_{\pm 0.12}$
MixUp	✗	63.74
CutMix	✗	65.09
MixMo	✗	64.80
Puzzle-Mix	✓	63.48
Co-Mixup	✓	64.15
Saliency grafting	✓	64.84
Augerino	✗	$55.02_{\pm 0.29}$
AdaAug	✓	$64.03_{\pm 0.19}$
-w/ LRP	✓	$62.01_{\pm 0.23}$
InstaAug (ours)	✓	$\mathbf{66.02}_{\pm 0.18}$
-w/o LRP	✓	$55.39_{\pm 0.19}$
-w/o input	✗	$63.20_{\pm 0.12}$
-class-specific	✗/✓	$60.55_{\pm 0.50}$

Table 4.5.1: Test accuracy for Tiny-ImageNet with cropping augmentation. The Instance column indicates whether the method is instance-specific or not. Statistics are computed over 10 runs, except for MixUp methods, whose results are directly taken from their respective papers. We omit comparison to other global augmentation schemes, as these only learn the size ranges of the cropping, which is already covered by the hyperparameter tuning of Random crop.

In order to ablate the effects of input-dependency and our location-related parameterization (LRP, see Figure 4.3.3) on InstaAug, we additionally assess the performance of *InstaAug (without LRP)* by using same uniform parameterization as Augerino; *InstaAug (without input)* that uses the LRP and general InstaAug setup, but shares the transformation distribution across all inputs rather than learning an input-specific augmentation; and *InstaAug (class-specific)*, which takes training labels instead of images as inputs. Test-time augmentation using 50 transformation samples is deployed for all variants of InstaAug, along with the Augerino, AdaAug, and random augmentation baselines (see Section 4.C.2). For InstaAug (class-specific), this test-time augmentation is based on random cropping, as class information is not available at test-time and simply omitting test-time augmentation performed poorly. Following prior works, we choose the PreActResNet-18 architecture [He et al., 2016b] with width = 1 as the classifier for all methods.

Table 4.5.1 shows the (top-1) test accuracy for each method. In agreement with prior works, we find that random cropping increases accuracy by 9.4% over no augmentation,

which is achieved where cropping scale $= [0.1, 1]$. InstaAug outperforms random cropping and its own global version without input by 1.5% and 2.8% respectively, highlighting the effect of learning instance-specific augmentation.

Allowing only for class dependence actually produces even worse performance than just ignoring the input completely, presumably because of the inevitable resulting mismatch in the augmentations used in training and testing. Methods with mean-field uniform parameterization (including Augerino and InstaAug without LRP) performed extremely poorly, noticeably worse than just random cropping. This is because they were found to become easily stuck at local minima with low cropping diversity, leading to similar performance as *no augmentation*. The original version of AdaAug achieves similar performance to Random crop, but is incapable of dealing with the large search space of LRP, leading to a small reduction in performance when this is added. Note that the potentially unexpectedly good performance of the random cropping baseline compared to the other global baselines stems from the careful hyperparameter sweep used to tune its crop size, which proved more effective than these more direct training mechanisms. See Section 4.E.3 for more discussion.

Figure 4.5.1 shows example crops and learned transformation distributions for InstaAug and a global augmentation scheme (InstaAug without input). We see that InstaAug is able to learn a cropping scheme that focuses on the key aspect of the input image, while the baselines cannot.

4.5.3 Applying InstaAug to a fixed classifier

InstaAug can also be used to learn suitable augmentations for a fixed pre-trained classifier. This can most notably be useful as a means to learn test-time augmentations. As the invariance module is itself only a small network, it can be done relatively cheaply, even when the dataset and downstream model are very large. We exploit this on the larger Imagenet dataset (224×224) [Deng et al., 2009], again focusing on cropping augmentations and utilizing the LRP parameterization from Section 4.3.3.

Training the invariance module in this setting is done in exactly the same way as elsewhere, using the training procedure of Section 4.3.2 with the normal training data. The only thing that is changed is that f is now fixed to a pre-trained classifier—specifically, the ResNet-50 [He et al., 2016a] from Wightman [2019] (which did not

Method	#Sample	ResNet50	ResNet18	XCiT
No aug	1	80.43	69.73	86.34
Random crop	4	$78.45_{\pm 0.04}$	$66.13_{\pm 0.04}$	$82.05_{\pm 0.01}$
AutoAug	4	$77.84_{\pm 0.05}$	$59.50_{\pm 0.01}$	$81.40_{\pm 0.00}$
FastAutoAug	4	$77.87_{\pm 0.06}$	$61.43_{\pm 0.02}$	$81.42_{\pm 0.01}$
InstaAug	4	$\mathbf{80.92}_{\pm 0.04}$	$\mathbf{70.59}_{\pm 0.05}$	$\mathbf{86.43}_{\pm 0.04}$
Random crop	10	$79.60_{\pm 0.01}$	$67.87_{\pm 0.01}$	$82.84_{\pm 0.00}$
AutoAug	10	$79.20_{\pm 0.04}$	$63.96_{\pm 0.03}$	$82.43_{\pm 0.02}$
FastAutoAug	10	$79.28_{\pm 0.01}$	$65.65_{\pm 0.02}$	$82.45_{\pm 0.02}$
InstaAug	10	$\mathbf{81.18}_{\pm 0.02}$	$\mathbf{70.96}_{\pm 0.03}$	$\mathbf{86.47}_{\pm 0.02}$

Table 4.5.2: InstaAug boosts the test accuracy (%) with test-time augmentation on Imagenet. Invariance modules learned on ResNet-50 can also be directly applied to other models such as ResNet-18 and XCiT to improve generalization without fine-tuning. By contrast, global augmentation schemes are actually detrimental to test-time augmentation.

use an invariance module during training)—rather than being simultaneously learned. We are thus simply learning invariances, without affecting the training of f.

In Table 4.5.2, we show the effect of using the learned invariance module for test-time augmentation, finding that it is able to noticeably improve accuracy, unlike the baseline test-time augmentations of random cropping, AutoAugment [Cubuk et al., 2018], and Fast AutoAugment [Lim et al., 2019]. Note that AdaAug cannot be used in this fixed-classifier setting.

In order to evaluate the generalization performance of our learned augmentation module, we further apply the augmentation trained on ResNet-50 to two different models *with zero fine-tuning*: ResNet-18 [He et al., 2016a] and XCiT [Ali et al., 2021]. We find that the learned augmentation transfers very effectively to these different models, which implies that the local invariances InstaAug learns to reflect the natural invariances of the underlying classification problem, rather than being specific to the model that was used to train the augmentation module.

4.5.4 Color jittering on textures

Color jittering is another important type of data augmentation, which can help models generalize to different lighting conditions. We benchmark on the texture classification

Method	Test aug?	Accuracy (%)
No aug	✗	$72.87_{\pm 0.10}$
Random aug	✗	$79.99_{\pm 0.13}$
Augerino	✗	$78.97_{\pm 0.10}$
AdaAug	✗	$75.27_{\pm 0.30}$
InstaAug	✗	$\mathbf{81.11}_{\pm 0.20}$
Random aug	✓	$80.55_{\pm 0.16}$
Augerino	✓	$79.34_{\pm 0.14}$
AdaAug	✓	$76.43_{\pm 0.15}$
InstaAug	✓	$\mathbf{81.35}_{\pm 0.19}$

Table 4.5.3: InstaAug achieves higher general accuracy than baseline methods when trained on D45 (Daylight, 4500K).

#Lighting conditions	1	2	4	8
No aug	$68.5_{\pm 2.6}$	$78.1_{\pm 1.8}$	$84.8_{\pm 0.7}$	$87.8_{\pm 0.5}$
Random augmentation	$72.7_{\pm 2.7}$	$80.8_{\pm 1.3}$	$85.9_{\pm 0.6}$	$87.3_{\pm 0.3}$
InstaAug	$\mathbf{76.0}_{\pm 2.5}$	$\mathbf{83.6}_{\pm 1.1}$	$\mathbf{88.2}_{\pm 0.5}$	$\mathbf{89.6}_{\pm 0.3}$

Table 4.5.4: InstaAug significantly outperforms baseline methods in general test accuracy (%) on different difficulty levels. Difficulty level is controlled by the number of randomly sampled lighting conditions seen. Test-time augmentation is included for random and InstaAug and we repeat each experiment for 10 times.

dataset RawFooT [Bianco et al., 2017]. RawFooT includes 68 different samples of raw food and each sample has an image taken under each of 46 lighting conditions (see Figure 4.C.1 for examples), which makes it an ideal testbed to investigate methods' generalization ability between different lighting conditions. We crop the original images to create the train set and test set. For each original image with a resolution of 800×800, we randomly sample 200 different 200×200 patches in the upper half as training images. The same procedure is taken on the lower half to produce test images, giving a train set and a test set for each different lighting condition. To evaluate the generalization ability to a broader range of lighting conditions, we evenly mix test images from all lighting conditions to form a general test set, while controlling the conditions during training.

We first train on a single lighting condition D45 (4500K, daylight) resembling natural light. Table 4.5.3 shows that InstaAug outperforms all baselines with and without test-time augmentation. We find that Augerino (with relaxed symmetry restrictions

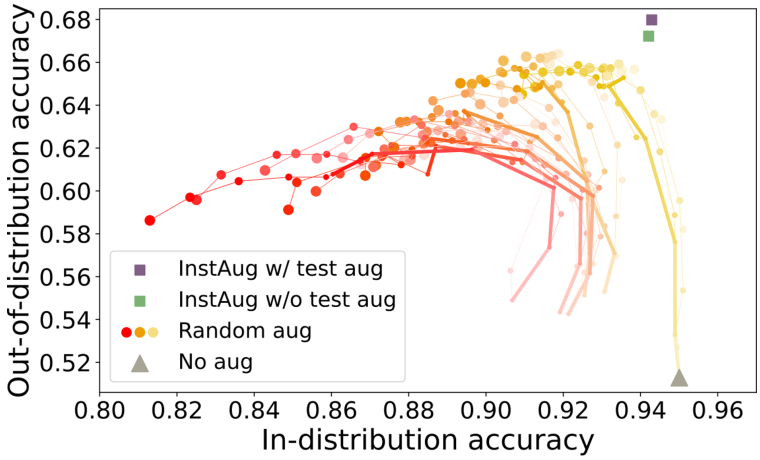


Figure 4.5.2: In- and out-of-distribution test accuracy for models trained on RawFooT D45. The round dots are random augmentation with different hyperparameter settings. The colors of dots change from yellow to red as hue jittering increases; more saturated dots indicate higher saturation jittering; larger dots mean higher brightness jittering. Thick lines connect dots with the same hue and brightness jitter, thin lines link dots with the same hue and saturation jitter.

on learned intervals) underperforms random augmentation because its parameters ϕ are often stuck near their initial values. We believe this is due to the conservative nature of using global augmentations (*cf.* Figure 4.3.2), where even a small change in the parameters may largely increase the training loss, which prohibits wide-ranging augmentations. AdaAug does not perform well either, which might be a result of its inability to learn the interval length for the distribution of each transformation.

We also compare in-distribution and out-of-distribution generalization by splitting the 46 test sets into two groups, according to the similarity of their lighting conditions to D45—see Section 4.D.2 for the details on the splitting method. In Figure 4.5.2 we can see that above a certain in-distribution performance, there exists a trade-off for random augmentation between in-distribution accuracy and out-of-distribution generalization, controlled through the hyperparameter settings. InstaAug, meanwhile, delivers higher out-of-distribution performance than any of the hyperparameter configurations, while also simultaneously giving better in-distribution accuracy to the vast majority of them as well.

We can further vary the difficulty of the classification task by using different numbers of lighting conditions in the training data. In Table 4.5.4, we randomly select a set number of lighting conditions to use as the training set for each baseline. As expected, the accuracy increases with the number of lighting conditions for all methods. However,

the effect of random augmentation saturates: it performs similarly to no augmentation with 8 lighting conditions. By contrast, InstaAug always provides improvements. In Section 4.D, we show that these gains come at very little computational overhead at both train and test time.

4.6 InstaAug for contrastive learning

Contrastive learning aims to learn features that are approximately invariant to certain augmentations. Typical contrastive learning methods, such as SimCLR [Chen et al., 2020, Ermolov et al., 2021], first sample two independent transformations, $\tau_1, \tau_2 \sim p(\tau)$, and apply them to an input image $\mathbf{x}$, generating two views $\mathbf{x}_1$ and $\mathbf{x}_2$. They then feed the transformed images to a neural encoder f, which is trained to maximize the similarity between $f(\mathbf{x}_1)$ and $f(\mathbf{x}_2)$, measured with a contrastive loss.

The choice of augmentations directly influences the learned invariance of the encoder and thus forms a crucial ingredient of contrastive learning [Bachman et al., 2019, Chen et al., 2020, Tian et al., 2020]. Existing schemes use global augmentations that often introduce unrealistic assumptions. For example, if there are multiple entities in an image, such as grass and cattle in Figure 4.1.1b, random cropping will pull features for different entities closer to each other. Consequently, we propose InstaAug as a more flexible instance-specific augmentation method for contrastive learning.

Applying InstaAug to contrastive learning is similar to the supervised case shown in Section 5.2. The main difference is, given an input $\mathbf{x}$, we sample two τ independently from the input-specific distribution $p(\tau; \phi(\mathbf{x}))$, before they are applied to $\mathbf{x}$. The training objective is correspondingly changed to minimizing the contrastive loss while keeping the diversity in a reasonable range.

We again consider TinyIN and evaluate three methods: InstaAug, InstaAug (without input), and Random crop. We exclude methods with uniform parameterization because of their earlier poor performance and note that AdaAug is not applicable for unsupervised learning. All experiments are based on the SimCLR framework and use the PreActResNet-18 network as the encoder. We train each model with a batch size of 512 for 500 epochs. We then train a linear classifier to evaluate feature quality. We use test-time augmentation—with 10 sampled crops—as this has been shown to improve performance [Foster et al., 2021].

Method	Accuracy (%)
Un-Mix [Shen et al., 2022]	49.58*
Random crop	$51.63_{\pm 0.30}$
InstaAug (without input)	$54.20_{\pm 0.23}$
InstaAug	$\mathbf{55.05}_{\pm 0.21}$

Table 4.6.1: Representations learned by InstaAug perform better in the downstream linear classification task than baselines. *Results of Un-Mix are directly taken from Shen et al. [2022], which has the same network structure (ResNet-18), training algorithms (SimCLR) and linear classifier as ours.

(a) (b) (c)

Figure 4.6.1: Examples of learned croppings by InstaAug for contrastive learning. Conventions as per Figure 4.5.1.

Table 4.6.1 shows that InstaAug outperforms the random and global augmentation schemes as well as Un-Mix [Shen et al., 2022], which is a recent variant of MixUp methods for contrastive learning. We see from the examples in Figure 4.6.1 that InstaAug focuses on the salient features containing important information. We also notice that the sizes of learned patches are correlated to the sizes of the objects in the images. Thus, InstaAug is able to learn sensible instance-specific augmentations in a fully unsupervised setting.

4.7 Conclusions

In this paper, we introduced InstaAug, a method for learning instance-specific data augmentations that capture local invariances of the underlying data-generating process. This is achieved by training an augmentation module that parametrizes an input-dependent distribution over transformations, whose samples can be used to augment the training data on the fly and/or for test-time augmentation. The main benefits

of InstaAug stem from its applicability to a wide range of settings, its ease of use, and crucially its capacity to learn meaningful augmentations that in turn improve performance. Empirically, we have demonstrated these benefits for both classification and contrastive learning problems, considering several classes of transformations.

Appendix

4.A Theoretical analysis of generalization error

We now provide a decomposition of the generalization error—i.e. the difference between the true risk and the training risk—when using ϕ during training of the downstream classifier f. Here we can view the objective of augmentation as adjusting the training objective to encourage the learned model to have a low true risk. As such, the generalization error provides a measure of the effectiveness of the augmentation for the training of f; by analyzing the behavior of the generalization error as a function of the augmentation module, we can derive a characterization of the desirable properties of the latter.

To start our analysis, we first define the true risk of the downstream model, f, as

$$R(f) := \mathbb{E}[\mathcal{L}(f(X), Y)] \tag{4.3}$$

where $(X, Y) \sim p_{\text{true}}(X, Y)$ are drawn from the true data generating distribution. In practice, one might also perform test-time augmentation, implying a different predictive function and thus different true risk, but for the purposes of our analysis, we will assume that this is not done, as this allows us to focus on the impact the invariance module has on f during training.

On the other hand, the implied training risk (i.e. our objective for training f) when using an invariance module is the augmented empirical risk

$$\hat{R}(f, \phi) := \mathbb{E}[\mathcal{L}(f(\tau(x_i)), y_i)] \tag{4.4}$$

where $i \sim \text{Uniform}\{1, \ldots, N\}$ is a uniformly sampled index for a point in the original training dataset $\{x_n, y_n\}_{n=1}^N$ and $\tau | i \sim p(\tau; \phi(x_i))$ is the sampled transformation. Note that the expectation in Equation (4.4) is only over i and τ, with the data-points

themselves not considered random variables for our purposes, because we are only provided with a single fixed training dataset.

The generalization error can now be defined as $\hat{R}(f, \phi) - R(f)$. At a high level, we are interested in finding a ϕ that ensures this has a low magnitude. More precisely, we want ϕ to ensure that the minimizer of the training risk, $\hat{f}^* := \arg\min_f \hat{R}(f, \phi)$, gives as low a true risk, $R(\hat{f}^*)$, as possible. Therefore, we want to keep the generalization error magnitude small across different f (relative to the corresponding variations in $\hat{R}(f, \phi)$ itself), so that the optima of the training and true risks are as similar as possible. In other words, we want a ϕ that ensures $\hat{R}(f, \phi) - R(f)$ is small (in magnitude) for *all* f, especially those close to $\hat{f}^*$. If we do hypothetically drive the generalization error to zero for all f, we will have a mechanism for directly training to the true risk using a finite original training dataset.

To aid with decomposing the generalization error, it is convenient to further define the following random variables through their conditional distributions:

$$\hat{Y}|i \sim p_{\text{true}}(Y = \hat{Y}|X = x_i) \quad \text{with} \quad \hat{Y} \perp\!\!\!\perp \tau, \tag{4.5}$$

$$\tilde{Y}|i, \tau \sim p_{\text{true}}(Y = \tilde{Y}|X = \tau(x_i)). \tag{4.6}$$

We can now write down our decomposition as follows:

$$\begin{aligned} \hat{R}(f, \phi) - R(f) = &\underbrace{\mathbb{E}[\mathcal{L}(f(\tau(x_i)), \hat{Y}) - \mathcal{L}(f(\tau(x_i)), \tilde{Y})]}_{\text{(A)}} \\ &+ \underbrace{\mathbb{E}[\mathcal{L}(f(\tau(x_i)), \tilde{Y}) - \mathcal{L}(f(X), Y)]}_{\text{(B)}} \\ &+ \underbrace{\mathbb{E}[\mathcal{L}(f(\tau(x_i)), y_i) - \mathcal{L}(f(\tau(x_i)), \hat{Y})]}_{\text{(C)}}. \end{aligned} \tag{4.7}$$

From this, we see that if the magnitude of (A), (B), and (C) are all small, then our generalization error magnitude will be small as well. Moreover, if we can construct a ϕ such that these terms are small for *all* f, then we can ensure effective generalization performance. We will now look at each term individually.

(A) provides a precise characterization of how well our transformation preserves the label distribution; it is the difference between the expected loss under the true label distribution of the untransformed inputs and the expected loss under the true label distribution of the transformed inputs, making predictions using the transformed

inputs in both cases. In particular, by noting that we have

$$\text{(A)} = \mathbb{E}\left[\mathbb{E}\left[\mathcal{L}(f(\tau(x_i)), \hat{Y}) - \mathcal{L}(f(\tau(x_i)), \tilde{Y})\middle| i, \tau\right]\right] \tag{4.8}$$

where $f(\tau(x_i))$ is deterministic given τ and i, we have that $\tilde{Y}|i, \tau, \overset{\text{d}}{=} \hat{Y}|i, \forall i, \tau$ is a sufficient (but not necessary) condition to ensure (A) $= 0$ for all f.[1] That is, it is zero for all f if the conditional distribution on the labels is the same for both the original and transformed inputs for all possible pairs (i, τ), i.e. all possible original inputs and sampled transformations. One simple way to ensure this is to have τ always be equal to the identity mapping, so this term prefers limited transformations.

By contrast, if the transformation destroys information about the label, $\hat{Y}|i$ and $\tilde{Y}|i, \tau$ will now differ, such that, in general, (A) $\neq 0$ and, moreover, it will vary with f. Here we typically expect that (A) ≥ 0,[2] as we are making predictions using the transformed inputs, so the expected loss under the true label distribution for the transformed inputs will tend to be less than that when labels are generated using the untransformed input. To keep the magnitude of (A) low, we need to ensure that transformations maintain the conditional label distribution as well as possible, i.e. that transformations preserve all input information that is salient for predicting labels.

Conveniently, minimizing $\hat{R}(f, \phi)$ with respect to ϕ, as done by the InstaAug training setup of Section 4.3.2, will naturally try to reduce (A). Given we expect the term to typically be positive, this provides an explanation for why InstaAug can be effective without any separate consideration in the objective for the need for transformations to maintain the class label distribution.

(B) represents how well our transformation captures the true input distribution. Here we can utilize the fact that, by the definition of $\tilde{Y}$,

$$\begin{aligned}\mathbb{E}\left[\mathcal{L}(f(\tau(x_i)), \tilde{Y})\middle|\tau(x_i) = x\right] &= \mathbb{E}\left[\mathcal{L}(f(X), Y)|X = x\right] \\ &=: r(x)\end{aligned} \tag{4.9}$$

to write it as

$$\text{(B)} = \mathbb{E}[r(\tau(x_i))] - \mathbb{E}[r(X)], \tag{4.10}$$

[1] Note that $\hat{Y} \overset{\text{d}}{=} \tilde{Y}$ alone is not generally sufficient, as matching in marginal distribution does not ensure that the joint distributions with i and τ also match, in turn yielding different expectations.

[2] Note, though, that this is not formally guaranteed, even for the cross entropy loss and an f that exactly captures the true distribution. This is because, while Gibbs' inequality ensures the optimal q given p for a cross-validation expected loss $\mathbb{E}_{p(Y)}[-\log q(Y)]$ is $q = p$, in general, the optimal p given q is not $p = q$.

where $r : \mathcal{X} \mapsto \mathbb{R}^+$ maps inputs to their true expected loss. We thus see that $\tau(x_i) \overset{\text{d}}{=} X$ is a sufficient (but not necessary) condition to ensure that (B) $= 0$ for all f. That is (B) is always 0 if the process of choosing one of the training inputs at random followed by applying a sampled transformation to that input produces samples distributed exactly according to the true input distribution. Unlike for (A), there is no simple scenario in which we can ensure this is true, with the use of the identity transformation now likely to give significant discrepancies by failing to provide sufficient coverage of the input space: though the x_i may originally have been sampled from $p_{\text{true}}(X)$, there is only a finite set of them, such that repeated sampling from this finite set represents a substantially different distribution to $p_{\text{true}}(X)$. In fact, (B) nicely encapsulates the desire to perform augmentation in the first place, by showing how it can be used to increase the coverage of the input space.

How to best manage Term (B) will vary depending on the type of model used and the form of our transformations. In some situations, it may be that no matter how diverse our transformations are within the class of those allowable, $\tau(x_i)$ will still only cover a subset of the support of X. Here the most important factor for keeping (B) small will be to maximize the diversity of the transformations, e.g. by maximizing their entropy, to ensure the best possible coverage of the true input space. In other cases, it might also be possible to "over–diversify" the inputs, such that $\tau(x_i)$ can become more diffuse than X for some choices of ϕ, potentially causing training to lack focus on the particular test-time input distribution we care about. Here we may need to ensure that the entropy of the transformation does not become so large as to cause such over-diversification, creating a more complex trade-off with the need to ensure sufficient coverage. These two scenarios respectively motivate the lower and upper bounds on the transformation distribution entropy used when training the augmentation module.[3]

For augmentation of high-dimensional data, the former, coverage-limited, scenario is expected to be significantly more likely, as our original training data will generally provide quite poor coverage of the true input distribution, while our transformations will not generally be sufficiently powerful to produce unrepresentative inputs. Moreover, when working with large deep learning models, prediction in one region of the input

[3]Note here that the entropy bounds in Section 4.3.2 are on are on the entropy on the parameters of τ, rather than $\tau(x_i)$ itself. This is because it is difficult to directly control the latter during the training, with the former providing a more practical proxy that is expected to generally be representative.

space is rarely harmed by the addition of data in another input region. Thus, for the typical scenarios, we expect InstaAug to be deployed in, increasing the entropy of the transformations will directly relate to reducing the magnitude of (B). Note here that it will typically be the case that (B) < 0 provided that the transformations maintain the label distribution, as the accuracy of the downstream model will typically be higher for the transformations of the original training data that for the test data.

Term (C) is the error from the fact that we only have one sample of the label for each original training input, rather than the full label distribution. As $\hat{Y} \perp\!\!\!\perp \tau$, we have limited ability to reduce it through controlling ϕ; it essentially represents the irreducible noise in $\hat{R}(f, \phi)$ from only having a finite number of true labels. Note that it is not related to the model's ability to generalize to unseen inputs, as it is based on variability in other possible labels we might have seen for our training inputs themselves; if $Y|X$ is actually deterministic, it is exactly zero. As such, it is of limited interest for our analysis, while it will thankfully generally be much smaller than the other terms for practical problems unless we have both a very small dataset and a very noisy true label distribution.

Putting everything together, we see that (A) and (B) respectively encapsulate the competing needs of the invariance module to maintain the conditional label distribution (i.e. preserve the label information) and maximize coverage of the input space. We have also seen that the former is typically naturally taken care of by minimizing $\hat{R}(f, \phi)$ with respect to ϕ, motivating the cross-entropy term in Equation (4.2), but the latter requires separate consideration, which we deal with through the regularization term.

4.B Details of Augerino

As a method to learn invariance, Augerino [Benton et al., 2020] is quite different from the previous approaches, which usually require an extra validation set. The basic idea behind Augerino is to use a few parameters (θ) to control the transformation distribution on input images and learn these parameters with the training loss of the classifier. Specifically, it minimizes the loss

$$\mathcal{L}_\lambda(\mathbf{x}; y) \triangleq \mathbb{E}[\mathcal{L}(f(\tau(\mathbf{x})); y)] + \lambda \cdot \mathbb{R}(\theta), \tag{4.11a}$$

where $\mathcal{L}(\mathbf{x}; y)$ is the cross-entropy loss and $\mathbb{R}(\theta)$ is a regularization function on the volume of the support of the distribution weighted by the hyper-parameter λ.

Comparison with InstaAug. InstaAug shares with Augerino the ideas of tuning augmentation parameters by the classifier loss and using test time augmentation to boost performance, but they are different in the following aspects. The most significant difference is that InstaAug is instance-specific, while Augerino learns global augmentations. Besides, Augerino uses a single scalar θ to parameterize a symmetric uniform distribution ($\mathcal{U}[-\theta, \theta]$) over each type of transformations, which lacks the flexibility to model more complex augmentations, such as cropping.

In addition, Augerino uses a fixed weight λ to balance the training loss and augmentation diversity. However, we find that, in more complicated settings, this is quite impractical. Specifically, we need different λ in different stages of training. If we use a large λ from the start of training, the diversity will quickly diverge to maximum, because the classifier is very weak and the loss is consequently dominated by the diversity term. This will block the training of the classifier because transformed samples from different classes are quite mixed with each other. Otherwise, if we choose a small λ, the diversity will converge to zero after a few epochs, yielding similar results as the vanilla model without augmentation. In neither of the case can we learn a useful augmentation. Consequently, InstaAug directly constrains the diversity to keep it stable during training.

4.C Method details

4.C.1 Regression and self-supervised learning

In Section 5.2, we use classification as an example to introduce InstaAug. However, InstaAug can be easily applied to other tasks including regression and self-supervised learning. For regression, the classifier (see Figure 5.2.1) is replaced by a regressor and the loss function $\mathcal{L}$ in Equation (4.2) is changed accordingly to absolute or square error. For self-supervised contrastive learning, we replace the classifier and cross-entropy loss with the feature extractor and contrastive loss (such as SimCLR loss [Chen et al., 2020]), respectively. In addition, the sampler samples 2 rather than 1 transformations to generate multiple views for an input $\mathbf{x}$.

Algorithm 1: Location related parameterization

Input: Image $\mathbf{x}$, layer number n_layer, channel number M_i
Output: Probability of patches $\mathbf{P}_{\text{crop}}$
$\mathbf{F_0^{'}} = \mathbf{x}$;
for ($i = 1; i \leq n\_layer; i = i + 1$)
  $\mathbf{F_i}$ = Conv2d($\mathbf{F_{i-1}^{'}}$, kernel=2, stride=1, output_channel=M_i)
  $\mathbf{F_i^{'}}$ = Pooling($\mathbf{F_i}$, kernel=2) ; `// Conv and pooling`
  $\mathbf{F_i^{''}}$ = Conv2d($\mathbf{F_i^{'}}$, kernel=1, stride=1, output_channel=1) `// Concentrate info to single channel`
  $\mathbf{logit_i}$ = Flatten($\mathbf{F_i^{''}}$) ; `// Use activations of units at different layers as logits for patches of different sizes`
logits = Concat([$\mathbf{logit_i}$])
$\mathbf{P}_{\text{crop}}$ = Normalize(Exponential(**logits**))

4.C.2 Test-time augmentation

Besides augmenting data during training, the learned invariance can also be applied to test-time augmentation. Given a test image $\mathbf{x}$, we sample n different transformations τ_i from $p(\tau; \phi(\mathbf{x}))$ and apply them to $\mathbf{x}$ to generate n different views $\tau_i(\mathbf{x})$. After feeding these views to the classifier, f, we use the mean logit $\frac{1}{n}\sum_{i=1}^{n} f(\tau_i(\mathbf{x}))$ to predict $\mathbf{x}$'s label. When only learning invariance for test-time augmentation, InstaAug can be trained with a fixed pre-trained classifier at a lower computation cost.

4.C.3 Other parametrization methods

Besides the uniform and location-related parameterization, we also tried VAE-like methods to parameterize augmentations, such as cropping. The main idea is to have a Gaussian latent variable and a neural decoder to map the latent Gaussian distributions to a continuous distribution on transformation parameters (in this case, the centers and sizes of crops). However, similar to the uniform parameterization, we find the VAE-like parameterization unstable and easily stuck at local minima.

4.C.4 Network structures

In all of our experiments, we use PreActResNet-18 as the base structure of ϕ for both uniform and location-related parameterizations. When dealing with multiple transformations, for example in Section 4.5.4, we use the same network to generate parameters for all transformations simultaneously.

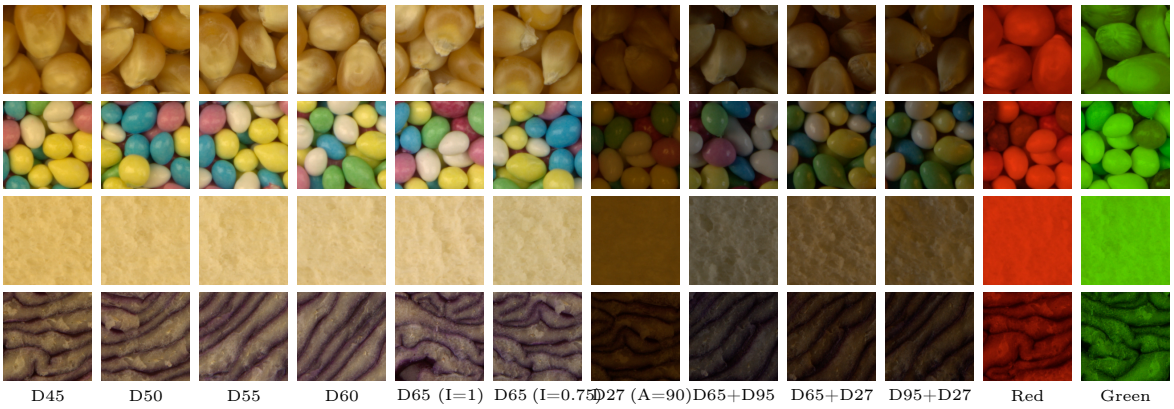


Figure 4.C.1: Examples of RawFooT data. Each row contains images in the same class (corn, candies, floor, red cabbage) under different lighting conditions. The left and right half of lighting conditions are in the easy and hard groups, respectively.

4.D Experimental details

4.D.1 Cropping

Supervised training Based on the Mixmo codebase[4] [Ramé et al., 2021], we use stochastic gradient descent (SGD) optimizer to train baselines and InstaAug. For the classifier, the initial learning rate is set to 0.2 (with momentum 0.9 and weight decay $1e-4$). A scheduler is used to decrease the learning rate by a factor of 0.9 once validation accuracy doesn't increase for 10 epochs. The learning rate of the augmentation module ϕ is fixed at $1e-5$. Batch size is set to 100 and we pre-train InstaAug for 10 epochs without augmentation. We train the model until convergence and the maximum epoch is set to 150.

Contrastive training We directly apply InstaAug on the codebase[5] from Ermolov et al. [2021]. Because of the characteristics of contrastive learning, we set the batch size to 512. Same as the supervised case, we use SGD optimizer to train the augmentation module ϕ. Differently, we use Adam optimizer [Kingma and Ba, 2015] (with learning rate $1e-3$ and weight decay $1e-6$) to train the base model. We train each model for 500 epochs and decrease the learning rate by a factor of 0.8 at step 450 and 475.

[4] https://github.com/alexrame/mixmo-pytorch.git, under Apache License v2.0.
[5] https://github.com/htdt/self-supervised.git, under Apache License v2.0.

Implementation of LRP As an example, we show how to implement location-related parameterization with a basic CNN structure in Algorithm 1.

4.D.2 Color jittering on textures

Training. We use PreActResNet-18 ($width = 1$) on texture recognition task on RawFooT and train it with SGD optimizer. The learning rate is 0.02 (with momentum 0.9 and weight decay $1e-4$) for the classifier and $1e-5$ for the augmentation module ϕ. We train each model for 50 epochs and learning rate schedulers are not necessary in this task.

Random augmentation baseline. We sweep over the variation range on each channel to find the best hyperparameters for the random augmentation baseline. For hue (h-jittering), we sweep between $[0, 0.5]$ with stride 0.1, and for saturation (s-jittering) as well as brightness value (v-jittering), we sweep between $[0, 1.0]$ with stride 0.2, which yields 216 different settings in total. The best accuracy shown in Table 4.5.3 is achieved where h,s,v= $0.0, 0.2, 0.8$.

In-distribution vs. out-of-distribution generalization. To further investigate the effect of each augmentation method, we additionally split the 46 test sets into two equally-sized groups. The first group contains lighting conditions similar to D45, such as daylight with different temperatures, for which the vanilla model without augmentation trained on D45 has high test accuracy. The second group contains lighting conditions that are dramatically different from D45, for example, pure red light, which are more difficult for the vanilla method. Then the average accuracy on the first group can be regarded as a measure of in-distribution generalization, while the accuracy on the second group reflects out-of-distribution generalization.

Group	Lighting id
Easy (1)	1-4,10,14-31
Hard (2)	5-9, 11-13, 32-46

Table 4.D.1: Splitting of Lighting conditions.

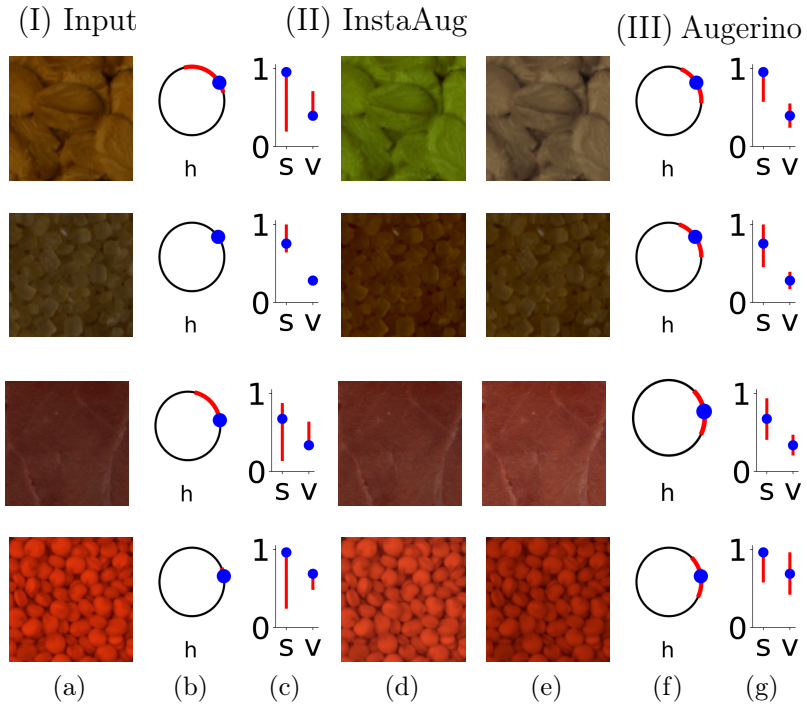


Figure 4.D.1: Examples of learned color jittering. (a) Original image; (b, f) Average hue (H) of original image (blue dot) and learned hue jittering (red arc) for InstaAug and Augerino; (c,g) learned saturation (S) and brightness value (V) of original image (blue dot) and learned hue jittering (red line segment) for InstaAug and Augerino; (d,e) examples of images transformed by InstaAug.

4.D.3 Time complexity

On a single 1080Ti, each iteration of training InstaAug on TinyIN takes 0.25s, and each epoch takes 250s. As it takes 150 epochs (1.5×10^5 iterations) for the model to converge, the total training time is about 10h. For random augmentation, each iteration takes 0.15s, and each epoch takes 150s. It takes the same 150 epochs (1.5×10^5 iterations) to converge, so the total training time is about 7h. All of the other augmentation learning approaches are slower than random augmentation, with some of them being noticeably slower than InstaAug itself. For example, the exploitation stage of AdaAug has the time complexity as random augmentation, which is 0.15s/iter, 150s/epoch, and 7h for the whole training process. However, its exploration took us more than 15h because it requires averaging the representations of a large number of augmented samples.

The training speed of InstaAug on RawFoot (color jittering) is similar to random augmentation (0.37s/iter vs. 0.40s/iter), though it takes more epochs (about 40) compared with random augmentation, which usually converges after 25 epochs.

4.E Additional results and discussion

4.E.1 RawFooT

Figure 4.D.1 shows some examples of learned color jittering. Though it's not easy to fully understand them, we can still find some patterns. For example, InstaAug tends to increase the brightness of darker images (row 1 and 3) and decrease the brightness of brighter images (row 4). Also, InstaAug is more likely to change saturation compared with hue and brightness, which is consistent with the common belief that saturation contains less information than hue and brightness.

H_{min}	H_{max}	Accuracy (%)
0.0	0.5	52.12
0.5	1.0	61.28
1.0	1.5	62.91
1.5	2.0	64.39
2.0	2.5	65.04
2.5	3.0	65.05
3.0	3.5	**66.03**
3.5	4.5	65.60
4.0	4.5	64.35
4.5	5.0	64.17
0.0	1.0	51.78
1.0	2.0	63.96
2.0	3.0	65.25
3.0	4.0	**65.78**
4.0	5.0	64.23

Table 4.E.1: Model performance with different choice of H_{min} and H_{max} on supervised cropping.

InstaAug's behavior is quite different on different samples. It even decides not to augment the H and V channels of the image in the second row. In comparison, Augerino adds or multiplies noise to each channel with the same distribution across all samples, which is harmful in many cases. For example, the input image in the last row is already very bright. but Augerino allows further increasing its brightness. Then brightness values of many pixels will be capped at 1.0, which leads to loss of information.

4.E.2 Hyperparameter ablation

The two hyperparameters of InstaAug are H_{min} and H_{max}, which reflect human preference on augmentation diversity. To investigate how H_{min} and H_{max} influence model performance and provide a guide on how to choose them, we perform an ablation

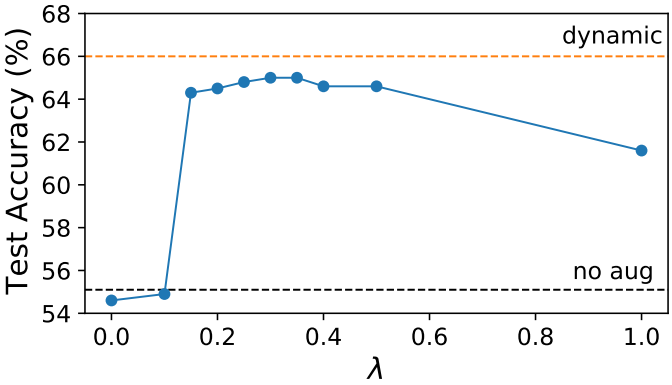


Figure 4.E.1: Model performance with fixed λ on supervised cropping, compared with dynamic tuning λ used in InstaAug.

study for the experiment of Section 4.5.2, wherein we sweep over possible intervals of length 0.5 and 1.0. From Table 4.E.1, we find that the best accuracy is achieved when $[\mathrm{H}_{\min}, \mathrm{H}_{\max}]$ is set to $[3, 3.5]$, while any sub-interval of $[2, 4]$ produces significantly better results compared with random augmentation.

To show the effect of dynamically tuning λ in InstaAug, we compare it with results with fixed λ in Figure 4.E.1. We find that for small $\lambda \leq 0.1$, the entropy term is nearly 0 throughout training, which gives a result similar to no augmentation. For large $\lambda \geq 0.5$, the model suffers from excessive augmentations throughout training which hinders the training of the classifier and InstaAug module. There is also a performance plateau for $0.15 \leq \lambda \leq 0.5$, whose accuracy is between 64.0 and 65.0. However, even the best of them is not as good as dynamic tuning, which is probably a result of their inability to keep transformation diversity stable during different stages of the training process.

4.E.3 Why is the random augmentation baseline so strong?

It is perhaps initially surprising that the Random Augmentation baseline in 4.5.2 is so strong compared to the other global augmentation schemes. In short, this occurs because the extensive hyperparameter sweep used for it turns out to be a more effective tuning mechanism than directly training global parameters simultaneously to the model. To be more precise, for any *global* cropping scheme (which includes random crop, Augerino, and InstaAug without input), there is little to be gained from using a non-uniform distribution on the position of the crops. As such, the only thing that

can be usefully learned is the distribution on the *size* of the crops themselves. For the random crop baseline, we do an exhaustive sweep to establish the best distribution on crop sizes, meaning that this baseline represents a near-optimal global cropping augmentation. By comparison, InstaAug (without input) must still learn the optimal cropping size distribution during training, and the results suggest that it does not always manage to do this perfectly, tending to prefer under-diverse transformations. This is perhaps not surprising, as it does not have access to a validation set, unlike the hyperparameter sweep implicitly being deployed for the random crop baseline. The problem is seen even more starkly for Augerino, where the lack of LRP causes training to become stuck in highly sub-optimal local optima that yield very little transformation diversity.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Learning instance-specific augmentations by capturing local invariances.
Publication Status	☑ Published □ Accepted for Publication □ Submitted for Publication □ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Ning Miao, Tom Rainforth, Emile Mathieu, Yann Dubois, Yee Whye Teh, Adam Foster and Hyunjik Kim. Learning instance-specific augmentations by capturing local invariances. International Conference on Learning Representations (ICLR), 2023

Student Confirmation

Student Name:	NING MIAO		
Contribution to the Paper	Ning is the lead author of this paper. He proposed the ideas, wrote all the codes, did all the experiments and wrote the manuscript.		
Signature	Ning Miao	Date	16/06/2024

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Tom Rainforth, Senior Researcher in Machine Learning		
Supervisor comments Ning led the paper, making a substantial contribution as per the description above.		
Signature	Date	20/06/2024

This completed form should be included in the thesis, at the end of the relevant chapter.

Chapter 5

SelfCheck: Using LLMs to Zero-Shot Check Their Own Step-by-Step Reasoning

5.1 Introduction

Recent years have witnessed dramatic changes in the areas of NLP and AI brought on by significant advances in LLMs. From GPT-3 [Brown et al., 2020], PaLM [Chowdhery et al., 2022], Llama [Touvron et al., 2023] and Falcon [Almazrouei et al., 2023] to GPT-4 [OpenAI, 2023] and PaLM-2 [Google, 2023], the increasing model sizes and exploding amount of training data have empowered LLMs to achieve human-level performance on a large range of tasks, including summarization, translation, and question answering. The invention of Chain-of-Thought prompting (CoT, Wei et al. [2022]) has further enhanced LLMs' ability to solve complex problems by generating step-by-step solutions.

However, the performance of even the largest LLMs is still unsatisfactory on more difficult reasoning problems. For example, GPT-4 with CoT prompting only correctly answers 42.5% of problems in the MATH dataset [Bubeck et al., 2023, Hendrycks et al., 2021], which is far below the human level. Such problems require careful multi-step reasoning to solve, and LLMs are consequently prone to make mistakes: even though their error rate on individual steps may be low, the probability of generating at least one erroneous step can still be quite high, undermining the final answer.

Recent works have tried to overcome this limitation by checking for errors in these step-by-step solutions [Cobbe et al., 2021, Li et al., 2022, Ling et al., 2023, Jiang et al., 2023]. Such checks can then be used to provide confidence scores in answers and select between different possible alternatives. This checking has typically been performed either by using an external verification model [Cobbe et al., 2021, Lyu et al., 2023, Peng et al., 2023], or through few-shot in-context learning [Brown et al., 2020] of an LLM [Weng et al., 2022, Ling et al., 2023].

Unfortunately, existing methods generally require extra training data and/or domain-specific exemplars, which often makes them inconvenient to use in practice and restricts them to specific domains or data formats. The aim of our work is thus to instead provide a general-purpose, zero-shot, approach to checking that relies only on the original LLM, without the need for additional external resources.

To this end, we introduce SelfCheck, a zero-shot step-by-step checker for self-identifying errors in LLM reasoning chains. SelfCheck uses the LLM to individually check the conditional correctness of each step based on directly related information from the question and the preceding steps, in a manner similar to a human going back to check their work. The results of these individual checks are then integrated to form an overall correctness estimation for the whole reasoning chain.

Key to SelfCheck's success is a novel mechanism for performing the checking of individual steps. As we will show, the naive approach of directly asking the LLM to check a step is typically ineffective. Instead, we introduce a multi-stage approach that breaks the problem down into a series of simpler tasks, leverages the generative strengths of the LLM, and decorrelates errors between the original generation and checking. Specifically, using separate calls to the LLM we first extract the target and relevant context for the step, then regenerate an independent alternative step from these, and finally compare the two. The original step is then deemed to pass the check if it matches the regeneration.

Besides providing an estimation of correctness for each solution, SelfCheck can also boost final answer accuracies for the original questions by weighted voting. Namely, given multiple solutions to a question, it uses confidence scores as weights to vote among the answers, which provides a soft way to focus on more accurate solutions.

We evaluate SelfCheck on three math tasks and one logical reasoning task. For all datasets, we find that using SelfCheck achieves a significant increase in final answer

accuracies compared with simple majority voting and other baselines. We also see that SelfCheck provides an accurate confidence estimation for LLM's solutions, which decreases the proportion of incorrect solutions by between 9% and 23% (depending on dataset) when filtering out solutions with low confidence scores. We further perform a number of ablations to justify some of our key design choices in the SelfCheck approach.

To summarize, we introduce SelfCheck as a novel and effective zero-shot schema for self-checking step-by-step reasoning in LLMs. Unlike previous methods, SelfCheck does not need any finetuning or example crafting, so can be directly applied to reasoning tasks in different domains. Our experiments confirm that it can, in turn, be used to improve final predictive performance of LLMs.

5.2 SelfCheck: using LLMs to check their own reasoning

Rather than relying on external resources or problem-specific data like the aforementioned approaches, it would be highly beneficial if we could develop self-contained checking schemes that require only the original LLM itself. In other words, we would like to use the LLM to identify errors in its own step-by-step reasoning, analogously to how a human might go back to check their working.

Unfortunately, directly asking the LLM to check its own reasoning is largely ineffective: it almost invariably declares that the original answer is correct, with Ling et al. [2023] finding answers checked in this way are deemed correct more than 90% of the time regardless of whether they actually are. As we will show in Section 5.5, individually prompting the LLM to check each step in the CoT reasoning fares slightly better, but is still only able to offer marginal gains compared to not checking at all.

A more nuanced method to perform this checking is thus required. To this end, we introduce **SelfCheck**, a general-purpose, zero-shot, checking schema for self-identifying errors in LLM CoT reasoning. Given a question, q, and its step-by-step solution, s, produced by some **generator** (which will generally be an LLM with appropriate CoT prompting), SelfCheck considers each step of s in turn and tries to establish its individual correctness based on the preceding steps. This checking is done by leveraging an LLM (which can either be the same LLM used to generate s or a separate one), but rather than directly asking the LLM to perform the check, we

instead introduce a novel step checking method (see Section 5.2.1) that exploits their generative modeling strengths. The results of the checks on individual steps are then combined into a single confidence score, $w \in [0, 1]$, for the whole solution. These confidence scores, in turn, allow us to improve predictive performance, by using them to perform weighted voting on multiple solutions to the same question.

5.2.1 Step checking

To check individual steps of the reasoning process, the first thing we should note is that the correctness of each step is highly dependent on its context, namely the question and previous steps in the solution. For example, we usually need to refer to previous steps for the definition of variables and the meaning of specific numbers. If each step is conditionally correct based on the provided context and the last step provides an answer in the required format, then the overall reasoning will itself be correct. The target of the step checking is thus simply to check the conditional correctness of each step based on the context provided by previous steps. That is, we only care about catching errors at the current step, and can assume all information from its context to be correct.

A simple idea to try and achieve this would be to feed the current step as well as all its context to an LLM and directly ask it to 'check the correctness of the step'. However, in practice, we find that this task is too difficult for the LLM to do effectively, even with careful prompting that exemplifies how to do the checking in detail (see Section 5.5). This difficulty comes first from the fact that there are multiple aspects to the checking problem that the checker must deal with simultaneously: it needs to understand the key content in the step and then collect all related information from the context, before actually checking for its correctness. Second, LLMs are trained as generative models, rather than directly as supervised approaches for checking, such that it is a problem that does not necessarily play to their strengths. Finally, there are likely to be strong correlations between the errors such a checker will make with the errors made in the original generation, undermining its usefulness.

To address these difficulties, SelfCheck instead decomposes the checking task for each step into four stages: *target extraction*, *information collection*, *step regeneration*, and *result comparison*. The LLM is used to execute each stage successively, with the outcome of the result comparison providing the correctness prediction.

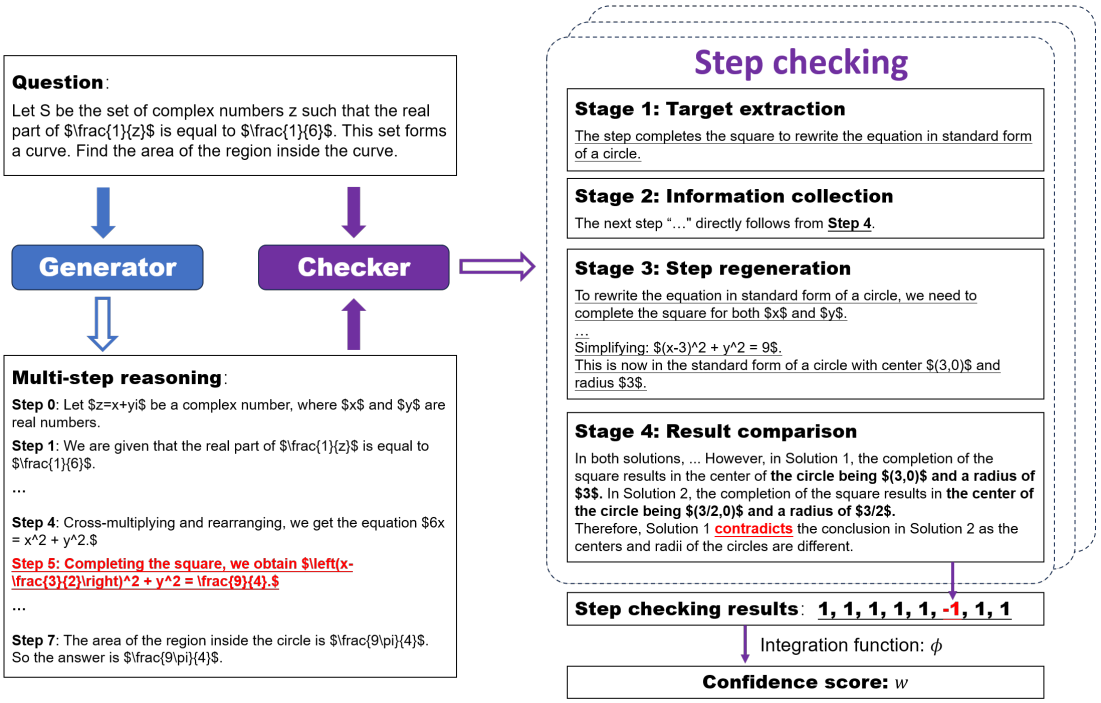


Figure 5.2.1: Example of using SelfCheck, focusing on the checking of a particular step (Step 5). To check the correctness of the step, SelfCheck goes through 4 stages. First, in the target extraction stage, it figures out that the main purpose of Step 5 is to complete the square. In the information collection stage, it then establishes that Step 5 only directly relies on Step 4. Next, the step regeneration stage instructs the LLM to complete the square independently, only using Step 4 as context. The regeneration result shows that the center and radius of the circle are $(3, 0)$ and 3, which is different from what is implied by the original Step 5. Consequently, the result comparison stage concludes that Step 5 is likely to be wrong. After checking all the steps, SelfCheck integrates the results to form an overall confidence score, w. See Section 5.A for a complete version of the example.

The idea behind this decomposition is to make the LLM focus on an easier task at each stage and ensure the individual tasks carried out are more closely aligned to the LLM's strengths. Moreover, by focusing on regenerating and then comparing, we hope to reduce the correlations between the errors of the checking and the original generation.

At a high level, the stages work by first prompting the LLM to figure out the target of the current step and what information it uses to achieve the target; we find that the LLM is usually able to perform these tasks extremely accurately. Then we ask the LLM to re-achieve the target using only the collected information, providing an alternative to the original step that maintains the same purpose in the overall

reasoning process. Here the clear description of the target and the simplified context we provide make the regeneration stage less challenging. As a result, we hope its output will be more reliable and thus serve as a useful reference. Even if this is not the case, it will still hopefully provide a viable alternative, with a distinct generation, that can be used for comparison. The last stage then uses the LLM to compare the original step with the regenerated output. If their main conclusions match/mismatch, this provides evidence that the original step was correct/incorrect.

A worked example of this step-checking process is provided in Figure 5.2.1. In the following, we describe each of the subtasks in detail and provide our specific instructions to the LLM. We note here that the different LLM queries are made independently, rather than keeping the queries and answers from previous stages in context. Thus, for example, when the LLM is called to carry out the step regeneration, it does *not* have access to the original generation. The same prompts are used across LLMs and datasets, thereby providing a general-purpose approach.

Target extraction To check a step (for example, Step 5 in Figure 5.2.1), we first need to figure out what the step is trying to achieve. Without a specific target, the regeneration stage would proceed in a random direction, making it impossible to serve as a reference to the original step. We thus use the LLM itself to extract the target of a step using the question and all previous steps (Steps 0-4 in Figure 5.2.1) with the following prompt (we omit some line breaks due to space limitations):

> *The following is a part of the solution to the problem [Question]: [Step 0,..., Step i]. What specific action does the step [Step i] take? Please give a brief answer using a single sentence and do not copy the steps.*

During execution, we copy the question and steps into [Question] and [Step 0, ..., Step i] to form the actual input to the LLM. The reason for requesting a brief answer is to try and keep the amount of information retained to the minimum needed, thereby avoiding unnecessary influence on the regeneration and hopefully reducing correlations in errors in turn.

Information collection To reduce the difficulty of the regeneration stage and avoid unrelated information from affecting the result, we filter out information that is not directly related to the current step. Specifically, we ask the LLM to select useful items from the question and all previous items with the following prompt, where [Information j] is simply the j-th sentence in the question:

This is a math question: [Question]. The following is information extracted from the question:
Information 0: [Information 0] Information 1: [Information 1] ...
The following are the first a few steps in a solution to the problem:
Step 0: [Step 0] Step 1: [Step 1] ... Step i-1: [Step i-1]
Which previous steps or information does the next step [Step i] directly follow from?

After retrieving the free-text response from the LLM, we extract step or information ids by regular expression. For example in Figure 5.2.1, the current step requires Step 4 and no information from the question as context. The selected steps and information are then fed into the regeneration stage.

Step regeneration Given the target and necessary information of the step, we can now ask the LLM to achieve the target independently with only the collected information, without seeing the original step. Because the step is usually a small jump from previous conclusions, and the information collection stage has already filtered out irrelevant information, we can usually trust regeneration results. The prompt for this stage is:

We are in the process of solving a math problem. We have some information from the problem:
Information 0: [Information I_0] Information 1: [Information I_1] ...
The following are some previous steps: Step 0: [Step S_0] Step 1: [Step S_1] ...
The target for the next step is: [Target]
Please try to achieve the target with the information from the problem or previous steps.

Here [Target] is the output from the target extraction stage. [Information I_i] and [Step S_i] correspond to the specific items selected by the information collection stage. In Figure 5.2.1, only Step 4 and no information from the question is directly related to the current step, so SelfCheck simply copies the content of Step 4 into [Step S_0] and removes the block containing [Information I_i].

Result comparison The last step is to compare results from the regeneration stage and the original step with the following prompt:

The following are 2 solutions to a math problem. Solution 1: [Regeneration output] Solution 2: [Step i]
Compare the key points from both solutions step by step and then check whether Solution 1 'supports', 'contradicts' or 'is not directly related to' the conclusion in Solution 2. Pay special attention to the difference in numbers.

If the regeneration output 'supports'/'contradicts' the original step, we can conclude that the original step is likely correct/incorrect respectively. Sometimes, the correctness

of the original step cannot be directly inferred from the regeneration output. For example, when the target is to simplify an equation, then there may be multiple valid solutions. In such cases, we are not sure about the correctness of the original step, which makes 'is not directly related to' the third possible outcome of the check.

5.2.2 Results integration

After running step-checking and getting a checking result for each step, we need an integration function ϕ to give a confidence score, $w \in [0, 1]$, for the overall correctness of the solution. The input of ϕ should be a vector in the form of $[r_0, r_1, ..., r_n]$, where each item r_i represents the step checking result for Step i. We will use $r_i = -1, 0$, and 1 to represent the step-checking results 'contradict', 'is not directly related to' and 'support' respectively. We find that the following simple integration function works well in practice

$$w = \phi([r_0, r_1, ..., r_n]) = 2 * \text{Sigmoid}\left(-\lambda_{-1} \sum_{i=0}^{n} \mathbb{1}_{r_i=-1} - \lambda_0 \sum_{i=0}^{n} \mathbb{1}_{r_i=0}\right), \tag{5.1}$$

where λ_{-1} and λ_0 are two non-negative hyperparameters with $\lambda_{-1} > \lambda_0$; we found nearly any sensible pairs of λ_{-1} and λ_0 worked similarly well in preliminary tests, so we simply fix $\lambda_{-1} = 1$ and $\lambda_0 = 0.3$ throughout our experiments. The rationale of this setup is that the more failed checks we see, the more likely the overall reasoning process, and thus the final solution, are wrong. Note here that, because the checks are themselves imperfect, we do not necessarily want to immediately reject the whole solution from a single step-check failure, especially for $r_i = 0$ cases. This is why we take a 'soft' approach to the verification with a confidence score. The number of successful checks, i.e. $\sum_{i=0}^{n} \mathbb{1}_{r_i=1}$, is deliberately not included in our integration function as an increased number of successful checks does not actually increase our confidence in the overall solution: shorter reasoning chains are generally preferable to longer ones for a given question and LLM.

Once calculated, the resulting confidence score can be directly used as a weight for voting between different possible solutions. We can thus use SelfCheck to increase the accuracy of an LLM's answers by generating multiple possible solutions, calculating confidence scores for each, and then choosing our final answer through weighted voting.

5.3 Related work

How to automatically check the correctness of a sequence of reasoning steps is a long-standing question. We now discuss how previous methods have tried to tackle this in an LLM context. We note that none of these works are able to work in the zero-shot setting covered by SelfCheck, requiring either problem-specific examples, an external model, and/or finetuning.

Few-shot verification Though our focus will be on zero-shot checking, for some problems one may have hand-crafted exemplars available that are specifically designed to that particular question-answering task. Previous methods have been designed to perform checking of LLMs' generated solutions in this few-shot checking scenario.

For example, the Self-Verification (SV) approach of Weng et al. [2022] verifies the whole solution by backward prediction. That is, it uses the conclusion from CoT reasoning to predict a masked condition in the question. However, it only supports single-step checking and is based on the assumption that every piece of information in the question can be recovered using a correct solution of it, which is often not the case. Consequently, it is only applicable to simpler tasks, such as GSM8K.

The Deductive Verification (DV) approach of Ling et al. [2023] instead looks to verify independent sub-tasks, as per SelfCheck. However, its verifier only supports checking reasoning chains in a special format called Natural Programs. As a result, it can only work with a specific specialised generator, without serving as a general verifier for multi-step reasoning.

Verification with external resources In some cases, there might be external resources available to verify the logical correctness or faithfulness of LLM outputs. Lyu et al. [2023] translate a question into a symbolic reasoning chain using an LLM and solve the problem by a symbolic logic solver. Peng et al. [2023] introduced an external database to check for incorrect knowledge in LLM outputs. These methods are limited by the availability of external resources and are typically restricted to checking for certain types of errors.

Training/finetuning a verifier A few other methods train or finetune a separate verifier model to check reasoning chains. Cobbe et al. [2021] finetuned a GPT-3 model on GSM8K to predict the correctness of a solution as a whole. Li et al. [2022] trained a binary *deberta-v3-large* [He et al., 2020] classifier on each domain to predict step correctness. More recently, Lightman et al. [2023] built a large dataset, which contains step-wise correctness labels from human labelers, and finetuned a GPT-4 model on it. Unlike SelfCheck, all of these methods require extra data and external computational resources, restricting their applicability and ease of use.

5.4 Experiments

We now run experiments on three math-reasoning datasets to evaluate SelfCheck's effectiveness in checking multi-step reasoning and improving final answer accuracies. Note here that our focus on math-reasoning problems is due to ease of performance evaluation and dataset availability; SelfCheck is directly applicable to other question-answering problems with nominal changes to our prompts. An additional experiment based on a logic reasoning task is provided in Section 5.D.

Datasets GSM8K [Cobbe et al., 2021], MathQA [Amini et al., 2019], and MATH [Hendrycks et al., 2021] consist of math problems on primary school, middle school, and competition levels, containing 1319, 2985, and 5000 test samples, respectively. For GSM8K and MathQA, we evaluate SelfCheck on the whole test sets. Due to limited resources, we use a subset of MATH test set taken from Ling et al. [2023].[1] Besides the levels of difficulty, the three datasets differ from each other in the following aspects. Firstly, MathQA provides 5 options to choose from for each problem, while GSM8K and MATH have no options. Secondly, GSM8K only has arithmetic problems, while MathQA and MATH contain more diverse problems in geometry, physics, probability, and algebra.

LLMs We use GPT-3.5 (gpt-3.5-0301) and GPT-4 (gpt-4-0613) as our LLMs, focusing in particular on the former due to budget restrictions. Note that the same prompts are used for all datasets with both LLMs during evaluation; no dataset-specific customization or tuning has been performed. When devising the prompts, a small number

[1] https://github.com/lz1oceani/verify_cot/tree/main/results/chatgpt3.5/natural_program/MATH_np.json

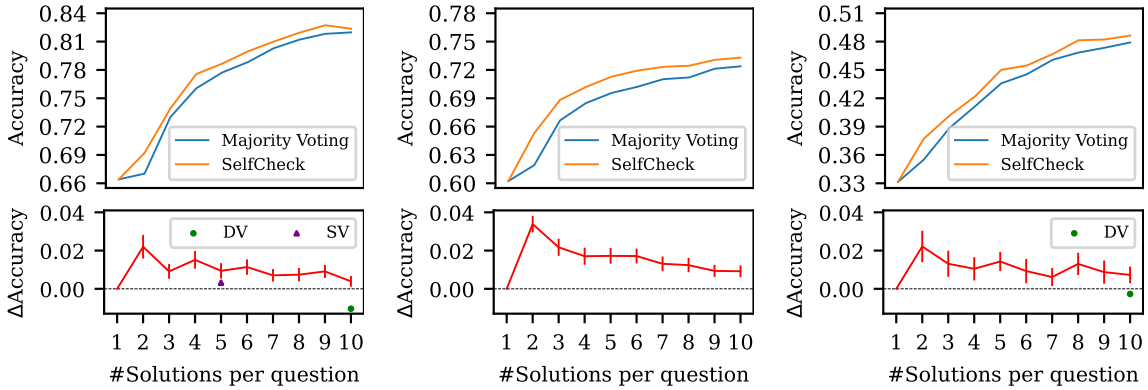
of training samples from MathQA dataset were utilized. An additional experiment using Llama2 (70B, 4-bit, Touvron et al. [2023]) is provided in Section 5.E.

Baselines We use majority voting (also known as Self-Consistency Decoding [Wang et al., 2022] in the context of CoT reasoning) as our main baseline following Ling et al. [2023] and Lightman et al. [2023]. Despite its simplicity, this is still quite a strong baseline in the current literature. In particular, most existing few-shot methods report similar results compared with it [Weng et al., 2022, Ling et al., 2023]. We also compare with previously quoted results from Self Verification (SV, Ling et al. [2023]) and Deductive Verification (DV, Weng et al. [2022]) when possible. We note though that these approaches are not directly comparable to SelfCheck in general, as they require additional exemplars which will often not be available in practice. Despite this, we will find that SelfCheck outperforms them when comparisons are possible.

We omit results from Faithful-CoT [Lyu et al., 2023], because it has already been shown to decrease the accuracies on GSM8K and MATH by 11.8% and 4.2%, respectively compared to majority voting [Ling et al., 2023]. It is also impossible for us to compare with training/finetuning based methods such as Lightman et al. [2023], because we have neither access to their finetuned models nor computation resources to repeat their training/finetuning. The significant extra data and resources they require also means their contributions are somewhat tangential to SelfCheck regardless.

5.4.1 Final answer correctness

Figure 5.4.1 shows the performance gains using the confidence scores from SelfCheck to do weighted voting compared with baseline methods. The upper plots show that accuracies of both SelfCheck and majority voting have the same increasing tendency as the number of generated solutions per question increases, which is a result of the variance reduction provided by averaging over more solutions. The bottom plots show the difference in accuracy between the two including the standard error in the estimate. We can see that by allocating higher weights to correct solutions, SelfCheck achieves significantly higher accuracies than majority voting for all solution numbers per question. We also find the improvements of SelfCheck (compared with majority voting) to be higher than Deductive Verification and Self-Verification in their reported settings, despite the use of in-context learning from additional examples. We will



(a) GSM8K (b) MathQA (c) MATH*

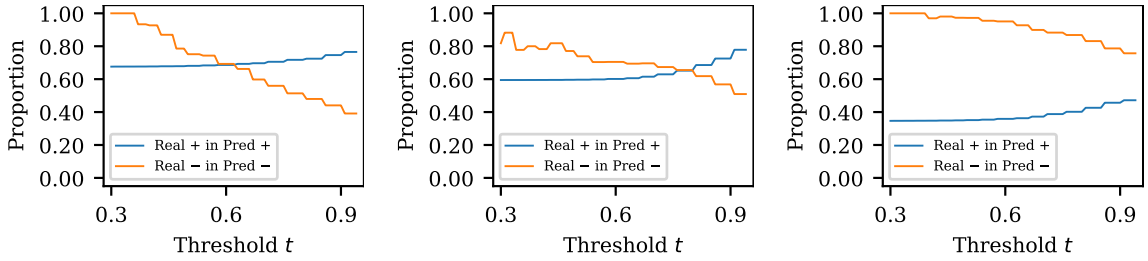
Figure 5.4.1: The upper plots show the accuracies of SelfCheck and majority voting for different numbers of generated solutions per question with GPT-3.5. The lower plots show the accuracy gaps between each method and majority voting, where DV and SV stand for Deductive Verification [Weng et al., 2022] and Self-Verification [Ling et al., 2023], respectively. It is difficult to compare with DV and SV with respect to absolute accuracies because they are using different generator models. However, we can see that SelfCheck achieves higher relative performance gains than both in their reported settings.

Dataset	Generator	Checker	✗✗ (%)	✗✓ (%)	✓✓ (%)	Acc(MV, %)	Acc(SelfCheck, %)	ΔAcc (%)
GSM8K	GPT-3.5	GPT-3.5	16.8	23.0	60.2	71.7	74.3	2.8±0.9
	GPT-4	GPT-4	8.8	8.2	83.0	87.1	86.9	-0.2±0.2
	GPT-4	GPT-3.5	8.8	8.2	83.0	87.1	**88.1**	1.0±0.3
MathQA	GPT-3.5	GPT-3.5	27.6	26.4	46.0	59.2	64.6	5.4±1.1
	GPT-4	GPT-4	16.2	11.0	72.8	78.3	80.9	2.6±0.4
	GPT-4	GPT-3.5	16.2	11.0	72.8	78.3	**81.2**	3.0±0.4
MATH*	GPT-3.5	GPT-3.5	52.6	23.2	24.2	35.8	38.0	2.2±0.7
	GPT-4	GPT-4	42.0	20.2	37.8	47.9	**51.3**	3.4±0.6
	GPT-4	GPT-3.5	42.0	20.2	37.8	47.9	48.9	1.0±0.8

Table 5.4.1: SelfCheck significantly increases final answer accuracies with both GPT-3.5 and GPT-4, even we only have 2 candidate solutions for each question. ΔAcc is the performance gain of SelfCheck compared with majority voting (MV), with the ± indicating the standard error. ✗✗, ✗✓and ✓✓represent the proportions of questions with 0, 1 or 2 correct solutions. Unlike in Figure 5.4.1, Acc(MV) here is essentially the average accuracy of 2 single generations. We see that the gains from SelfCheck are typically larger in cases where it is common for only one of the solutions to be correct, as these are the cases using weighted voting can influence the final answer.

perform additional ablations on how performance changes when ensembling over a larger number of solutions in Section 5.5.1.

To investigate the effect of using more powerful LLMs, and of using a different LLM for the generation and checking, we further conducted experiments with GPT-4 and a mix of GPT-4 and GPT-3.5. Because of the high cost of calling the GPT-4 API, we



(a) GSM8K (b) MathQA (c) MATH*

Figure 5.4.2: When raising the classification thresholds t, the proportions of real correct solutions in predicted correct solutions (Real + in Pred +) increase for GSM8K (67.5%→76.5%), MathQA (59.4%→82.2%) and MATH (34.6%→50.8%).

randomly sample 500 questions from each dataset to form the test sets and generate 2 (instead of 10) answers to each question. In Table 5.4.1, we see that SelfCheck significantly outperforms majority voting with both GPT-3.5 and GPT-4. We also notice that using GPT-3.5 to check GPT-4 generated answers yields surprisingly good results, actually outperforming checking with GPT-4 on the simpler GSM8K and MathQA tasks. This is likely because using different LLMs helps to further decorrelate the errors of the generator and the checker, and shows that using a cheaper LLM can still often be sufficient for the checking. For the more difficult problems in MATH, using GPT-4 as checker always produces better results, but even here the checking from GPT-3.5 is beneficial compared to doing no checking at all.

5.4.2 Verification performance

Besides serving as a confidence score calculator to improve the performance of voting, SelfCheck can also predict the correctness of a single solution. To do so, we simply set a threshold t to the confidence score, where solutions with confidence scores $w \geq t$ are classified as correct.

Figure 5.4.3 shows the ROC curves for each dataset. As a comparison, directly prompting GPT-3.5 to verify whole reasoning chains leads to no meaningful control on the false and true positive rates (FP and TP): they are always both 100% on MATH and 98%

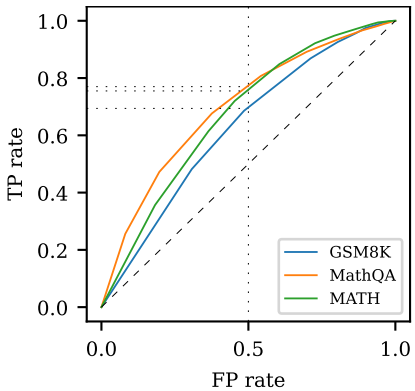


Figure 5.4.3: True positive rates (TP) vs. false positive rates (FP) as classification threshold, t, is varied.

on GSM8K, as observed by Ling et al. [2023]. In other words, the checker always predicts the answer as correct, providing no useful information.

As well as verification accuracies, we may also care about the solution quality after filtering out solutions with low confidence scores w. Figure 5.4.2 shows that by increasing the threshold t, SelfCheck can filter out more incorrect solutions, such that a higher proportion of the solutions that pass the check are indeed correct (Real + in Pred +). Though this is at the cost of misclassifying more of the real correct solutions as incorrect, this can be a useful feature in cases where the risk of choosing an incorrect solution is higher than rejecting a correct one.

5.5 Analysis

We now perform some ablations to justify some of the key design choices made by SelfCheck and provide insights on its behavior. Limited by budget and time, all experiments in this section are performed on a subset of the MathQA test set with 100 randomly selected questions.

5.5.1 More solutions per question?

Serving as a method to reduce variance, majority voting increased final answer accuracies on different datasets when we increased from 2 to 10 solutions in Figure 5.4.1. In cases where we only care about final predictive performance and majority voting can be applied (see Section 5.D for an example where it cannot, but SelfCheck can), one might thus question whether it is better to simply use our computational resources to keep increasing the size of this ensemble, rather than relying on a checking scheme. Note here that the cost of running the verifier is around twice that of the original generation for zero-shot generators (it can cost less than the original generation for few-shot generators with longer prompts).

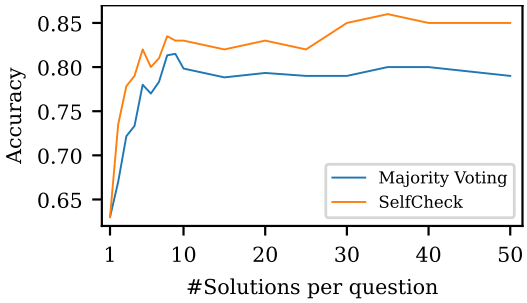


Figure 5.5.1: SelfCheck achieves significantly higher final answer accuracies than majority voting for large ensembles of solutions.

However, as shown in Figure 5.5.1, the improving performance from using a larger ensemble saturates for majority voting, with the accuracy never going above that achieved when $n = 9$, thereby never reaching the performance we already achieved by SelfCheck for an ensemble of size $n = 5$. Moreover, the performance of SelfCheck continues to increase as the ensemble grows. By lowering the weights (confidence) of incorrect solutions, SelfCheck increases the chance of selecting the correct answers, even when their generation probabilities in the generator LLM are low. Therefore, with SelfCheck, LLMs can effectively rectify their own biased beliefs by themselves.

5.5.2 Albation studies

In order to pick apart the effect of several critical design choices for SelfCheck, we compare SelfCheck with some of its variants with respect to final answer and verification accuracies on MathQA.

Global v.s. step-by-step checking The first question is can we simply ask an LLM to check the whole solution without taking steps into consideration. To answer it, we prompt the LLM to perform global checking with the following instruction:

> *The following is a question and a solution to it from a student. Carefully check whether the solution is correct step by step. End your response with your conclusion that starts with "Correct", "Wrong" or "Not Sure".*
> *Question: [Question] Solution: [Step 0, Step 1,..., Step n]*

Similar to the findings of Ling et al. [2023], we find that the global checker outputs "correct" most of the time and rarely recognizes an error. Consequently, its final answer accuracies are very close to majority voting (in Figure 5.5.2) and its verification accuracy (55.0%) is only marginally above random guess (50.0%). This lack of ability to deal with the difficulty of global checking is what makes step checking necessary.

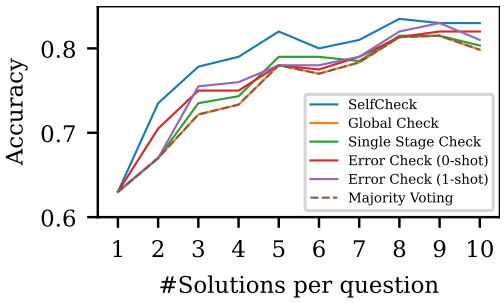


Figure 5.5.2: Generation accuracies for variants of SelfCheck on MathQA with GPT-3.5.

Single-stage v.s. multiple-stage step checking Next, we ask whether we really need to decompose the step checking into several stages. To answer this, we design the following prompt to use the LLM directly.

The following is a question and the first a few steps in its solution.
Question: [Question] Solution: [Step 0, Step 1,..., Step i-1]
Check the correctness of the next step: [Step i]
Please consider the information it relies on and check step by step. Please end your response with your conclusion that starts with "Correct", "Wrong" or "Not Sure".

Figure 5.5.2 and Table 5.5.1 show that although this is better than global checking, it is still significantly worse than SelfCheck with its multi-stage checking. This indicates that checking a step in a single stage is still too challenging for the LLM, so it is necessary to further decompose step checking into a pipeline of easier sub-tasks.

Method	Accuracy (%)
SelfCheck	**66.7%**
Global Check	55.0%
Single stage Check	57.2%
Error Check (0-shot)	63.1%
Error Check (1-shot)	64.2%

Table 5.5.1: Verification accuracies for variants of SelfCheck on MathQA with GPT-3.5. The reported verification accuracy is the average of true positive and true negative rates.

Error check v.s. regenerate and compare We now justify the choice to perform step regeneration and comparison instead of direct error checking for each step. To do so, we replace our regeneration stage and comparison stage with a single error-checking stage. We first compare with a zero-shot version of the variant with the following prompt:

Given the following information:
Information 0: [Information I_0] Information 1: [Information I_1] ...
Step 0: [Step S_0] Step 1: [Step S_1] ...
Check the correctness of the next step [Step i]
Please check for grounding errors, reasoning errors and calculation errors step by step. Please end your response with your conclusion that starts with "Correct", "Wrong" or "Not Sure".

We then add an examplar from Ling et al. [2023] (see Section 5.B) to make a more powerful one-shot error checker. However, results in Figure 5.5.2 and Table 5.5.1 show that even with a very detailed and instructive example, direct error checking still performs worse than our regenerate and compare approach, which supports our previous argument that LLMs are better at generation than checking.

5.6 Conclusions

In this paper, we have introduced SelfCheck, a general-purpose, zero-shot, step-by-step checking scheme for LLMs. Unlike previous approaches, SelfCheck does not require any additional data or external resources: it uses the LLM to identify errors in its own reasoning, leveraging a novel regenerate-and-compare approach. By using the results of this checking to perform weighted voting over different solutions, we find that SelfCheck is able to, in turn, increase final predictive accuracy.

Appendix

5.A A complete example of SelfCheck

To help readers get a better idea of how SelfCheck works, we show a complete version of the example in Figure 5.2.1, including the exact inputs and outputs of different stages. Please note that we have compiled all latex codes for readability, though the inputs and outputs to LLMs are in their original form. The question is

> *Let S be the set of complex numbers z such that the real part of $\frac{1}{z}$ is equal to $\frac{1}{6}$. This set forms a curve. Find the area of the region inside the curve.*

We feed the following instruction to the LLM that serves as the generator:

> *Solve the following problem step by step. Please start each step with "Step :" and split sentences in each step with "\n\n". Please finish you response with "So the answer is ...". Problem: Let S be the set of complex numbers z such that the real part of $\frac{1}{z}$ is equal to $\frac{1}{6}$. This set forms a curve. Find the area of the region inside the curve.*

After getting the following solution, we start checking **Step 5** as an example:

> *Solution:*
> *(Step 0) Let $z = x + yi$ be a complex number, where x and y are real numbers.*
> *(Step 1) We are given that the real part of $\frac{1}{z}$ is equal to $\frac{1}{6}$.*
> *(Step 2) Using the fact that the real part of a complex number $a + bi$ is a, we have* $\operatorname{Re}\left(\frac{1}{z}\right) = \operatorname{Re}\left(\frac{1}{x+yi}\right) = \frac{x}{x^2+y^2}$.
> *(Step 3) Setting* $\operatorname{Re}\left(\frac{1}{z}\right) = \frac{1}{6}$, *we get* $\frac{x}{x^2+y^2} = \frac{1}{6}$.
> *(Step 4) Cross-multiplying and rearranging, we get the equation* $6x = x^2 + y^2$.
> *(Step 5) Completing the square, we obtain* $\left(x - \frac{3}{2}\right)^2 + y^2 = \frac{9}{4}$.
> *(Step 6) This is the equation of a circle with center* $\left(\frac{3}{2}, 0\right)$ *and radius* $\frac{3}{2}$.
> *(Step 7) The area of the region inside the circle is* $\frac{9\pi}{4}$. *So the answer is* $\frac{9\pi}{4}$.

Variable definition extraction This optional stage extracts the definitions of variables that appear in the reasoning chain. SelfCheck feeds the definitions to the

regeneration stage to avoid LLM not understanding the meaning of a variable. The input to the LLM is:

The following is the solution to a math question: Question: Let S be the set of complex numbers z such that the real part of $\frac{1}{z}$ is equal to $\frac{1}{6}$. This set forms a curve. Find the area of the region inside the curve.

Solution:
(Step 0) Let $z = x + yi$ be a complex number, where x and y are real numbers.
...
(Step 7) The area of the region inside the circle is $\frac{9\pi}{4}$. So the answer is $\frac{9\pi}{4}$.

Are there variables in the solution? If so, please list the definition of variable in the form of "1. variable is defined as...".

The underlined parts are the type of question, the question itself and the steps in its solution, respectively.

The output from the LLM is:

Yes. There are variables in the solution.
*1. **z is defined as a complex number of the form $x + yi$, where x and y are real numbers.***

The bold part is then saved to form a part of the input in the regeneration stage.

Target extraction To get a brief and clear target of the current step, the input to the LLM is:

The following is a part of the solution to the problem: Let S be the set of complex numbers z such that the real part of $\frac{1}{z}$ is equal to $\frac{1}{6}$. This set forms a curve. Find the area of the region inside the curve.

(Step 0) Let $z = x + yi$ be a complex number, where x and y are real numbers.
...
(Step 5) Completing the square, we obtain $\left(x - \frac{3}{2}\right)^2 + y^2 = \frac{9}{4}$.

What specific action does the step "Completing the square, we obtain $\left(x - \frac{3}{2}\right)^2 + y^2 = \frac{9}{4}$." take? Please give a brief answer using a single sentence and do not copy the steps.

The underlined parts are the question and reasoning steps before the current one, including the current one.

The output of the LLM is:

The step completes the square to rewrite the equation in standard form of a circle.

The whole sentence is saved and forms the most important part of the input in the regeneration stage.

Information Collection To get sentences in the question and previous steps in the solution that are directly related to the current step, the input to the LLM is:

This is a math question: Question: Let S be the set of complex numbers z such that the real part of $\frac{1}{z}$ is equal to $\frac{1}{6}$. This set forms a curve. Find the area of the region inside the curve.

The following is information extracted from the question:
Information 0: Let S be the set of complex numbers z such that the real part of $\frac{1}{z}$ is equal to $\frac{1}{6}$.
Information 1: This set forms a curve.
Information 2: Find the area of the region inside the curve.

The following is the first a few steps in a solution to the problem:
Step 0: Let $z = x + yi$ be a complex number, where x and y are real numbers.
...
Step 4: Cross-multiplying and rearranging, we get the equation $6x = x^2 + y^2$.

Which previous steps or information does the next step "Completing the square, we obtain $\left(x - \frac{3}{2}\right)^2 + y^2 = \frac{9}{4}$." directly follow from?

The underlined information (contents after Information X:) is simply a list of sentences in the question. The underlined steps are reasoning steps before (not including) the current one in the solution.

The output from the LLM is:

The next step "Completing the square, we obtain $\left(x - \frac{3}{2}\right)^2 + y^2 = \frac{9}{4}$." directly follows from ***Step 4****.*

Then we use regular expression to extract the information and step id from LLM's output. Some steps may only rely on information from the question while others may only use results from previous steps.

Step regeneration In this stage, we instruct the LLM to achieve the extracted target with only collected information with the following prompt:

We are in a process of solving a math problem.

Variables are defined as: 1. z is defined as a complex number of the form $x + yi$, where x and y are real numbers.

The following are some previous steps: Step 0: Cross-multiplying and rearranging, we get the equation $6x = x^2 + y^2$.

The target for next step is: The step completes the square to rewrite the equation in standard form of a circle.

Please try to achieve the target with the information from previous steps.

The underlined parts are the type of question, variable definition (if any), collected information from the question (if any), collected previous steps (if any) and the extracted target. Please notice that the current step does not directly rely on any information from the question, we removed the corresponding block from the input.

The output from the LLM is:

To rewrite the equation in standard form of a circle, we need to complete the square for both x and y.
Starting from the given equation: $6x = x^2 + y^2$.
We add and subtract $(6/2)^2 = 9$ on the right-hand side:
$6x + 9 - 9 = x^2 + y^2 + 9 - 9$.
Simplifying: $(x - 3)^2 + y^2 = 9$. This is now in the standard form of a circle with center $(3, 0)$ and radius 3.

Then we can directly compare the regenerated step with the original one in the next stage.

Result comparison The input to the LLM is:

The following are 2 solutions to a math problem:

Solution 1: To rewrite the equation in standard form of a circle, we need to complete the square for both x and y.
Starting from the given equation: $6x = x^2 + y^2$.
We add and subtract $(6/2)^2 = 9$ on the right-hand side: $6x + 9 - 9 = x^2 + y^2 + 9 - 9$.
Simplifying: $(x - 3)^2 + y^2 = 9$. This is now in the standard form of a circle with center $(3, 0)$ and radius 3.

Solution 2: Completing the square, we obtain $\left(x-\frac{3}{2}\right)^2+y^2=\frac{9}{4}$.

Compare the key points from both solutions step by step and then check whether Solution 1 "supports", "contradicts" or "is not directly related to" the conclusion in Solution 2. Pay special attention to difference in numbers.

The underlined parts are simply the problem type, the regenerated step and the original step.

The output is:

In both solutions, the goal is to rewrite the given equation in standard form of a circle. However, in Solution 1, the completion of the square results in the center of the circle being $(3,0)$ *and a radius of* 3*. In Solution 2, the completion of the square results in the center of the circle being* $(3/2,0)$ *and a radius of* $3/2$*.*

Therefore, Solution 1 ***contradicts*** *the conclusion in Solution 2 as the centers and radii of the circles are different.*

If the word 'support' or 'contradict' appears in the last line of the output, SelfCheck outputs a step score 1 or -1. Otherwise, SelfCheck outputs 0 meaning that two steps are not related.

5.B Examplar for one-shot error checking

The following example from Ling et al. [2023] is used to perform one-shot error checking.

Here is some information:
"1. Alice gets 2 apples.
2. Alice gets twice as many oranges as bananas.
3. Original price per apple: $2
Discount: 50%
*Price per apple after discount: $2 * (1 - 0.50) = $2 * 0.50 = $1"*

Based on the given information, here is a reasoning process:
"Calculate Alice's cost of the 5 apples after a 50% discount.
Price per apple after discount: $1
Apples be eaten: 3
Oranges be eaten: 6
Total apples: 5
Total oranges: x
Total bananas: 2x

Alice's total number of fruits: 5 + x + 2x = 5 + 4x
*Alice's total cost of 5 apples: $1 * (5 - 3) = $2"*

Double-check the reasoning process, let's analyze its correctness, and end with "yes" or "no".

Answer:
Let's think step by step.

Grounding check
Purpose: Find references for numbers in the reasoning process sequentially
Rule: Any ungrounded number makes the check fails

"x" is a variable, we do not care about variables.
"y" is a variable, we do not care about variables.

"5" is a quantity from the given information that represents the number of apples.
"50%" is a quantity from the given information that represents the discount.
"$1" is a quantity from the given information that represents the price per apple after the discount.
"3" is a quantity not from the given information, it is ungrounded.
"6" is a quantity not from the given information, it is ungrounded.
"$2" is a quantity from the given information that represents the original price per apple.
Result: Based on the check, there are ungrounded numbers, so the grounding check fails.

Reasoning check:
Purpose: Check whether the reasoning logic correctly answers the question
Rule: Any logic error makes the check fails

To calculate the total cost of apples after a 50% discount, we should multiply the number of apples by the discounted price. But to answer the question, we do not need to calculate the total number of fruit Alice gets.
Result: Based on the check, there are some logic mistakes, so the reasoning check fails.

Calculation check:
Purpose: Check whether the calculation process is consistent
Rule: Any inconsistent calculation makes the check fails
calculation1:

*equation: $1 * (5 - 3), answer: $2*
(5 - 3) = 2
*$1 * 2 = $2 is consistent with the answer, so the calculation is correct.*
calculation2:
equation: 5 + x + 2x, answer: 5 + 4x
x + 2x = 3x
5 + 3x is inconsistent with the answer, so the calculation is incorrect.

Result: Based on the check, the calculation process is inconsistent, so the calculation check fails.

Check results: Ground check fails, Reasoning check fails, Calculation check fails.
Rule: Any failed check makes the reasoning incorrect.
So the answer is "no".

5.C More examples on target extraction

The following are some real examples of extracted targets on MathQA with GPT-3.5.

***Step**: According to the problem, the profit was divided in the ratio of 2:3, which means that x received 2/5 of the profit and y received 3/5 of the profit.*
***Extracted target**: It calculates the ratio of the profit earned between x and y based on the information given in the problem.*

***Step**: Since the interest has increased by 4%, the new rate of interest (r2) is 16.67% + 4% = 20.67%.*
***Extracted target**: The step calculates the new rate of interest after increasing it by 4%.*

***Step**: The sum of the ages of the first 14 students is 56 + 160 = 216 years.*
***Extracted target**: The step calculates the sum of the ages of the first 14 students of the class.*

***Step**: We can rearrange 8a = 9b to get a/b = 9/8.*
***Extracted target**: The step rearranges the given equation to express a in terms of b and then obtains the ratio of a to b.*

***Step**: So, the equation becomes: 33 + (1+y/100)*50 = 83.*
***Extracted target**: The step sets up the equation to calculate the percentage increase in Shyam's weight.*

***Step**: Find the ratio of investment of each person in the business*
Investment of A : Investment of B : Investment of C
= 6500 : 1300 : 7800
= 5 : 1 : 12
***Extracted target**: The step finds the ratio of investment of each person in the business.*

5.D An experiment on logic reasoning task

In this section, we verify SelfCheck's generalization ability to the task of logic reasoning using the dataset of SNR.[2] Each test sample of SNR consists of a logic question, and its target answer. Different from other datasets shown in the paper, the target answers of SNR samples are natural-language sentences (such as "The hedgehog is cold and scary."), instead of multiple-choice options (for MathQA) or single math expressions (for GSM8K and MATH). Because of the variability of natural languages, exact matching between the generated and target answers or among generated answers becomes infeasbile. Majority voting thus cannot be used for this problem, but we can still use SelfCheck to pick out the best solution from all candidates.

To test performance, we randomly select 100 test samples from the SNR test set and generate 2 answers for each of the questions. Because it is not possible to perform automatic comparison between the generated answers and the target answers, we invited volunteers to perform human evaluation. To better deal with logic problems, we replaced all 'math' with 'logic' in the prompts and deleted the instruction 'pay special attention to the difference in numbers' in the result comparison stage. The prompts used were otherwise identical to those outlined in Section 5.2.

From the results shown in Table 5.D.1, we can see that SelfCheck increases the accuracy on SNR by 3.5%, which is a statistically significant improvement at the 5% threshold based on a t-test.

Dataset	Generator	Checker	Base Acc (%)	SelfCheck Acc (%)	ΔAcc (%)
SNR	GPT-3.5	GPT-3.5	44.5	48.0	3.5±2.0

Table 5.D.1: SelfCheck increases predictive accuracy on SNR by picking out correct answers two independent generations. Base acc means single solution accuracy here, since majority voting is not feasible for SNR.

5.E An experiment on Llama2

To test SelfCheck's generalization ability to other LLMs, we also tested it with Llama2 (70B, 4-bit, Touvron et al. [2023]) on GSM8K (the generative accuracy of Llama2 on the other datasets was too low to perform any meaningful checking). Restricted by

[2] https://huggingface.co/datasets/lighteval/synthetic_reasoning_natural/blob/main/hard/test.jsonl

the speed of Llama2 (70B, 4-bit) on our server (which is only about 20 tokens/s), we randomly select 500 test samples and generate 3 solutions for each question.

From Table 5.E.1, we see a statistically significant improvement in accuracy compared with majority voting using this different LLM; the gains here are actually larger than those observed for GPT-3.5.

Dataset	Generator	Checker	MV Acc (%)	SelfCheck Acc (%)	ΔAcc (%)
GSM8K	Llama2	Llama2	40.3	43.2	2.9±0.3

Table 5.E.1: SelfCheck significantly increases predictive accuracy on GSM8K with Llama2.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	SelfCheck: Using LLMs to Zero-Shot Check Their Own Step-by-Step Reasoning
Publication Status	☑Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Ning Miao, Yee Whye Teh and Tom Rainforth. SelfCheck: Using LLMs to Zero-Shot Check Their Own Step-by-Step Reasoning. International Conference on Learning Representations (ICLR), 2024

Student Confirmation

Student Name:	NING MIAO		
Contribution to the Paper	Ning is the lead author of this paper. He proposed the ideas, wrote all the codes, did all the experiments and wrote the manuscript.		
Signature	Ning Miao	Date	16/06/2024

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Tom Rainforth, Senior Researcher in Machine Learning		
Supervisor comments Ning led the paper, making a substantial contribution as per the description above.		
Signature	Date	20/06/2024

This completed form should be included in the thesis, at the end of the relevant chapter.

Chapter 6

Conclusions, limitations and Future Outlook

In this work, we explained what inductive biases are, why they are important in machine learning, and how to incorporate them in different parts of machine learning algorithms.

- In Chapter 1, we defined inductive biases as the preferences of a machine learning algorithm given a training set. We showed that inductive biases are critically important because they decide how a model generalizes outside of training data.
- In Chapter 2, we reviewed and discussed the inductive biases inherent in different design choices of model structures, training algorithms, and inference algorithms. We analyzed the inductive biases of mainstream neural networks, generative models, optimizers, regularizers, and inference methods. We also used examples to show previous endeavors in incorporating inductive biases, such as invariance and equivariance into a machine learning model. We also discussed the connections of inductive biases with several closely related topics in machine learning, including Bayesian machine learning, meta-learning, and transfer learning.
- Starting from Chapter 3, we introduced three novel methods to incorporate different inductive biases into machine learning models. In Chapter 3, we proposed InteL-VAEs to instill global inductive biases on target distributions into VAEs. We showed that a simple semi-learnable layer between the encoders and the decoders of VAEs would be enough to instill various distributional knowledge, including connectivity, sparsity, and other topological information, into VAEs. Our experiments

demonstrate that the instilled inductive biases can greatly boost the generalization performance and feature quality of VAEs, showing the importance of inductive biases in generative models.

- In Chapter 4, we introduced InstaAug to instill local invariances into models by learning instance-specific data augmentations. Namely, we loosen the overly strong inductive bias of global invariances in traditional data augmentation methods to make the model more flexible. The strong performance of InstaAug on supervised learning and contrastive learning shows that local invariances fit the real situation of the physical world better than global invariances. Meanwhile, InstaAug provides a way to transform the restriction inductive biases in global augmentations to preference inductive biases in instance-specific data augmentations.

- In Chapter 5, we focused on the inductive biases in the inference stage of LLMs. It is the first work to show that by properly prompting LLMs to incorporate suitable inductive biases for checking, we can empower LLMs to check for their own reasoning chains. This work provides a new path to reduce hallucination and improve the reliability of LLM reasoning that does not rely on any labeled data or human feedback.

To conclude, inductive bias is a very useful perspective for analyzing existing machine learning algorithms as well as designing new ones. However, there are some limitations of the proposed methods in incorporating inductive biases, which require further research.

For example, we focused on translating human knowledge into the inductive biases of ML algorithms, but as described in Section 2.5, inductive biases can also be learned from data. The problem with current inductive bias learning methods, such as meta-learning and transfer learning is that their learned inductive biases are difficult to interpret or manipulate, because of their black-box nature. Assume we are transferring the knowledge from a pre-training dataset to a related domain, we need to transfer every piece of learned knowledge. While some of the knowledge might be helpful, some others might not apply to the new domain. Hence, it would be helpful if we could learn interpretable and manipulatable inductive biases from data. For example, we could represent critical inductive biases in natural language, similar to the way we human beings pass on knowledge to peers.

Meanwhile, we still do not know what the most suitable inductive biases are for different tasks and how to incorporate them into ML models, especially LLMs. In the past few years, lots of LLM research has been done on manually designing prompts to bias LLMs to expected directions. Though great progress has been made on various tasks, LLMs are still far from perfect on complicated problems, such as math reasoning. As a result, we need to keep exploring more suitable inductive biases, as well as more efficient ways to incorporate them into ML algorithms.

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, Dandelion 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 `https://www.tensorflow.org/`. Software available from tensorflow.org. 53

Subutai Ahmad and Luiz Scheinkman. How can we be so dense? the benefits of using highly sparse representations. *arXiv preprint arXiv:1903.11257*, 2019. 27

Alexander Alemi, Ben Poole, Ian Fischer, Joshua Dillon, Rif A Saurous, and Kevin Murphy. Fixing a broken elbo. In *International Conference on Machine Learning*, pages 159–168. PMLR, 2018. 28

Alaaeldin Ali, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. Xcit: Cross-covariance image transformers. *Advances in neural information processing systems*, 34:20014–20027, 2021. 69

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. Falcon-40B: an open large language model with state-of-the-art performance. 2023. 89

Shunichi Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, (3):299–307, 1967. 1

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1245. URL https://aclanthology.org/N19-1245. 98

Abdul Fatir Ansari and Harold Soh. Hyperprior induced unsupervised disentanglement of latent representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3175–3182, 2019. 27

Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. *Advances in neural information processing systems*, 32, 2019. 55, 72

Gregor Bachmann, Sotiris Anagnostidis, and Thomas Hofmann. Scaling mlps: A tale of inductive bias. *Advances in Neural Information Processing Systems*, 36, 2024. 12

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 50

Gabriel Barello, Adam S. Charles, and Jonathan W. Pillow. Sparse-coding variational auto-encoders. Preprint, Neuroscience, August 2018. 33, 39

Matthias Bauer and Andriy Mnih. Resampled priors for variational autoencoders. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 66–75. PMLR, 2019. 24, 27, 33

Mohamed Ishmael Belghazi, Sai Rajeswar, Olivier Mastropietro, Negar Rostamzadeh, Jovana Mitrovic, Aaron Courville, and AI Element. Hierarchical adversarially learned inference. *stat*, 1050:4, 2018. 45

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013. 27

Gregory Benton, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. Learning invariances in neural networks. *Advances in Neural Information Processing Systems*, 2020. 14, 18, 56, 57, 62, 63, 65, 79

Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018. 37

Simone Bianco, Claudio Cusano, Paolo Napoletano, and Raimondo Schettini. Improving cnn-based texture classification by color balancing. *Journal of Imaging*, 3(3):33, 2017. 70

Arwen V Bradley, Carlos A Gomez-Uribe, and Manish Reddy Vuyyuru. Shift-curvature, sgd, and generalization. *Machine Learning: Science and Technology*, 3(4):045002, 2022. 15

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 40

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 3, 20, 89, 90

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023. 89

Yuri Burda, Roger B Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *ICLR (Poster)*, 2016. 24, 31

Francesco Paolo Casale, Adrian V Dalca, Luca Saglietti, Jennifer Listgarten, and Nicoló Fusi. Gaussian process prior variational autoencoders. In *NeurIPS*, 2018. 24, 33

Anthony L Caterini, Robert Cornish, Dino Sejdinovic, and Arnaud Doucet. Variational inference with continuously-indexed normalizing flows. 2020. 28

Anadi Chaman and Ivan Dokmanic. Truly shift-invariant convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3773–3783, 2021. 63

Shuxiao Chen, Edgar Dobriban, and Jane H. Lee. Invariance reduces variance: Understanding data augmentation in deep learning and beyond. *CoRR*, abs/1907.10905, 2019. URL http://arxiv.org/abs/1907.10905. 55

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020. 55, 72, 80

Xiangning Chen, Cihang Xie, Mingxing Tan, Li Zhang, Cho-Jui Hsieh, and Boqing Gong. Robust and accurate object detection via adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16622–16631, 2021. 64

Yu-hsin Chen, Ignacio Lopez-Moreno, Tara N Sainath, Mirkó Visontai, Raziel Alvarez, and Carolina Parada. Locally-connected and convolutional neural networks for small footprint speaker recognition. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015. 11

Tsz-Him Cheung and Dit-Yan Yeung. Adaaug: Learning class- and instance-adaptive data augmentation policies. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=rWXfFogxRJN. 56, 63

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 1

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. 20, 89

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 90, 98

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*. PMLR, 2016. 14, 63

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *Conference on Computer Vision and Pattern Recognition*, 2018. 18, 56, 57, 63, 69

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020. 18, 55, 57, 63

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. *arXiv:1804.00891 [cs, stat]*, September 2018a. URL http://arxiv.org/abs/1804.00891. 26, 33

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. *34th Conference on Uncertainty in Artificial Intelligence (UAI-18)*, 2018b. 24

Alfredo De la Fuente and Robert Aduviri. Replication/machine learning. 2019. 52

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. Ieee, 2009. 12, 22, 68

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012. 8

Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. In *International conference on machine learning*, pages 1889–1898. PMLR, 2016. 14

Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016. 24, 33, 37

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv:1605.08803 [cs, stat]*, February 2017. URL http://arxiv.org/abs/1605.08803. 33

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020. 11, 12

Harris Drucker and Yann Le Cun. Improving generalization performance using double backpropagation. *IEEE transactions on neural networks*, 3(6):991–997, 1992. 16

Jeffrey R Edwards and Mark E Parry. On the use of polynomial regression equations as an alternative to difference scores in organizational research. *Academy of Management journal*, 36(6):1577–1613, 1993. 9

Aleksandr Ermolov, Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening for self-supervised representation learning. In *International Conference on Machine Learning*, pages 3015–3024. PMLR, 2021. 72, 82

Luca Falorsi, Pim de Haan, Tim R. Davidson, Nicola De Cao, Maurice Weiler, Patrick Forré, and Taco S. Cohen. Explorations in homeomorphic variational auto-encoding. *arXiv:1807.04689 [cs, stat]*, July 2018. URL http://arxiv.org/abs/1807.04689. 24, 28, 33

Adam Foster, Rattana Pukdee, and Tom Rainforth. Improving transformation invariance in contrastive representation learning. In *International Conference on Learning Representations*, 2021. 55, 72

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. 1

Francis Galton. Regression towards mediocrity in hereditary stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 15:246–263, 1886. 9

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011. 53

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 55

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 4, 12

Google. Palm 2 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 89

Soorya Gopalakrishnan, Zhinus Marzi, Upamanyu Madhow, and Ramtin Pedarsani. Combating adversarial attacks using sparse representations. *arXiv preprint arXiv:1803.03880*, 2018. 27

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. *arXiv:1810.01367 [cs, stat]*, October 2018. URL http://arxiv.org/abs/1810.01367. 33

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*, 2016. 24

Ved Prakash Gupta and Mukund Madhav Mishra. On the topology of certain matrix groups. *THE MATHEMATICS STUDENT*, page 61, 2018. 36

Wes Gurnee. Inductive biases of sgd training. 2022. 15

Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. In *European Conference on Computer Vision*, pages 1–16. Springer, 2020. 18, 57, 63

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a. 12, 68, 69

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b. 67

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2020. 23, 98

David P Helmbold and Philip M Long. On the inductive bias of dropout. *The Journal of Machine Learning Research*, 16(1):3403–3454, 2015. 17

David P Helmbold and Philip M Long. Fundamental differences between dropout and weight decay in deep networks. *arXiv preprint arXiv:1602.04484*, 2016. 17

David P Helmbold and Philip M Long. Surprising properties of dropout in deep networks. *Journal of Machine Learning Research*, 18(200):1–28, 2018. 17

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. 89, 98

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6629–6640, 2017. 37

Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018. 27

Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 599–619. Springer, 2012. 16

Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In *International Conference on Machine Learning*, pages 2731–2741. PMLR, 2019. 18, 57, 63

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 12

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 1, 32

Matthew D Hoffman and Matthew J Johnson. Elbo surgery: yet another way to carve up the variational evidence lower bound. 2016. 27

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 9

Xianxu Hou, Linlin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141. IEEE, 2017. 53

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. page 10, 2018. 33

Ke Huang and Selin Aviyente. Sparse representation for signal classification. *Advances in neural information processing systems*, 19:609–616, 2006. 38

Niall Hurley and Scott Rickard. Comparing measures of sparsity. *IEEE Transactions on Information Theory*, 55:4723–4741, 2009. 39, 51

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015. 14, 64

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. Towards mitigating llm hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843, 2023. 20

Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T Kwok. Backward reasoning in large language models for verification. *arXiv preprint arXiv:2308.07758*, 2023. 90

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *IJCAI*, 2017. 27

Matthew J. Johnson, David Duvenaud, Alexander B. Wiltschko, Sandeep R. Datta, and Ryan P. Adams. Composing graphical models with neural networks for structured representations and fast inference. *arXiv:1603.06277 [stat]*, July 2017. URL http://arxiv.org/abs/1603.06277. 36

Olav Kallenberg and Olav Kallenberg. *Foundations of modern probability*, volume 2. Springer, 1997. 58

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017. 22

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019. 23

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2649–2658. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/kim18b.html. 27

Ildoo Kim, Younghoon Kim, and Sungwoong Kim. Learning loss for test-time augmentation. *Advances in Neural Information Processing Systems*, 33:4163–4174, 2020a. 64

Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *International Conference on Machine Learning*, pages 5275–5285. PMLR, 2020b. 65, 66

JangHyun Kim, Wonho Choo, Hosan Jeong, and Hyun Oh Song. Co-mixup: Saliency guided joint mixup with supermodular diversity. In *International Conference on Learning Representations*, 2020c. 65, 66

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. 2, 15, 82

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014. 24, 30, 61

Alexej Klushyn, Nutan Chen, Richard Kurle, Botond Cseke, and Patrick van der Smagt. Learning hierarchical priors in vaes. 47

Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015. 22

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 491–507. Springer, 2020. 22

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012. 55

Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. In *International Conference on Learning Representations*, 2018. 45

Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik. Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation. *Medical Imaging with Deep Learning*, 4:2, 2018. 22

Jouko Lampinen and Aki Vehtari. Bayesian approach for neural networks—review and case studies. *Neural networks*, 14(3):257–274, 2001. 21

Itay Lavie, Guy Gur-Ari, and Zohar Ringel. Towards understanding inductive bias in transformers: A view from infinity. *arXiv preprint arXiv:2402.05173*, 2024. 11

Tuan Anh Le, Maximilian Igl, Tom Rainforth, Tom Jin, and Frank Wood. Auto-encoding sequential monte carlo. In *International Conference on Learning Representations*, 2018. 31

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1, 32

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336*, 2022. 90, 98

Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M Robertson, and Yongxin Yang. Dada: differentiable automatic data augmentation. *arXiv preprint arXiv:2003.03780*, 2020. 18, 57, 63

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. 98, 99

Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. *Advances in Neural Information Processing Systems*, 2019. 18, 56, 57, 63, 69

Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=8s8K2UZGTZ. 20

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. *arXiv preprint arXiv:2306.03872*, 2023. 90, 91, 97, 98, 99, 100, 102, 103, 104, 110

Seppo Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki, 1970. 1

Bing Liu, Liang-Ping Ku, and Wynne Hsu. Discovering interesting holes in data. In *Proceedings of the Fifteenth international joint conference on Artifical intelligence-Volume 2*, pages 930–935, 1997. 34

Liyuan Liu, Jialu Liu, and Jiawei Han. Multi-head or single-head? an empirical comparison for transformer training. *arXiv preprint arXiv:2106.09650*, 2021. 12

Yang Liu, Yao Zhang, Yixin Wang, Feng Hou, Jin Yuan, Jiang Tian, Yang Zhang, Zhongchao Shi, Jianping Fan, and Zhiqiang He. A survey of visual transformers. *IEEE Transactions on Neural Networks and Learning Systems*, 2023. 11

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024. 10

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 40, 46

Canjie Luo, Yuanzhi Zhu, Lianwen Jin, and Yongpan Wang. Learn to augment: Joint data augmentation and network optimization for text recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13746–13755, 2020. 64

Clare Lyle, Mark van der Wilk, Marta Kwiatkowska, Yarin Gal, and Benjamin Bloem-Reddy. On the benefits of invariance in neural networks. *arXiv preprint arXiv:2005.00178*, 2020. 55

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*, 2023. 90, 97, 99

David JC MacKay. Probable networks and plausible predictions-a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469, 1995. 21

Chris J Maddison, John Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Whye Teh. Filtering variational objectives. In *NIPS*, 2017. 31

Diego Marcos, Michele Volpi, Nikos Komodakis, and Devis Tuia. Rotation equivariant vector field networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017. 63

K. V. Mardia and Peter E. Jupp. *Directional Statistics*. Wiley Series in Probability and Statistics. J. Wiley, Chichester ; New York, 2000. ISBN 978-0-471-95333-3. 34

Emile Mathieu, Charline Le Lan, Chris J. Maddison, Ryota Tomioka, and Yee Whye Teh. Continuous hierarchical representations with poincar\'e variational auto-encoders. January 2019a. URL https://arxiv.org/abs/1901.06033v3. 33

Emile Mathieu, Tom Rainforth, N Siddharth, and Yee Whye Teh. Disentangling disentanglement in variational autoencoders. In *International Conference on Machine Learning*, pages 4402–4412. PMLR, 2019b. 24, 25, 26, 28, 32, 33, 36, 39, 45, 51, 53

Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. Cgmh: Constrained sentence generation by metropolis-hastings sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6834–6842, 2019. 19

Ning Miao, Emile Mathieu, N Siddharth, Yee Whye Teh, and Tom Rainforth. On incorporating inductive biases into vaes. In *International Conference on Learning Representations*, 2021. 5

Ning Miao, Tom Rainforth, Emile Mathieu, Yann Dubois, Yee Whye Teh, Adam Foster, and Hyunjik Kim. Learning instance–specific augmentations by capturing local invariances. In *Proceedings of the 40th International Conference on Machine Learning*, 2023a. 5

Ning Miao, Yee Whye Teh, and Tom Rainforth. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. In *The Twelfth International Conference on Learning Representations*, 2023b. 5

Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE, 2018. 55

Christian Naesseth, Scott Linderman, Rajesh Ranganath, and David Blei. Variational sequential monte carlo. In *International Conference on Artificial Intelligence and Statistics*, pages 968–977. PMLR, 2018. 31

Yoshihiro Nagano, Shoichiro Yamaguchi, Yasuhiro Fujita, and Masanori Koyama. A wrapped normal distribution on hyperbolic space for gradient-based learning. *arXiv:1902.02992 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1902.02992. 24, 33

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015. 15

Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011. 38

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 89

Ivan Ovinnikov. Poincar\'e wasserstein autoencoder. January 2019. URL https://arxiv.org/abs/1901.01427v2. 33

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *arXiv:1705.07057 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1705.07057. 33

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019. 33

Joonhyung Park, June Yong Yang, Jinwoo Shin, Sung Ju Hwang, and Eunho Yang. Saliency grafting: Innocuous attribution-guided mixup with calibrated label mixing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7957–7965, 2022. 65, 66

Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2 (11):559–572, 1901. 9

Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, et al. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint arXiv:2302.12813*, 2023. 90, 97

Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017. 55

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International conference on machine learning*, pages 5301–5310. PMLR, 2019. 15

Alexandre Ramé, Rémy Sun, and Matthieu Cord. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 823–833, 2021. 65, 66, 82

Rajesh Ranganath, Dustin Tran, and David Blei. Hierarchical variational models. In *International Conference on Machine Learning*, pages 324–333. PMLR, 2016. 47

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International conference on learning representations*, 2016. 22

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *NIPS*, 2019. 24, 33

Luis A. Pérez Rey, Vlado Menkovski, and Jacobus W. Portegies. Diffusion variational autoencoders. *arXiv:1901.08991 [cs, stat]*, March 2019. URL http://arxiv.org/abs/1901.08991. 33

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015. 24, 30, 33

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 1

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika*, 71:599–607, 1986. 17

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. PMLR, 2016. 22

Igal Sason. On data-processing and majorization inequalities for f-divergences with applications. *Entropy*, 21(10):1022, October 2019. ISSN 1099-4300. doi: 10.3390/e21101022. 43

Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3839–3848, 2018. 28

Pola Schwöbel, Martin Jørgensen, Sebastian W Ober, and Mark Van Der Wilk. Last layer marginal likelihood for invariance learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3542–3555. PMLR, 2022. 64

Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1009. URL https://aclanthology.org/P16-1009. 18

Divya Shanmugam, Davis Blalock, Guha Balakrishnan, and John Guttag. Better aggregation in test-time augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1214–1223, 2021. 57

Zhiqiang Shen, Zechun Liu, Zhuang Liu, Marios Savvides, Trevor Darrell, and Eric Xing. Un-mix: Rethinking image mixtures for unsupervised visual representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2216–2224, 2022. 73

Wenxian Shi, Hao Zhou, Ning Miao, and Lei Li. Dispersed exponential family mixture vaes for interpretable text generation. In *International Conference on Machine Learning*, pages 8840–8851. PMLR, 2020. 24, 28, 33

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019. 55

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013. 65

Ondrej Skopek, Octavian-Eugen Ganea, and Gary Bécigneul. Mixed-curvature variational autoencoders. November 2019. URL https://arxiv.org/abs/1911.08411v2. 24, 33

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *NIPS*, 2016. 27, 45, 46, 47

Tiecheng Song and Hongliang Li. Wavelbp based hierarchical features for image classification. *Pattern Recognition Letters*, 34(12):1323–1328, 2013. 27

Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. Scale-equivariant steerable networks. In *International Conference on Learning Representations*, 2019. 63

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 16

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017. 22

Alex Tamkin, Mike Wu, and Noah Goodman. Viewmaker networks: Learning views for unsupervised representation learning. In *International Conference on Learning Representations*, 2020. 64

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. 10

Zhiqiang Tang, Xi Peng, Tingfeng Li, Yizhe Zhu, and Dimitris N Metaxas. Adatransform: Adaptive data transformation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3006, 2019. 63

Keyu Tian, Chen Lin, Ser-Nam Lim, Wanli Ouyang, Puneet K. Dokania, and Philip Torr. A continuous mapping for augmentation design. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=gB4hvxTzQLQ. 64

Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? *Advances in Neural Information Processing Systems*, 33:6827–6839, 2020. 55, 72

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996. 16

Jakub M Tomczak and Max Welling. Vae with a vampprior. In *21st International Conference on Artificial Intelligence and Statistics, AISTATS 2018*, 2018. 24, 26, 33, 37

Francesco Tonolini, Bjørn Sand Jensen, and Roderick Murray-Smith. Variational sparse coding. In *Uncertainty in Artificial Intelligence*, pages 690–700. PMLR, 2020. 24, 27, 33, 39, 52

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 20, 89, 99, 113

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*, 2020. 24, 46, 47, 48

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6309–6318, 2017. 24, 28, 33

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 1, 32

Grace Wahba. *Spline models for observational data*. SIAM, 1990. 9

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2022. 99

Stefan Webb, Adam Golinski, Robert Zinkov, Siddharth Narayanaswamy, Tom Rainforth, Yee Whye Teh, and Frank Wood. Faithful inversion of generative models for effective amortized inference. In *NeurIPS*, 2018. 25

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022. 3, 4, 89

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011. 64

Yixuan Weng, Minjun Zhu, Shizhu He, Kang Liu, and Jun Zhao. Large language models are reasoners with self-verification. *arXiv preprint arXiv:2212.09561*, 2022. 90, 97, 99, 100

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 68

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992. 61

Daniel Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. *Advances in Neural Information Processing Systems*, 2019. 63

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*, 2017. 63

John Wright, Allen Y. Yang, Arvind Ganesh, S. Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, 2009. doi: 10.1109/TPAMI.2008.79. 27

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 39

Jin Xu, Hyunjik Kim, Thomas Rainforth, and Yee Teh. Group equivariant subsampling. *Advances in Neural Information Processing Systems*, 2021. 63

Kenneth Yip and Gerald Jay Sussman. Sparse representations for fast, one-shot learning. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, pages 521–527, 1997. 27

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019. 65, 66

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. 65, 66

Richard Zhang. Making convolutional networks shift-invariant again. In *International conference on machine learning*. PMLR, 2019. 63

Tong Zhang and Bin Yu. Boosting with early stopping: Convergence and consistency. 2005. 17

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Learning hierarchical features from generative models. In *International Conference on Machine Learning*, 2017. 27, 45, 46, 47

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Balancing learning and inference in variational autoencoders. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 5885–5892, 2019. 31

Yu Zheng, Zhi Zhang, Shen Yan, and Mi Zhang. Deep autoaugment. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=St-53J9ZARf. 18, 57, 63

Allan Zhou, Tom Knowles, and Chelsea Finn. Meta-learning symmetries by reparameterization. *arXiv preprint arXiv:2007.02933*, 2020a. 56, 65

Fengwei Zhou, Jiawei Li, Chuanlong Xie, Fei Chen, Lanqing Hong, Rui Sun, and Zhenguo Li. Metaaugment: Sample-aware data augmentation policy learning. *Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021. 63

Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Chu Hong Hoi, et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems*, 33:21285–21296, 2020b. 2, 15

Yanzhao Zhou, Qixiang Ye, Qiang Qiu, and Jianbin Jiao. Oriented response networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 63