Towards More Data Efficient Deep Learning

Peter Noel Hayes

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

January 1, 2024

I, Peter Noel Hayes, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Deep learning is the most predominant learning paradigm in artificial intelligence. The deep neural network models deployed in practise are increasingly data and resource hungry. This thesis introduces several methodological ideas to improve the data efficiency of deep learning algorithms across a diverse range of applications.

The first section deals with supervised deep learning in settings where collecting labelled data is expensive in time and/or cost. We focus on the scenario where multiple weak and relatively cheap sources of supervision are also available. We develop an approach that jointly trains the supervised model and a separate label model to aggregate weak supervision sources and show it outperforms existing weak learning approaches across a benchmark of natural language processing problems.

The second section focuses on unsupervised deep learning; specifically the problem of generative modelling. We study generalisation in variational inference when neural network based amortization is used. We introduce a wake-sleep style training scheme for variational autoencoders that improves generalization performance for a given budget of training data and demonstrate the utility of this approach in image modelling and compression applications.

The third section explores how to improve the efficiency of deep reinforcement learning (RL). We propose a model-based RL framework that learns a low dimensional representation of the environment while avoiding the need to learn a generative model of the environment. We demonstrate gains in efficiency over model-free methods when learning directly from pixels in a control problem.

The final section tackles how to align large pretrained generative models to human preferences. We discuss an alternative approach to reinforcement learning from human feedback based on a maximum likelihood criterion and introduce a simple active learning regime for more efficiently collecting preference data.

Impact Statement

The work presented in this thesis has the potential to have both academic, industrial and societal impact related to the practical application of deep learning methods. In particular where data efficiency is of concern, which is increasingly the case as models grow bigger and more data hungry as we discus in chapter 1.

Collecting labelled data is the crux for many applications of supervised deep learning, which is arguably the most widely applied form of deep learning in industry to date. In fact, there is a dedicated industry around data annotation services concerned with helping companies efficiently collect high quality labels for training and evaluating their AI applications. Chapter 3 improves upon existing methods in weak learning which have had widespread practical and commercial impact in this industry [1]. Of particular societal impact are those problems wherein expensive domain experts such as doctors are required, in applications such as medical image based diagnostics, to provide supervision.

Chapter 4 sheds light on the problem of generalization in unsupervised generative models and has already had some academic impact; being peer reviewed and published at a top tier conference [2] and cited multiple times in independent follow up research. It also has positive industrial potential for improving state-of-the-art image modelling and compression applications where strong generalization for the class of model studied is required.

With the advent of large generative models reaching human capabilities across a range of increasingly diverse tasks, an increasingly pressing academic, industrial and societal problem is that of aligning these models with human

preferences. Both from the perspective of improving the utility of these models and relating to AI safety. The methods presented in chapter 6 present a practically simple approach to making better use of data for preference fine-tuning models, which is currently seen as an important stage in the training pipeline towards this alignment goal. It also touches on important themes of how to automate the evaluation of these models using other models. This work has already had academic impact evidenced by being published at a top tier journal and cited multiple times in independent follow on work [3].

Finally, creating agents that can effectively plan and act in the physical and digital world is a panacea for value creation using AI in industry and society. Deep reinforcement learning is the methodology underpinning most innovations in this direction. Chapter 5, although prototypical in nature and still with practical limitations, contains promising ideas around efficiently learning directly from pixels while filtering out redundant information. We have seen related methods scaled up to provide state of the art results in reinforcement learning [4].

Acknowledgements

I'm grateful to Prof. Ingemar Cox for providing me with the opportunity, for his guidance on the scientific process, many contributions and his patience throughout. I'm also grateful to Prof. David Barber for his generosity in time and ideas throughout this process, without which much of this work would not have been possible. I'm grateful to my collaborators and colleagues; with specials thanks to Mingtian Zhang and Hippolyt Ritter for their numerous and significant contributions. I'm grateful to Alex Botev, Thomas Anthony and Vasileios Lampos for their early guidance. Finally, to the rest of my colleagues - Harshil Shah, Raza Habib, Alex Mansbridge and Thomas Bird - for making this an enjoyable process and for all their help along the way.

Contents

List of Figures

List of Tables

Notation

Throughout this work we will largely follow these notational conventions unless specified otherwise:

x	Scalar
$\boldsymbol{x}$	Vector
$\boldsymbol{X}$	Matrix
x	Scalar random variable
$\mathbf{x}$	Vector random variable
$\mathbf{X}$	Matrix random variable
$\mathcal{X}$	Set, usually of vectors $\mathcal{X} = \{\boldsymbol{x}^1, \cdots, \boldsymbol{x}^N\}$
$\boldsymbol{x}$, $\boldsymbol{z}$	Observed and latent observations respectively
$\boldsymbol{y}$, $\tilde{\boldsymbol{y}}$	Label and weak/noisy label observations respectively
$\boldsymbol{x}^n$	n'th element of a set
$\boldsymbol{x}_i$	i'th dimension of a vector
$\boldsymbol{x}_t$	Vector at step t in an episode
$\check{\boldsymbol{x}}$	Test datapoint
$\mathcal{N}$	Gaussian distribution
$\mathcal{H}_p(\mathbf{x})$	Entropy of distribution $p(\mathbf{x})$
$p(.)$ $q(.)$ $\rho(.)$	Probability density/mass functions
$\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]$	Expectation of $\mathbf{x}$ under distribution p
$\mathcal{L}$	Objective function
KL(.||.)	Kullback-Leibler divergence
$\boldsymbol{\theta}$, $\boldsymbol{\phi}$, $\boldsymbol{\eta}$	Vectors of learnable parameters
$f_{\boldsymbol{\theta}}$	Neural network parameterized by $\boldsymbol{\theta}$
$\nabla_{\boldsymbol{\theta}}$	First order gradient w.r.t $\boldsymbol{\theta}$
$O(\cdot)$	Big-O notation

Chapter 1

Introduction

Deep learning is a foundation of modern day progress in Artificial Intelligence (AI) research [5, 6], the impact of which can now be felt across society. We've gone from seeing success in classical narrow applications - such as image classification [7, 8], speech recognition [9] and basic natural language processing [10, 11] - to experiencing intelligent systems that are redefining industries. From autonomous driving technology [12], to coding co-pilots that significantly increase the productivity of software engineers [13], to personal assistants like ChatGPT that arguably pass the original Turing test [14]. The international community are now cooperating on ensuring the next phase of AI development is inclusive in terms of economic growth and sustainable development, and to foster public trust in AI systems [15].

Fundamentally deep learning is a machine learning methodology that utilizes function approximators consisting of multiple layers of artificial neurons; so-called neural networks models. These models transform raw input data, such as text or image pixels, through a series of linear and non-linear operations defined by the neurons, producing complex hierarchical representations. These representations are used by the model to make predictions. The form the prediction takes is dependent on the specific task; in image classification, the prediction would be the class of the image, for example a cat if learning to categorise animals. The learning process involves adjusting the parameters of the neurons at each layer to satisfy some criteria over a given training dataset.

One of the origins of deep learning research can be traced back to the development of the brain inspired perceptron model by Frank Rosenblatt in the 1950s [16]. This laid the foundation for modern day neural network models that form the basis for deep learning research. However in 1969 Minksy and Papert published their work Perceptrons that highlighted the inability for the simple perceptron model to solve non-linear problems such as XOR [17]. This work contributed to a decrease in interest and funding in neural network research for the subsequent couple of decades. Alternative more traditional statistical methods prospered, like support vector machines [18] and decision trees [19]. It was not until the early 2000s when deep learning really saw a more widespread research resurgence, which has continued to grow in momentum to produce the modern day impact described above. Prior methodological advancements, like stacking multiple perceptrons to create *deep* models [20] and the efficient back-propagation learning algorithm [21], were combined with significantly larger and more efficient computational resources and significantly larger data sets. During the 2010s, this resurgence and scaling-up lead to variations of deep neural network models getting close to or surpassing human level performance on a range of specific narrow problems such as image classification [7, 8], reading comprehension [22, 23] and game playing [24, 25].

Now in recent years the popular trend coined Generative AI by marketeers has emerged; transcending both academia and industry. This is mainly due to the advent of very large language models (LLMs) trained on internet scale data such as GPT-4 [26]. This class of model is capable of generating remarkably human-like text and exhibiting more general capabilities than previous systems. This new capability is linked to a hypothesis in deep learning known as the *scaling laws*, which has supporting empirical evidence [27]. Scaling laws suggest that the performance of models improves as a power-law function of their size, training data and computational resources. This implies that by simply scaling up existing models - some combination of increasing the number of model parameters, training on larger datasets and leveraging more computation

- we can expect to achieve better performance. On one hand, this suggests an exciting path forward: continued growth in model and dataset size and computational resources could lead to further advancements in capabilities. On the other hand, this poses serious challenges: larger models and datasets may become impractical for many important use cases, require vast amounts of energy and raise important questions about the democratization of AI, as only a few entities may have the resources to gather such datasets and train such models [28].

Hence a drawback of deep learning methodology is the typical requirement for increasingly large amounts of data to effectively train and generalize to new unseen data. This requirement seems to be at odds with how humans and animals are understood to learn [29]. For instance, unlike the deep learning image classifiers referenced above, a child does not need to see thousands of images of a cat to recognize one. Equipped with only a few examples, they can effectively generalize and identify a cat in various positions, sizes, and colors. Similarly, a human player does not need tens of thousands, if not millions, of games to begin to effectively understand the rules and devise strategies for board games like chess or go. This apparent limitation in learning efficiency is the core motivation that ties together the work in this thesis.

Our core research question is how do we make these deep learning systems more data efficient such that they can do more with less data? We seek simple and practical adjustments to deep learning techniques to make them more data efficient. To better understand what we mean by data efficiency, we must first consider the different applications of deep learning before discussing further our specific contributions. There are three canonical types of learning methodology where deep neural networks are applied that we touch on in this thesis; supervised learning, unsupervised learning and reinforcement learning. We discuss each informally here before more careful treatment in the next chapter.

Supervised learning (SL) is where deep learning found much of it's early

success. In this paradigm, models are trained to learn a mapping from an input datapoint to a corresponding output or target label, based on a labeled dataset of input/output pairs. A common example being data classification. Our aim in SL is to minimize the discrepancy between the model's predictions and the actual targets. The measure of data efficiency in this setting usually refers to how many labels are required to achieve a specified performance. In many important real world applications of supervised learning the collection of labelled data can be very expensive in time and/or cost and can severely limit adoption. One such example is medical imaging, on tasks like tumour detection, where the collection of labelled data involves the participation of medical experts who's time is sparse and expensive [30, 31, 32].

Unsupervised learning, in contrast, involves training models on datasets without explicit target labels. The learning process is guided by an objective of discovering underlying structures or distributions in the available collection of unlabelled datapoints. This form of learning is particularly useful for tasks such as generative modeling or dimensionality reduction, where the goal is to generate new data that represent the underlying distribution of the input data, or to understand the underlying structure in the data for downstream tasks. The measure of data efficiency in this setting usually refers to how many datapoints are required to achieve a particular performance level. Although there is no need for labels like in supervised learning, there are various practical problems that benefit from unsupervised deep learning and where data is sparse and expensive to collect. For example in chemistry and biology research, deep generative models are used to generate molecular structures for potential new drugs; the models are trained on existing molecular structures that are expensive to produce [33, 34].

Lastly, reinforcement learning (RL) is a type of learning where an agent learns to make decisions by interacting with an environment. Each interaction with the environment is known as an episode. The agent receives feedback in the form of rewards based on the actions taken and our goal is to learn a policy

model that determines which actions to take at a given step in the episode, that maximizes the cumulative reward over time. Deep neural networks are used as the policies and other components of various state of the art RL frameworks. This form of learning naturally applies to problems that involve sequential decision making, such as game playing, robotics, and autonomous driving [35]. The measure of data efficiency in this setting is typically the number of episodes required to reach a target cumulative reward. In certain real world examples each episode is associated to a physical cost. For example in real robotic control, the cost could be the fuel source and wear and tear on the robot itself [36, 37].

How to improve data efficiency is a very multi-faceted problem in deep learning. A plethora of different broad strategies have emerged in the literature towards this goal, many of which are complimentary and are often combined in practise with great effect. The following topics are relevant to the contributions in this thesis and will be discussed in more detail in the relevant chapters: weak learning (chapter 3), transfer learning (chapter 4), active learning (chapter 6) and training data augmentation (chapters 4 and 5).

- **Weak learning** encompasses techniques that leverage noisy or imprecise labels to train models usually in the supervised regime, thereby reducing the dependence on large, clean datasets [38, 39].

- **Transfer learning** involves reusing a model developed for one task as the starting point for a model on subsequent (usually related) tasks in order to speed up learning [40].

- **Active learning** is an iterative process where the model actively queries an oracle (such as a human expert) during training to label new data points that it predicts will be most beneficial for learning, which reduces the amount of data required for training overall [41].

- **Data augmentation** generally involves generating additional training data to improve performance either by applying transformations or noise to the existing data, using a trained generative model to synthesize

additional examples, or by bootstrapping predictions from the model being trained [42].

Other related research areas are meta-learning [43] and semi-supervised learning [44], which lie outside of the scope of this thesis.

In the next section 1.1 we describe how each of the contributions in this thesis map onto these problem areas and summarise the specific contributions that help improve performance around data efficiency.

Throughout this thesis we adopt a basic probabilistic modeling perspective on deep learning. While deep neural network models are in principle complex, highly non-linear functions, they can be formulated as parameterising probability distributions of observed data. This perspective allows us to leverage a basic toolbox of probability theory and statistical inference, providing a common, principled language for developing and communicating our methods.

1.1 Outline and contributions

This section will clarify the structure of this thesis and discuss how it relates to my existing publications. The majority of the work presented in this thesis was born from collaborations, to varying degrees, as are many machine learning papers given their multi-faceted and interdisciplinary nature. Where I am not the first author of the associated publication, I summarise the contribution boundaries below.

This thesis is broadly composed of 4 independent chapters that deal with ideas for improving data efficiency within the learning domains discussed in the previous section. The first part develops a weak learning approach for improving label efficiency for deep supervised learning models, the second improves on generalisation performance in deep unsupervised generative models, the third introduces a deep model based reinforcement learning method, and finally the fourth improves supervised preference fine-tuning of large pre-trained generative models. More specifically the structure of the thesis is the following:

Chapter 1 - Background: Provides details on the necessary background

for the contributions to follow; building upon the concepts presented in this introduction. Further background related to the sub-topics (weak learning, variational methods, reinforcement learning and active learning) is provided in the relevant chapters.

Chapter 2 - Integrated Weak Learning: focuses on the paradigm of weak learning where it's assumed that multiple weak and relatively cheap sources of supervision are available alongside the labelled data in supervised learning. Typically a two-stage process is followed, where a separate label model is constructed in order to denoise and aggregate the weak sources to produce labels that can be added to the training set to improve performance. We develop an approach that instead jointly trains the end model with the label model. This chapter is based on the following paper:

P. Hayes, M. Zhang, R. Habib, J. Burgess, E. Yilmaz, and D. Barber. Integrated Weak Learning. In *arXiv preprint arXiv:2206.09496*. 2022.

Chapter 3 - Generalization Gap in Amortized Inference: deals with the problem of unsupervised deep generative modelling. Specifically we focus on the popular setting where neural networks are used to amortise the cost of inference across datapoints in probabilisitic deep models. We introduce a wake-sleep style training scheme for the variational autoencoder class of model that improves generalization performance for a given budget of training data by augmenting the dataset with predictions from the model.

This chapter is based around joint work in the following paper. My contributions were around ideation on the generalization gap that our collaborator David originally seeded, experiment design and the transfer learning experiments that use the learned representations. My other collaborator and lead author Mingtian formalised the initial ideas, suggested the wake-sleep connection and did the main application to compression.

Mingtian and I wrote the conference paper together with feedback from David.

M. Zhang, P. Hayes, and D. Barber. Generalization Gap in Amortized Inference. In *Advances in Neural Information Processing Systems.* 2022.

Chapter 4 - Solipsistic Reinforcement Learning: explores how to improve the data efficiency of deep reinforcement learning. We introduce a model-based RL framework that learns a low dimensional representation of the environment while avoiding the need to learn a generative model of the environment. This is based on the following paper where I am joint first author:

P. Hayes, M. Zhang, Z. Andi, and D. Barber. Solipsistic Reinforcement Learning. In *International Conference on Learning Representations Workshop on Self-Supervision for Reinforcement Learning.* 2021.

Chapter 5 - Active Preference Learning: is based around a setting of particular practical interest; we build upon methods to align large pre-trained deep generative models to human preferences. We introduce a simple active learning regime for more efficiently collecting preference data. This work is based on the following paper and was carried out mainly in collaboration with William Muldrew at UCL. I seeded the original idea, implemented the first version, collaborated on experiment design and wrote the paper with feedback from co-authors William and Mingtian. William implemented the experiments scaling up to larger language models in the conference paper and collaborated on iterating on the methodology and experiment design throughout the project.

W. Muldrew, P. Hayes, M. Zhang, and D. Barber. Active Preference Learning for Large Language Models. In *International Conference on Machine Learning.* 2024.

The following research was also published during my PhD but, although aspects can be related to the broad theme, will not be covered by this thesis:

M. Morris, P. Hayes, I. J. Cox, and V. Lampos. Neural network models for influenza forecasting with associated uncertainty using Web search activity trends. In *PLoS Computational Biology*. 2023.

M. Zhang, P. Hayes, T. Bird, R. Habib, and D. Barber. Spread Divergence. In *International Conference on Machine Learning*. 2020.

E. Yilmaz, P. Hayes, R. Habib, J. Burgess, and D. Barber. Sample efficient model evaluation. In *arXiv preprint arXiv:2109.12043*. 2021.

M. Zhang, O. Key, P. Hayes, D. Barber, B. Paige, and F.-X. Briol. Towards Healing the Blindness of Score Matching. In *Advances in Neural Information Processing Systems Workshop on Score-Based Methods*. 2022.

Chapter 2

Background

This chapter builds upon the concepts introduced in the last chapter to detail the necessary terminology, notation and methodology on probabilistic models and neural networks and their relevant training protocols.

Given this thesis is composed of multiple pieces of relatively independent work, the relevant recent work and topic specific background related to active learning, model based RL and fine-tuning generative models, will be covered in the relevant subsequent chapters. The background in this section are the prior building blocks of notation and modelling concepts required before covering these specific topics further.

The contents to follow assume a general background in multivariate calculus, linear algebra and probability theory. For a more thorough resource on the topics covered here, we recommend reading [51], [52] and [53].

2.1 Probabilistic modelling

Given an observed dataset $\mathcal{X}_{train} = \{\boldsymbol{x}^1, \cdots, \boldsymbol{x}^N\}$, where $\boldsymbol{x} \in \mathbb{R}^D$ or $\boldsymbol{x} \in \{0,1\}^D$, sampled from some underlying unknown data distribution $p_d(\mathbf{x})$, the goal of unsupervised learning is to approximate $p_d(\mathbf{x})$ with another distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$. We refer to $p_{\boldsymbol{\theta}}(\mathbf{x})$ as our model with parameters $\boldsymbol{\theta}$ that we aim to determine or *learn* using the observed data. The process of learning in our setting generally entails minimising some measure of distance between $p_{\boldsymbol{\theta}}(\mathbf{x})$ and the unknown $p_d(\mathbf{x})$ by changing $\boldsymbol{\theta}$. A principled method we'll use throughout this thesis is

to minimize the Kullback-Leibler (KL) divergence. That is, we want to find the parameters $\hat{\boldsymbol{\theta}}$ such that

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathrm{KL}(p_d(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) \tag{2.1}$$

where the KL is defined as

$$\mathrm{KL}(p_d(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_d(\mathbf{x})}[\log p_d(\mathbf{x})] - \mathbb{E}_{p_d(\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x})]. \tag{2.2}$$

We use $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$ to denote the expectation of function $f(\mathbf{x})$ with respect to distribution $p(\mathbf{x})$, $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] \equiv \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x}$. The first term represents the negative entropy of the data distribution $-\mathcal{H}_{p_d}(\mathbf{x}) \equiv \mathbb{E}_{p_d(\mathbf{x})}[\log p_d(\mathbf{x})]$, which is a constant (i.e. it does not depend on $\boldsymbol{\theta}$). The second cross entropy term involves integrating over the unknown data distribution $p_d(\mathbf{x})$, which can be approximated using a Monte-Carlo estimate with the observed data $\mathcal{X}_{train}$ that in general we assume to be sampled identically and independently (i.i.d);

$$\mathbb{E}_{p_d(\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x})] \approx \frac{1}{N}\sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^n). \tag{2.3}$$

Estimating $\boldsymbol{\theta}$ by minimizing the KL divergence in this way is equivalent to maximum likelihood estimation (MLE). The likelihood here refers to how likely a set of parameters are given the data. We will cover off how this minimization procedure works in practise later in section 2.3. This optimization process using $\mathcal{X}_{train}$ is referred to as training or learning.

Traditionally in the case of supervised learning, where we instead have labelled data pairs $\mathcal{X}_{train} = \{(\boldsymbol{x}^1, \boldsymbol{y}^1), \cdots, (\boldsymbol{x}^N, \boldsymbol{y}^N)\}$, the underlying data distribution is now the joint distribution $p_d(\mathbf{x}, \mathbf{y}) = p_d(\mathbf{y}|\mathbf{x})p(\mathbf{x})$. We wish to approximate the true conditional distribution $p_d(\mathbf{y}|\mathbf{x})$ with a supervised model $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$. We can similarly use maximum likelihood estimation for fitting our

model using the observed data

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \frac{1}{N} \sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}^n | \boldsymbol{x}^n). \tag{2.4}$$

For specific machine learning problems, there are two design choices typically required to define the likelihood. Firstly, depending on the domain of $\mathbf{x}$ (in the case of unsupervised learning) or $\mathbf{x}$ and $\mathbf{y}$ (in the case of supervised learning), we must choose an appropriate class of probability distribution for the likelihood. Secondly, we require a parameterization $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ for the chosen distribution to define how the parameters might relate to the input data - in our setting this is a deterministic function. To illustrate this, sticking with the supervised setting and the choice of distribution, in binary classification settings where $y^n \in \{0, 1\}$ we can use a Bernoulli likelihood function

$$p_{\boldsymbol{\theta}}(\mathrm{y} = 1 | \boldsymbol{x}) = \sigma(f_{\boldsymbol{\theta}}(\boldsymbol{x})), \tag{2.5}$$

where $\sigma(\boldsymbol{x}) = (1 + e^{\boldsymbol{x}})^{-1}$ is the sigmoid function. In multi-class classification settings, where y^n can now take one of K different classes, we can define $\boldsymbol{y}^n$ as a K dimensional one-hot vector and use a categorical distribution

$$p_{\boldsymbol{\theta}}(\mathbf{y}_i = 1 | \boldsymbol{x}) = \mathrm{softmax}(f_{\boldsymbol{\theta}}(\boldsymbol{x}))_i \tag{2.6}$$

where $\mathrm{softmax}(\boldsymbol{x})_i = e^{\boldsymbol{x}_i} / \sum_{\boldsymbol{x}^{i'}} e^{\boldsymbol{x}_{i'}}$ is the softmax function that maps to the probability simplex. And finally in the case of regression, we can use a Gaussian distribution where the mean of the distribution is the learned function of data

$$p_{\boldsymbol{\theta}}(\mathbf{y} | \boldsymbol{x}) = \mathcal{N}(\mathbf{y} | f_{\boldsymbol{\theta}}(\boldsymbol{x}), \nu). \tag{2.7}$$

The variance ν can also be learnt but is usually considered as part of a more Bayesian treatment where a prior is introduced on the parameters.

In the limit of an infinite amount of data the MLE estimate $\hat{\boldsymbol{\theta}}$ will converge to the true value that recovers p_d, but, crucially, assuming a well specified model

[1]. A model is considered well-specified if there exists a choice of distribution p (e.g exponential family) and parameterization $\boldsymbol{\theta}$ such that $\mathrm{KL}\Big(p_d(\mathbf{x})||p_{\hat{\boldsymbol{\theta}}}(\mathbf{x})\Big) = 0 \Rightarrow p_d(\mathbf{x}) = p_{\hat{\boldsymbol{\theta}}}(\mathbf{x})$. For a given choice of distribution, a significant focus of machine learning is concerned with the choice of a function approximator $f_{\boldsymbol{\theta}}$ and how to update the parameters during training. In principle, the goal is to select a model with enough capacity to capture the underlying complexity of the data. Capacity here refers to the ability of a function approximator to fit a wide variety of functions. If the capacity is too low, say for example using a strictly linear model to model data with non-linear correlations, the model will be unable capture the true data-generating process. This results in what we refer to as under-fitting. This is where neural networks and deep learning come in. The promise of deep learning is to provide a very general purpose and powerful function approximator. With tens of billions of dimensions in $\boldsymbol{\theta}$ in modern state-of-the-art models, they have capacity for learning the most complex of distributions. In section 2.2 we will drill into the specifics of neural network functions and then in 2.3 how optimization works in practise using efficient gradient descent methods.

In practise we only have access to finite noisy samples from $p_d(\mathbf{x})$ in X_{train}. Hence when measuring the performance of a trained model $p_{\hat{\boldsymbol{\theta}}}(\mathbf{x})$, we do so by computing the test likelihood (and other evaluation metrics of interest that are use case specific) on a separate held out (from training) dataset $\mathcal{X}_{test} = \{\check{\boldsymbol{x}}^1, \ldots, \check{\boldsymbol{x}}^M\} \sim p_d(\mathbf{x})$. We refer to this as the test or the generalization performance of the model. If the model capacity is too high and the learning procedure is not robust, the model may end up encoding the noise or spurious correlations in the training samples; as opposed to capturing the true underlying generative process leading to a model that does not generalise well to test data. We refer to this as over-fitting. Therefore the power of neural networks can yield inferior performance versus simpler models for certain use cases where

[1]There are also other technical requirements such as the data is sampled i.i.d., the parameter space is identifiable, the likelihood function is twice differentiable with respect to the parameters, and the Fisher information matrix is positive definite - see [54] for a more thorough treatment.

data is sparse and/or noisy. We will explore the concepts of over-fitting and under-fitting in more detail in chapter 4 in context of unsupervised generative models.

2.1.1 Latent variables

The probabilistic models discussed so far have been specified with fully observed random variables. It is often useful to introduce unobserved latent random variables into the specification of our models to better model the phenomena of interest. Consider for example the problem of modelling the topics that appear in a collection of documents; many techniques, such as Latent Dirichlet Allocation (LDA) [55], assume that an observed document $\boldsymbol{x}^n$ is a mixture of unobserved latent topics $\boldsymbol{z}^n$ that one would like to infer. Another more general example that motivates the use of latent variables is if we assume our observed data actually lies on some lower dimensional latent manifold that we wish to learn.

Latent variable models assume the data is generated by the transformation of one or more underlying latent variables. As with the fully observed case, we can use maximum likelihood estimation; but here we must marginalise out the unobserved variables

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z}. \tag{2.8}$$

With $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$, this integral may not be tractable for certain choices of $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ and attempting to do the integration numerically would mean we no longer have a closed form objective for optimizing $\boldsymbol{\theta}$.

We can get around this intractability by first forming a lower bound on the log-likelihood using Jensen's inequality

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z})] =: \mathcal{L}(\mathbf{x}, q, \boldsymbol{\theta}), \tag{2.9}$$

which introduces $q(\mathbf{z})$, which is referred to as a variational distribution. And by then using the classic Expectation Maximisation (EM) algorithm [56]. Here

we have assumed our latent is continuous; for discrete variables the integral can be replaced with a sum.

The EM procedure optimizes the log likelihood by iteratively increasing the lower bound by alternating between the steps:

E-step: hold $\boldsymbol{\theta}$ fixed and optimize $\mathcal{L}(\mathbf{x}, q, \boldsymbol{\theta})$ w.r.t $q(\mathbf{z})$.

M-step: hold $q(\mathbf{z})$ fixed and optimize $\mathcal{L}(\mathbf{x}, q, \boldsymbol{\theta})$ w.r.t $\boldsymbol{\theta}$.

With some simple manipulation we can show

$$\mathcal{L}(\mathbf{x}, q, \boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}) + \mathbb{E}_{q(\mathbf{z})}[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})}]. \tag{2.10}$$

The second term here is the KL divergence $\mathrm{KL}(q(\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$ that is minimised if and only if the two distributions are the same. Hence the lower bound is equal to the marginal log likelihood when $q(\mathbf{z})$ is equal to the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Therefore at the E-step $\mathcal{L}(\mathbf{x}, q, \boldsymbol{\theta})$ is maximised by setting $q(\mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$.

Within the realm of probabilistic deep learning, neural networks are often used to parameterise $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ to increase the expressiveness of the model. With this choice, the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ cannot be easily computed and therefore directly applying the EM algorithm to optimize the log likelihood is not possible. Approximate inference methods like variational inference techniques have been developed to overcome this source of intractability.

2.1.2 Variational inference

The core idea behind variational inference is to convert a slow or intractable inference problem into a tractable and more scalable optimization problem [57, 58]. The basic approach in this latent variable setting is to parameterise an inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ with parameters $\boldsymbol{\phi}$ and then maximise the same lower bound on the log likelihood as in the EM algorithm, but with respect to both

the generative parameters $\boldsymbol{\theta}$ and inference parameters $\boldsymbol{\phi}$;

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\Big[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\Big] \tag{2.11}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\Big) \tag{2.12}$$

$$=: \mathrm{ELBO}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}). \tag{2.13}$$

This version of the lower bound is commonly referred to as the Evidence Lower Bound (ELBO). The specific choice of the parametric form of $q_{\boldsymbol{\phi}}$ is usually a trade-off between data and computational efficiency, and expressivity. Our focus is around variations where neural networks are utilized. Specifically, in chapter 4 we focus on the popular choice where $q_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})$ is a Gaussian $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}_1}(\mathbf{x}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\phi}_2}(\mathbf{x})))$, with the mean $\boldsymbol{\mu}_{\boldsymbol{\phi}_1}(\mathbf{x})$ and variance $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}_2}(\mathbf{x})$ are neural networks that take $\boldsymbol{x}^n$ as inputs and produce the parameters of the latent distribution [59, 60]. The cost of inference is amortised here because $\boldsymbol{\phi}$ is generally shared across datapoints. The concept of amortization using neural networks is touched on in chapter 3 and more so in chapter 4. Here we have assumed a continuous latent, but it is also possible to support discrete latents using an appropriate choice of distribution, such as the categorical.

In the next section we will first go into detail on the different functional forms neural networks can take, before then outlining how optimization happens in practise.

2.2 Neural networks

In the previous section we introduced probabilistic modelling and discussed at a high level where neural networks come into the fold as powerful function approximators used to parameterise probability distributions. Now we first define the specific types of neural network architectures that are used in practise throughout this thesis. Specifically we cover feed-forward networks, convolutional networks, recurrent networks and transformers. Each of which has inductive biases that make them more effective for certain types of problem.

We will then cover the optimization procedures for training these types of models when used to learn distributions.

2.2.1 Feed-forward networks

Arguably the simplest form of neural network is the feedforward network, or sometimes referred to as the multi-layer perceptron (MLP). This consists of an input layer, one or more hidden layers, and an output layer. Each layer is composed of units or neurons that apply a nonlinear transformation to the weighted sum of their inputs

$$\boldsymbol{h}_0 = \boldsymbol{x} \tag{2.14}$$

$$\boldsymbol{h}_l = a_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1}) \quad \text{for } l = 1, \ldots, L \tag{2.15}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_L\}$ are the parameters to optimize and a_l denotes the non-linear activation function applied element-wise. Activation functions introduce non-linearity into the network, allowing it to learn complex functions. Examples of activation functions a_l include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). The ReLU function $\text{relu}(\boldsymbol{x})_i = \max(\boldsymbol{x}_i, 0)$ is particularly popular in deep learning models due to its computational efficiency and favourable properties when computing gradients during optimization (see section 2.3). The form of the output layer is dependent on the task; different output domains and parameterizations for our $p_{\boldsymbol{\theta}}$ will result in different choices here as touched on in section 2.1. This class of neural network with an appropriate number of neurons and layers has been shown to be a universal function approximator [61]. We use simple and relatively small (in terms of parameter count) feed-forward architectures as baselines in many of the experiments throughout this thesis.

2.2.2 Convolutional networks

Convolutional neural networks (CNNs) incorporate a layer design with an inductive bias that is translation-invariant and which leverages the spatially local features of input data. The hallmark feature of these networks is the

convolution operation that identifies local patterns within the input. Unlike dense or fully connected layers that calculate a weighted sum of all inputs, convolutional layers apply filters across local patches of the input. This model architecture was inspired originally by the organization of the visual cortex and is particularly adept at image recognition where it's beneficial to recognize objects regardless of their position in the visual field [62].

For an input tensor $\boldsymbol{X} \in \mathbb{R}^{C \times H \times W}$, where C is the number of channels (e.g. 3 for RGB images), and H and W are the height and width of the input, and a parameter tensor $\boldsymbol{W} \in \mathbb{R}^{K \times C \times M \times N}$; the convolution operation at layer l for output channel k is computed as follows:

$$\boldsymbol{H}_l^{(i,j,k)} = a_l \left(\sum_{c=0}^{C-1} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \boldsymbol{W}_l^{(k,c,m,n)} \cdot \boldsymbol{X}^{(c,i+m,j+n)} \right), \tag{2.16}$$

where $\boldsymbol{H}_l$ is commonly referred to as the feature map at layer l, (i, j) indexes the spatial position on the feature map and k indexes the output channels. $M \times N$ is the size of the convolutional kernel and K the number of output channels. The weight tensor $\boldsymbol{W}_l$ represents the parameters of the convolutional filters, and a_l is a nonlinear activation function such as the ReLU.

In CNNs, pooling layers are commonly interspersed with convolutional layers to reduce the spatial dimensions of the feature maps. The purpose of pooling is to down-sample the input representation, making the network more computationally efficient and robust to small variations in the location of features within the inputs. The most common form of pooling is max pooling. This takes the maximum value within a local neighborhood of the feature map. For example, with a 2×2 max pooling operation, the input is partitioned into 2×2 blocks, and the maximum value from each block is taken to form the down-sampled output. This operation can be defined as:

$$\boldsymbol{P}_l^{(i,j,k)} = \max_{p \in \mathcal{P}_{i,j}} \boldsymbol{H}_{l-1}^{(p,k)}, \tag{2.17}$$

where $\boldsymbol{P}_l$ is the pooled feature map, $\mathcal{P}_{i,j}$ is the pooling window corresponding

to the spatial position (i, j), and $\boldsymbol{H}_{l-1}$ is the feature map from the previous layer. In chapters 4 and 5 we make use of CNNs for image modelling problems. There are two additional architectural components we exploit that are regularly incorporated into CNNs (and more broadly) that help bolster empirical performance - in particular on larger dimensional and more complex image datasets. They are batch normalization and residual connections.

Batch normalization

Batch normalization is a technique that has been found empirically to help stabilize and accelerate the optimization of many deep neural network architectures by normalizing layer inputs. There is a lack of formal understanding as to when and why batch norm improves performance, but there is at least some consensus in the literature that it aids optimisation by allowing larger learning rates by smoothing the objective function [63]. Applied before the activation function, it adjusts the inputs to have zero mean and unit variance by introducing scale and shift parameters that are also learnable (i.e. added to $\boldsymbol{\theta}$):

$$\boldsymbol{h}_l^{\text{norm}} = \boldsymbol{\gamma} \odot \frac{\boldsymbol{h}_l - \mathbb{E}[\boldsymbol{x}]}{\sqrt{\mathbb{V}[\boldsymbol{x}] + \epsilon}} + \boldsymbol{\beta}. \tag{2.18}$$

$\boldsymbol{h}_l$ are the inputs to batch norm, $\mathbb{E}[\boldsymbol{x}]$ and $\sqrt{\mathbb{V}[\boldsymbol{x}]}$ are the mean and variance of the samples provide in batch to the model (more on mini-batching in section 2.3). ϵ is a small constant for numerical stability and $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are parameters jointly optimized during training along with the parameters of the layers to which batch norm is applied.

Residual connections

Residual connections, also known as skip connections, facilitate the training of very deep architectures. By adding the input of a layer onto its output, they enable the network to learn modifications to the identity mapping rather than complete transformations, in some cases simplifying the learning process:

$$\boldsymbol{h}_l = g_l(\boldsymbol{h}_{l-1}) + \boldsymbol{h}_{l-1}, \tag{2.19}$$

where $\boldsymbol{h}_l$ denotes the output of the l-th layer, and g_l is the residual mapping to be learned by that layer. Intuitively, this approach allows each layer to focus on learning the small, incremental changes that need to be applied to the input data. The path for information flow also ensures that the signal is not diluted as it passes through multiple layers, which is particularly beneficial for maintaining the strength of the signal in deep networks. This type of layer was popularised by the the so called ResNet architecture [8].

2.2.3 Recurrent networks

Recurrent neural networks (RNNs) incorporate a layer design with an inductive bias that is well suited to handle sequential data, such as sequences of images describing a scene, or sequences of words in a sentence for natural language processing. The defining characteristic of a recurrent layer is a hidden state vector that captures information from previous inputs in the sequence. This stateful design enables RNNs to more easily exhibit temporal dynamics and to process sequences of variable length. The hidden state $\boldsymbol{h}_t$ is updated at each time step t based on the previous hidden state and the current input according to the layer definition

$$\boldsymbol{h}_t = a_t(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t), \tag{2.20}$$

where $\boldsymbol{x}_t$ is the input vector at time step t, $\boldsymbol{W}_{hh}$ represents the weights applied to the hidden state from the previous time step, which are shared across time and $\boldsymbol{W}_{xh}$ is the weight matrix for the input at the current time step. a_t is again a nonlinear activation function such as the ReLU. RNN layers can be stacked together in order to better model higher order temporal interactions and they can also be combined with convolutional layers (in particular for feature encoding temporal image data) and feed-forward layers (commonly for the output layers of the network).

This recurrent neural network (RNN) form inherently faces difficulties in modeling long-range dependencies within sequences. This limitation is closely associated with the spectral properties of the recurrent weight matrix $\boldsymbol{W}_{hh}$. During the optimization process the eigenvalues of $\boldsymbol{W}_{hh}$ have significant implications for the network's ability to propagate information through time. Informally, eigenvalues of the matrix with small magnitudes can cause gradients to vanish (see section 2.3). This phenomenon results in the RNN's inability to retain and utilize long-term dependencies within the data. On the other hand, if the eigenvalues are large, the network is susceptible to the gradients growing exponentially during back-propagation, rendering the optimization process unstable (see section 2.3). To address these limitations, variations of RNNs have been designed with gating mechanisms within the recurrent component of the layer [64, 65]. For example, the Gated Recurrent Unit (GRU) which modifies the standard hidden state update rule with two gates: the update gate and the reset gate;

$$\boldsymbol{u}_t = \sigma(\boldsymbol{W}_u[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]), \tag{2.21}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]), \tag{2.22}$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_h[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t]), \tag{2.23}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{u}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{u}_t \odot \tilde{\boldsymbol{h}}_t, \tag{2.24}$$

where $\boldsymbol{u}_t$ is the update gate vector, $\boldsymbol{r}_t$ is the reset gate vector, $\tilde{\boldsymbol{h}}_t$ is an activation vector, $\boldsymbol{W}_u$, $\boldsymbol{W}_r$, and $\boldsymbol{W}_h$ are parameter matrices and $\odot$ denotes the Hadamard product (element-wise multiplication). Intuitively, the update gate helps the model determine how much of the past information needs to be passed along to the future, while the reset gate decides how much of the past information to forget. Mechanistically it makes it far more difficult for the mentioned eigenvalues above to become too big or too small during optimization and so results in a more robust model that is better able to learn long term dependencies. One apparent downside of the RNN architecture is that it is difficult to implement in

a highly paralleled way to best exploit GPU computation. In the next section we discuss another architecture that is very effective for sequence modelling that is more naturally parrellelizable.

2.2.4 Transformers

The final NN variant we discuss is that of the Transformer [10]. The transformer architecture has won in popularity over the RNN for sequence modelling for popular tasks like language modelling. It forms the basis for the large pre-trained generative models that are the subject of the latest AI trends and the scaling laws as discussed in section 1. At the core of the transformer is the inductive bias of attention for sequence modelling. Attention is favoured for certain use cases over the recurrent layer introduced in the last section due to its innate parallelization capabilities, significantly enhancing computational efficiency on specialized harder like GPUs.

The attention mechanism [10] assigns importance to each element in a sequence, enabling the model to focus selectively on different parts of the sequence when making predictions. For a sequence of input vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$, we first stack and transform these inputs into different matrices for queries $\boldsymbol{Q}$, keys $\boldsymbol{K}$, and values $\boldsymbol{V}$ (according to the typical transformer terminology for the projections of input vectors) through learned parameter matrices specific to each.

$$\boldsymbol{Q} = \boldsymbol{W}_Q \begin{bmatrix} \vdots \\ \boldsymbol{x}_t \\ \vdots \end{bmatrix}, \qquad \boldsymbol{K} = \boldsymbol{W}_K \begin{bmatrix} \vdots \\ \boldsymbol{x}_t \\ \vdots \end{bmatrix}, \qquad \boldsymbol{V} = \boldsymbol{W}_V \begin{bmatrix} \vdots \\ \boldsymbol{x}_t \\ \vdots \end{bmatrix}, \tag{2.25}$$

where each query (i.e. row) in $\boldsymbol{Q}$, key in $\boldsymbol{K}$, and value in $\boldsymbol{V}$ corresponds to one of the input vectors $\boldsymbol{x}_t$ and $\boldsymbol{\theta} = \{\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V\}$. The attention function is then

$$\mathcal{A}_{\boldsymbol{\theta}}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}. \tag{2.26}$$

The dot product $\boldsymbol{Q}\boldsymbol{K}^\top$ measures the similarity between queries and keys and the

softmax function converts these similarity scores into a probability distribution. The scaling factor in the denominator, where d_k is the dimensionality of the keys, helps with numerical stability during optimization. The output of the attention function is a weighted sum of the value vectors, where the weights reflect the relevance assigned to each input. The transformer architecture is composed of a stack of layers that usually follow the same structure. Each layer contains two main subcomponents: a multi-head attention and a position-wise feed-forward network.

Multi-head attention

The multi-head attention mechanism extends the attention mechanism by running it in parallel with different learned projections of queries, keys, and values. This allows the model to capture different aspects of the sequence information from different representation sub-spaces. For a given number of heads K, the multi-head attention is defined as:

$$\mathcal{A}^K_{\boldsymbol{\theta}}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{Concat}(\mathcal{A}_{\boldsymbol{\theta}_1}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}),\ldots,\mathcal{A}_{\boldsymbol{\theta}_K}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}))\boldsymbol{W}_O \tag{2.27}$$

where $\boldsymbol{\theta}_k$ is the parameter matrices for the k-th head and $\boldsymbol{W}_O$ is the output parameter matrix that combines the outputs from all the K attention heads.

Position-wise feed-forward network

Since the attention mechanism itself does not take into account the order of the sequence, positional encodings are often added to the input vectors to inject information about the position of the tokens in the sequence. The positional encodings have the same dimension as the input vectors so that the two can be summed together. The positional encodings can be defined using a specific function, which in the case of the original transformer is a combination of sine

and cosine functions with different frequencies:

$$\mathcal{P}(i, 2j) = \sin\left(\frac{i}{10000^{2j/d}}\right), \tag{2.28}$$

$$\mathcal{P}(i, 2j+1) = \cos\left(\frac{i}{10000^{2j/d}}\right), \tag{2.29}$$

where i is the position of the token in the sequence, j is the index in the positional encoding vector. The intuition behind this choice is that these functions can provide a unique encoding for each position (up to a certain sequence length). The use of geometrically spaced frequencies (as in the term $10000^{2j/d}$) means that each subsequent dimension of the positional encoding corresponds to a wavelength that is longer than the previous one. This helps the model to capture both fine-grained and coarse positional relationships. The patterns created by these frequencies allow the model to learn how far apart tokens are in the sequence (their relative positions), which is key to understanding the structure and meaning within sequences such as sentences.

In chapters 3 and 6 we make heavy use of variations of the transformer model and focus on natural language processing applications in our experiments.

2.3 Learning with gradients

In section 2.1 we introduced a probabilistic modeling perspective on machine learning and established maximum likelihood estimation (MLE) as a principled objective for optimizing neural network parameters that parameterize the probability distributions we aim to learn from data. Section 2.2 then explored the specific functional forms of neural networks that are relevant to this work. We now turn our attention to the optimization process that underpins the learning of neural network parameters within this framework; with gradients of the objective w.r.t to $\boldsymbol{\theta}$ being the key.

2.3.1 Stochastic gradient descent

Gradient descent and its stochastic variants are the backbone of neural network optimization. Given a differentiable objective function $\mathcal{L}(\boldsymbol{\theta}) \in \mathbb{R}$, which typically

represents the negative log-likelihood in our case, the goal is to find the parameters $\boldsymbol{\theta}$ that minimize this function. Unfortunately for our choices of model, there is no closed-form solution to this minimization problem and so we instead aim to find a good local minimum using gradient descent methods. The core principle of gradient descent is to iteratively update $\boldsymbol{\theta}$ (from a carefully chosen initialization) in the direction that most steeply decreases $\mathcal{L}$ using an update rule at iteration t until some convergence criteria is met:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t). \tag{2.30}$$

α denotes the learning rate that determines the step size of each update and $\nabla_{\boldsymbol{\theta}}$ the first order gradient of the scalar objective.

An important insight towards scaling up the training of NNs to arbitrarily large datasets is that replacing the full gradient in equation 2.30 (which is computed over all of $\mathcal{X}_{train}$) with the gradient w.r.t a much smaller sampled mini-batch of data works well in practise. This stochastic gradient descent (SGD) variant is based on the unbiased Monte Carlo estimate of the gradient by sampling (without replacement) a mini-batch of data $\mathcal{B} \subset \mathcal{X}_{train}$ of size B and computing the approximate gradient

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \approx \frac{N}{M} \sum_{i}^{B} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{x}^i)). \tag{2.31}$$

In practise each so-called epoch of training consists of iterating through the full training dataset worth of mini-batches, updating the parameters with each mini-batch using SGD. There are many variants of SGD that are found to perform better in practise. One such method that we use exclusively in our experiments is ADAM [66], which computes adaptive learning rates for each parameter using past gradients. For a thorough introduction on this topic see chapter 8 of [5].

2.3.2 Automatic differentiation

To calculate the gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ required for these optimizers, deep learning frameworks leverage automatic differentiation techniques. Unlike symbolic differentiation or numerical finite differences approaches, these are techniques that recursively apply the chain rule of calculus to compute derivatives computationally efficiently. Autodiff underpins the operation of modern deep learning libraries such as PyTorch [67] and JAX [68] that we use throughout this thesis to implement our experiments. Consider computing the gradient w.r.t to the parameters $\boldsymbol{W}_L$ of the final layer of the neural network as defined in section 2.2.1 using the chain rule:

$$\nabla_{\boldsymbol{W}_L}\mathcal{L}(\boldsymbol{h}_{L-1}, \boldsymbol{y}, \boldsymbol{W}_L) = \frac{\partial \boldsymbol{p}_L}{\partial \boldsymbol{W}_L}\frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{p}_L}\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_L}, \tag{2.32}$$

where $\boldsymbol{p} = \boldsymbol{W}\boldsymbol{h}$. Notice that $\boldsymbol{h}_L$ is a function of parameters $\boldsymbol{W}_{L-1}$ that we would also like the derivatives of; and so in practise this chain rule is applied again recursively across all layers of the network. We can illustrate this computation as an Abstract Syntax Tree (AST) (also known as a computation graph/tree) to help describe the workings of Autodiff:

Autodiff frameworks generally decompose the computation of a neural network into such a directed graph with nodes for each elementary operation for which derivatives are well-defined. During training, when evaluating the loss for a given batch of inputs and the latest parameter values, which is referred to as a forward pass, intermediary values n calculated at each node i are stored. These are the input values needed to compute the partial derivatives for the chain rule at this node. Then, in order to compute the overall derivative w.r.t to all the parameters $\nabla_{\boldsymbol{\theta}}\mathcal{L}$, the graph is traversed in reverse ancestral (or backwards) order implementing the following Autodiff procedure:

1. Define $t_1 = 1$ for the first node of the reverse graph.

2. For the next node i in the backwards traversal, find the child nodes $C(i)$

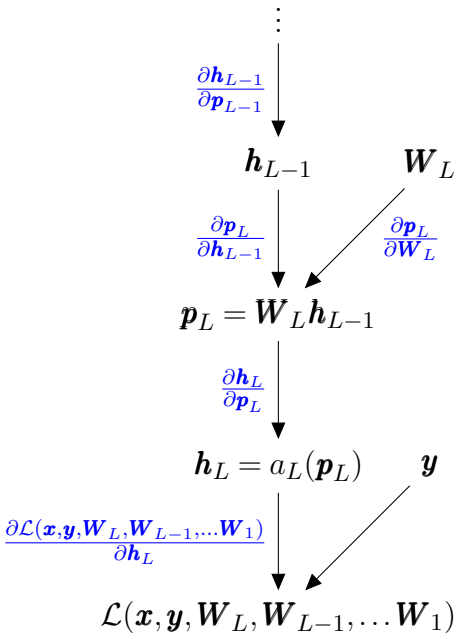


Figure 2.1: Computation graph focusing in on the final layer of a feed-forward neural network. The nodes contain the intermediary operations required by the forward pass and the edges are annotated with the relevant partial derivatives needed by the chain rule.

and define:

$$t_i = \sum_{c \in C(n)} \frac{\partial n_c}{\partial n_i} t_c \tag{2.33}$$

3. Repeat this process until you reach the root nodes of the graph.

4. The total derivatives of the final node w.r.t to the root nodes (in our case the layer-wise parameter values $\boldsymbol{W}_l$) are given by the values for t at those nodes.

This so called backward pass is efficient because it avoids redundant calculations and information is split between parents only when required. This procedure is generally referred to as backwards mode Autodiff, or back propagation [69, 70].

Autodiff is broader than just variations of this backpropagation algorithm. There is also forward mode Autodiff. Forward-mode instead computes the derivative of each intermediate node with respect to a single input variable during the forward pass of the graph and can be less memory intensive than

backwards mode. However, this variation, unlike backpropagation, is less well suited to problems with many inputs and a single output, which is characteristic of the loss functions we deal with for training our neural network models. Forward mode can be more suitable for functions that have a low-dimensional input space and a high-dimensional output space. For further reading see [71].

2.3.3 Reparameterization trick

In section 2.1.1 we introduced latent variable models and the ELBO for jointly training the parameters of our model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ and inference network $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. Using the gradient based methods naively to optimize the ELBO w.r.t $\boldsymbol{\phi}$ can be challenging due to the high variance of the Monte Carlo estimate:

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] \approx \frac{1}{S} \sum_{s=1}^{S} \Big(\nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}^s|\mathbf{x}) \Big) \log p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}^s). \tag{2.34}$$

The so-called reparameterization trick is a way to provide a much lower variance unbiased estimate of this same gradient [59]. With this approach we reparameterize $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ as a differentiable transformation of some other random variable $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. This does restrict our choice of variational distribution to those that can be transformed in this way; such as the Gaussian distribution. With this reparameterization we have $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x}))]$ with gradient estimate:

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x}))] && (2.35)\\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\phi}} f(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x}))] && (2.36)\\
&\approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\phi}} f(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^s, \mathbf{x})) && (2.37)
\end{aligned}$$

A common choice that we use in this work is a Gaussian distribution where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$ then $\boldsymbol{z} = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}) \odot \boldsymbol{\epsilon}$ has the distribution $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\mathbf{x}))$. In context of Autodiff, this moves the stochastic variable into an input leaf node on the graph that's not dependent on the parameters.

Chapter 3

Integrated Weak Learning

In this chapter we focus on the problem of improving data efficiency in supervised learning. As introduced in chapter 1, data efficiency in this setting refers to the number of labelled datapoint pairs required to achieve a target performance level. We focus on the paradigm of weak learning.

The general research question that motivates weak learning is given a relatively little reliably-annotated data and a set of weaker sources of noisy labels, how should one best combine them to train a supervised machine learning model? In this work, we focus on how to best parameterise and train the label and end model during training?

Our contributions are as follows:

- We introduce Integrated Weak Learning, a general framework that integrates weak supervision into the training process of traditional supervised models. Our approach jointly trains the original model and a label model that aggregates multiple sources of weak supervision.

- We introduce a label model that can learn to aggregate weak supervision sources differently for different datapoints and takes into consideration the performance of the end-model during training.

- We show that our approach outperforms existing weak learning techniques across a set of 6 benchmark classification datasets using mainly transformer based architectures as introduced in section 2.2. When both

a small amount of labeled data and weak supervision are present we observe an increase in performance of on average between 2-5 point test F1 points and as much as a 20 point gain over non-integrated methods.

3.1 Weak learning

To overcome the cost of manual data annotation, it has become increasingly common to include cheaper but less reliable sources of supervision when training deep learning models [39, 72, 73, 74, 75]. These noisy sources of supervision might include crowd labels, weaker models, distant supervision by knowledge bases, or manually curated heuristic rules, amongst others[72]. Given relatively little reliably-annotated data and a set of weaker sources of noisy labels, how should one best combine them to train a supervised machine learning model?

Early attempts at answering this question [72] typically decompose the problem into two stages. They first stage considers how to form an estimate of the unobserved label given a set of noisy labels and then consider how to train an end-model on the denoised labels. This approach has had considerable practical success [74, 76], enabling deep learning systems to be deployed in industry without manual labeling. Two-stage methods have the advantage that after denoising, the rest of the training pipeline remains essentially unchanged. To achieve this though, they make quite an unnatural independence assumption; they typically ignore the dependence of the approximate labels on the input data. The cost of this assumption is that valuable information from the end-model can not be incorporated in denoising and the estimated accuracy of different supervision sources is fixed across the entire dataset. In addition, in general, a small amount of task-specific oracle sourced labels continues to be critical for reliable results [74, 77], even when given access to pre-trained models in transfer learning settings [78].

In this chapter we present a modeling framework, Integrated Weak Learning (iWL), that simultaneously denoises the weak supervision sources and trains the end-model. Though there have been other approaches to joint model training

and denoising [79, 80], they have typically relied on intuitive heuristics, required extensive changes to the training objective and have high empirical variance. Our approach is based on a simple application of maximum likelihood learning in an appropriately chosen probabilistic model that assumes that the true label is an unobserved latent variable (as discussed in section 2.1). It can learn to exploit the relevant expertise of different weak supervision sources, whilst taking into consideration the performance of the end-model.

In traditional supervised learning we have access to a set of labeled data pairs $\{(\boldsymbol{x}^1, y^1), \cdots, (\boldsymbol{x}^N, y^N)\}$ that are identically and independently (i.i.d) sampled from a true underlying joint distribution $(\boldsymbol{x}^n, y^n) \sim p_d(\mathbf{x}, \mathrm{y}) = p_d(\mathrm{y}|\mathbf{x})p(\mathbf{x})$. We are interested in applications where y_n is expensive (in time and/or cost) to define, such as when a judgement from a human expert such as a doctor is required. A model $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ is then specified to approximate the true conditional distribution $p_d(\mathrm{y}|\mathbf{x})$. In the classification setting, each label y takes a discrete value in $\{1, \ldots, C\}$ where C is the number of output classes for the problem. The parameter $\boldsymbol{\theta}$ can be estimated by maximum likelihood estimation as discussed in section 2.1 using the objective

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(y^n|\boldsymbol{x}^n). \tag{3.1}$$

In weak learning (WL), instead of requiring access to labels on all datapoints, we assume access to K weak supervision sources $\{\rho_1(\tilde{\mathrm{y}}|\mathbf{x}), \ldots, \rho_K(\tilde{\mathrm{y}}|\mathbf{x})\}$. Each $\rho_k(\tilde{\mathrm{y}}|\mathbf{x})$ can provide an approximate labels $\tilde{y}_k$ given an input data $\boldsymbol{x}$, resulting in a weakly labeled dataset $\mathcal{W} = \left\{(\boldsymbol{x}^1, \tilde{y}_1^1, \ldots, \tilde{y}_K^1), \ldots, (\boldsymbol{x}^W, \tilde{y}_1^W, \ldots, \tilde{y}_K^W)\right\}$, where W is the number of data-points that have weak labels only. In the data programming formalism [72] these supervision sources are encapsulated into so-called *labeling functions*. They usually can easily be applied across large unlabeled datasets. In principle, the domain of $\tilde{y}$ can be different from the domain of the true label y. For example, in our experiments we use labeling functions that can return one of $C+1$ classes $\tilde{y} \in \{0, \ldots, C\}$, where the class

0 represents an abstain where the labeling function refrains from making a decision.

If the weak labels have some useful signal for the true data generating process, then they can still serve as a useful source of supervision for training an end-model. Each of the labeling functions may not be very informative about the true label across all data points, but may have some useful specialisation for certain regions of the inputs space; and collectively, if combined appropriately, these sources may help accurately predict the true label. The simplest approach to aggregating weak supervision sources is to take the majority vote label from the weak labels for each datapoint. Popular WL approaches improve on this by training a probabilistic model that learns an estimate for the accuracy of each labeling function and use this to calculate the distribution $p(\mathrm{y}|\tilde{\mathrm{y}}_1^n, \ldots, \tilde{\mathrm{y}}_K^n)$ [1, 73].

In real world applications it is common for the practitioner to have access to both weak labels and a (relatively small) set of training data with 'strong' labels (i.e. oracle labels we assume come from the true underlying data distribution, such as those generated by human subject-matter experts). With this in mind, we denote the dataset that contains both strong and weak labels as $\mathcal{S} = \left\{(\boldsymbol{x}^1, y^1, \tilde{y}_1^1, \ldots, \tilde{y}_K^1), \ldots, (\boldsymbol{x}^S, y^S, \tilde{y}_1^S, \ldots, \tilde{y}_K^S)\right\}$, where $y^s \sim p_d(\mathrm{y}|\boldsymbol{x}^s)$.

In the next section we present a simple probabilistic graphical model that makes it possible to learn the parameters of the model $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ both when the only labels are from weak supervision and when there are some strong labels present.

3.2 A joint training approach

Given a weakly supervised dataset $\mathcal{W}$, we wish to specify a probabilistic model over the weak label $\tilde{\mathrm{y}}$ and the input $\mathbf{x}$ that would allow us to jointly learn an end-model $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ and learn to leverage the weak labels (which is also commonly referred to as denoising the weak labels). A simple approach would

be to parameterize the full distribution without loss of generality as:

$$p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathbf{x}) = \sum_{\mathrm{y}} p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y},\mathbf{x}) p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x}), \tag{3.2}$$

where we have marginalized over the unobserved true label y. However, this simple model has a potentially degenerate solution in which the distribution $p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y},\mathbf{x})$ becomes independent or approximately independent of the label y, i.e the model learns to ignore the latent $p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y},\mathbf{x}) \approx p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathbf{x})$. If that were to happen, then there would be no sharing of information between the weak labels and the true label y. In order to ensure that the weak labels contribute to the training of the end-model, we must constrain the distribution $p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y},\mathbf{x})$ so that the information flow from $\mathbf{x}$ to $\tilde{\mathrm{y}}$ is limited. The simplest way to achieve this constraint is to introduce an assumption that $\tilde{\mathrm{y}}$ is independent of $\mathbf{x}$ given y:

$$p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathbf{x}) = \sum_{\mathrm{y}} p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y}) p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x}). \tag{3.3}$$

We refer to $p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y})$ as the label model and $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ is free to be any parametric end-model. We explore alternative forms of the label model in the following section.

This model formulation allows us to jointly estimate the parameters $\{\boldsymbol{\phi},\boldsymbol{\theta}\}$ using maximum likelihood estimation as introduced in section 2.1. In practice we have K weak supervision sources. If we assume these sources are conditionally independent given y, our model becomes

$$p_{\boldsymbol{\theta},\boldsymbol{\Phi}}(\tilde{\mathrm{y}}_1,\ldots,\tilde{\mathrm{y}}_K|\mathbf{x}) = \sum_{\mathrm{y}} p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x}) \prod_{k=1}^{K} p_{\boldsymbol{\phi}_k}(\tilde{\mathrm{y}}_k|\mathrm{y}), \tag{3.4}$$

where we denote $\boldsymbol{\Phi} = \{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_K\}$. Figure 3.1a shows the corresponding graphical model.

The log-likelihood for the data-points that include both strong and weak labels $\mathcal{S} = \left\{(\boldsymbol{x}^1, y^1, \tilde{y}_1^1,\ldots,\tilde{y}_K^1),\ldots,(\boldsymbol{x}^S, y^S, \tilde{y}_1^S,\ldots,\tilde{y}_K^S)\right\}$ becomes:

$$\mathcal{L}_S(\boldsymbol{\theta}, \boldsymbol{\Phi}) \equiv \frac{1}{S} \sum_{s=1}^{S} \left(\log p_{\boldsymbol{\theta}}(y^s|\boldsymbol{x}^s) + \log \sum_{y^s} p_{\boldsymbol{\theta}}(y^s|\boldsymbol{x}^s) \prod_{k=1}^{K} p_{\boldsymbol{\phi}_k}(\tilde{y}_k^s|y^s) \right). \quad (3.5)$$

For the data-points that only have weak labels $\mathcal{W} = \left\{ (\boldsymbol{x}^1, \tilde{y}_1^1, \ldots, \tilde{y}_K^1), \ldots, (\boldsymbol{x}^W, \tilde{y}_1^W, \ldots, \tilde{y}_K^W) \right\}$ we have the log-likelihood term

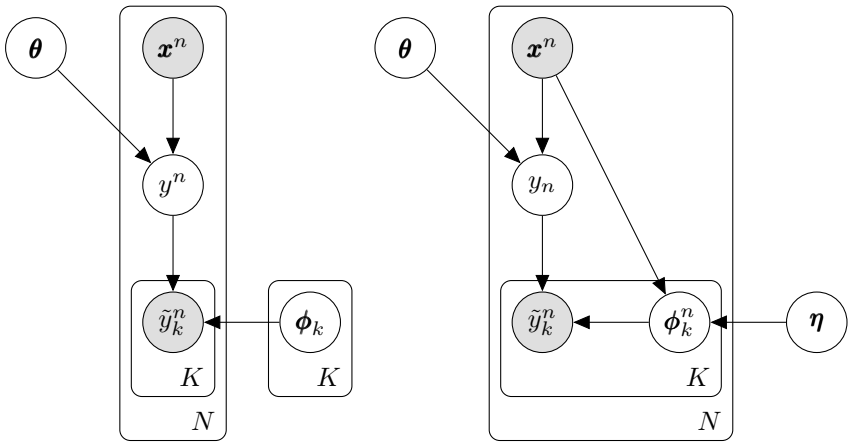
$$\mathcal{L}_W(\boldsymbol{\theta}, \boldsymbol{\Phi}) \equiv \frac{1}{W} \sum_{w=1}^{W} \log \sum_{y^w} p_{\boldsymbol{\theta}}(y^w|\boldsymbol{x}^w) \prod_{k=1}^{K} p_{\boldsymbol{\phi}_k}(\tilde{y}_k^w|y^w). \quad (3.6)$$

Therefore, our overall likelihood training objective for integrating weak supervision (iWL) into model training is

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\Phi}) = \lambda_S \mathcal{L}_S(\boldsymbol{\theta}, \boldsymbol{\Phi}) + \lambda_W \mathcal{L}_W(\boldsymbol{\theta}, \boldsymbol{\Phi}), \quad (3.7)$$

where λ_S, λ_L are user defined scaling parameters that control how much the strongly and weakly labeled datasets influence the model training. We can then jointly estimate both sets of parameters $\{\boldsymbol{\Phi}, \boldsymbol{\theta}\}$ by directly optimising objective 3.7. The label model can then learn to take into consideration the performance of the end-model during training because the gradients used to update the label model parameters will contain information about how the end model performance is impacted under out joint objective. Once trained, the discriminator $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ can be used independently to the label model to make predictions. This framework is flexible for supporting different choices of label and end-model.

In practise we can use stochastic gradient descent to train both sets of parameters jointly by sampling mini-batches of data from $\mathcal{W}$ and $\mathcal{S}$ to calculate L^S and L^W respectively for each parameter update step - see section 3.4 for further details. We compute the marginalization over y exactly in equations 3.5 and 3.6. This is efficient in this setting because the number of classes C to sum over tends to be relatively small.



(a) Basic graphical model (iWL) **(b)** Including $\boldsymbol{x}$ (iWLD)

Figure 3.1: Graphical models for the integrated weak learning variants. $\boldsymbol{\theta}$ and $\boldsymbol{\Phi} = \{\boldsymbol{\phi}^1, \ldots, \boldsymbol{\phi}^K\}$ are the parameters of the end-model and label model respectively. K is the number of available weak supervision sources and N the number of observed training datapoints. The left model assumes the generative process is the same for all datapoints, whereas the right model assumes similar datapoints will have a similar noisy label generative process.

3.2.1 Design of label model

In the previous section, we assumed that the weak labels only depend on the underlying true label y, yielding the label model in equation 3.4. In this case $p_{\boldsymbol{\phi}_k}(\tilde{\mathrm{y}}|\mathrm{y})$ is parameterized by a linear transition matrix i.e. $p_{\boldsymbol{\phi}_k}(\tilde{\mathrm{y}}_k = i|\mathrm{y} = j) = \boldsymbol{\phi}_k^{ij}$, with $i \in \{1, \ldots, C+1\}$, including the abstain label, and $j \in \{1, \ldots, C\}$. Each row in the transition matrix sums to one: $\sum_i \boldsymbol{\phi}_k^{ij} = 1$.

A natural extension is to consider incorporating a dependency on $\mathbf{x}$. The label model could then represent different transitions for different data-points. Intuitively this would allow the label model to understand the relevant expertise of the different labeling functions and emphasize appropriately. This is particularly relevant given in practice labeling functions tend to be quite specialized in the data-points they perform well on [39]. However, we must take care when introducing $\mathbf{x}$-dependence to constrain the flow of information so that the label model does not become independent of y.

We therefore also introduce a variation where we parameterize the transition matrix $\boldsymbol{\phi}_k$ itself as a function of $\mathbf{x}$ - see figure 3.1b for the updated graphical

model. This ensures that $p(\tilde{\mathrm{y}}|\mathbf{x},\mathrm{y})$ is still parameterized by a constrained linear transition matrix. The label model becomes $p(\tilde{\mathrm{y}}_k|\mathrm{y},\boldsymbol{\phi}_k = f_{\boldsymbol{\eta}}(\mathbf{x})[k])$, where $f_{\boldsymbol{\eta}}(\cdot)$ is a neural network that maps a data point $\boldsymbol{x}$ to the K linear transition matrices and we use $f_{\boldsymbol{\eta}}(\mathbf{x})[k]$ to denote the kth matrix. This allows our label model to produce similar transitions $\boldsymbol{\phi}_k$ for similar $\boldsymbol{x}$ and amortises the cost of computing the transition for each data-point. Therefore, the full model can be written as

$$p_{\boldsymbol{\eta},\boldsymbol{\theta}}(\tilde{\mathrm{y}}_1,\ldots\tilde{\mathrm{y}}_K|\mathbf{x}) = \sum_{\mathrm{y}} p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x}) \prod_{k=1}^{K} p(\tilde{\mathrm{y}}_k|\mathrm{y},\boldsymbol{\phi}_k = f_{\boldsymbol{\eta}}(\mathbf{x})[k]). \tag{3.8}$$

This model extends equation 3.4 with a more complex label model parameterized by $\boldsymbol{\eta}$ instead of $\boldsymbol{\phi}_k$. In practise the objective and training procedure is the same as that described in section 3.2 using stochastic gradient descent and exact marginalisation over y, but we replace $\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\Phi})$ with $\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\eta})$. Hence during training we are learning the parameters $\boldsymbol{\eta}$ of our label model neural network that learns to output the transition matrices given an $\boldsymbol{x}$, as opposed to directly learning the transition matrices independent of $\boldsymbol{x}$.

In principle, it is still possible though unlikely that the label model $p(\tilde{\mathrm{y}}_k|\mathrm{y},\boldsymbol{\phi}_k = f_{\boldsymbol{\eta}}(\mathbf{x})[k])$ becomes independent of the label y because it learns to ignore y - for example in the regime where we have very little or no strong labels to help infer the latent configuration during training. In section 3.4 we validate that this does not happen in practice and demonstrate how incorporating the $\mathbf{x}$ dependency in this way improves performance across a range of benchmark problems. As with existing weak learning techniques, it is possible to extend our label model to more explicitly consider correlations between labeling functions - see appendix A.4 for more details.

3.2.2 Discussion of model

As with prior works [1], our model can suffer from y being unidentifiable. Fortunately, in our framework, access to strongly labeled data naturally mitigates against this. For notational simplicity we consider the case of only one labeling function $\rho(\tilde{\mathrm{y}}|\mathbf{x}) = \sum_{\mathrm{y}} p_{\boldsymbol{\phi}_*}(\tilde{\mathrm{y}}|\mathrm{y}) p_d(\mathrm{y}|\mathbf{x})$ where $p_d(\mathrm{y}|\mathbf{x})$ is the true label generation

distribution and we assume $p_{\boldsymbol{\phi}_*}(\tilde{\mathrm{y}}|\mathrm{y})$ parameterized by a linear transition matrix, i.e. $p_{\boldsymbol{\phi}_*}(\tilde{\mathrm{y}}=i|\mathrm{y}=j)=\boldsymbol{\phi}_*^{ij}$ with known parameters $\boldsymbol{\phi}_*$. In this case, training our model $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ using equation 3.7 is able to identify the true underlying $p_d(\mathrm{y}|\mathbf{x})$. Specifically, maximizing the likelihood function is equivalent to minimizing the KL divergence

$$\mathbb{E}_{\rho(\tilde{\mathrm{y}}|\mathbf{x})}(\log p_{\boldsymbol{\theta},\boldsymbol{\phi}_*}(\tilde{\mathrm{y}}|\mathbf{x}))=-\mathrm{KL}\left(\sum_{\mathrm{y}}p_{\boldsymbol{\phi}_*}(\tilde{\mathrm{y}}|\mathrm{y})p_d(\mathrm{y}|\mathbf{x})||\sum_{\mathrm{y}}p_{\boldsymbol{\phi}_*}(\tilde{\mathrm{y}}|\mathrm{y})p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})\right)+const, \tag{3.9}$$

where the constant is the entropy of $p_{\boldsymbol{\phi}_*}(\tilde{\mathrm{y}}|\mathrm{y})$ with fixed $\boldsymbol{\phi}_*$. During training, as $\boldsymbol{\theta}\to\boldsymbol{\theta}_*$ the KL divergence goes to 0 and we have

$$\sum_j \boldsymbol{\phi}_*^{ij}p_d(\mathrm{y}=j|\mathbf{x})=\sum_j \boldsymbol{\phi}_*^{ij}p_{\boldsymbol{\theta}_*}(\mathrm{y}=j|\mathbf{x})\quad\forall i. \tag{3.10}$$

As long as the linear transition matrix $\boldsymbol{\phi}_*$ (with size $C+1\times C$) does not degenerate (i.e. has rank C), then the mapping from the distribution of y to $\tilde{\mathrm{y}}$ is injective. We then have $p_{\boldsymbol{\theta}_*}(\mathrm{y}|\mathbf{x})=p_d(\mathrm{y}|\mathbf{x})$, which successfully recovers the underlying true model. To note, when $\boldsymbol{\phi}$ is a function of $\boldsymbol{x}$ (3.1b) the same reasoning can be applied if we assume the true transition generation function $f_{\boldsymbol{\eta}_*}$ is also known.

When the true label model $p_{\boldsymbol{\phi}_*}(\tilde{\mathrm{y}}|\mathrm{y})$ is unknown, we need to learn $\boldsymbol{\phi}$. In this case we cannot guarantee to identify the true underlying model using only the weak labels. To give an example, we first assume that we have learned a label model $p_{\boldsymbol{\phi}_1}(\tilde{\mathrm{y}}|\mathrm{y})$ using MLE with a model $p_{\boldsymbol{\theta}_1}(\mathrm{y}|\mathbf{x})$. We can always construct an alternative label model $p_{\boldsymbol{\phi}_2}(\tilde{\mathrm{y}}|\mathrm{y})$ (with $\boldsymbol{\phi}_2=\boldsymbol{\phi}_1\times\boldsymbol{M}^{-1}$, where $\boldsymbol{M}$ is a $C\times C$ invertible transition matrix, and model $p_{\boldsymbol{\theta}_2}(\mathrm{y}|\mathbf{x})$ (with $p_{\boldsymbol{\theta}_2}(\mathrm{y}=i|\mathbf{x})=\sum_j \boldsymbol{M}_{ij}p_{\boldsymbol{\theta}_1}(\mathrm{y}=j|\mathbf{x})$) that can give the same marginal distribution:

$$\sum_{\mathrm{y}}p_{\boldsymbol{\phi}_1}(\tilde{\mathrm{y}}|\mathrm{y})p_{\boldsymbol{\theta}_1}(\mathrm{y}|\mathbf{x})=\sum_{\mathrm{y}}p_{\boldsymbol{\phi}_2}(\tilde{\mathrm{y}}|\mathrm{y})p_{\boldsymbol{\theta}_2}(\mathrm{y}|\mathbf{x}). \tag{3.11}$$

This observation easily generalizes to the case of K weak supervision sources. Similar counter examples are constructed in the context of disentangled repre-

sentation learning, where the true representation cannot be identified under maximum likelihood learning [81].

Incorporating strong labels $y \sim p_d(\mathrm{y}|\mathbf{x})$, like we discuss in section 3.2, can alleviate this unidentifiability issue. Intuitively, when the number of strong labels goes to infinity $S \to \infty$, then the first term in Equation 3.5 $\frac{1}{S}\sum_{s=1}^{S} \log p_{\boldsymbol{\theta}}(y^s|x^s)$ will reach an optimum when $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x}) = p_d(\mathrm{y}|\mathbf{x})$, which is due to the consistency of MLE [82]. In practice, we find that a relatively small number of strong labels is effective at mitigating this issue allowing us to recover a useful end-model. We leave to future work exploring more deeply the relationship between the identifiability problem and the number of strong labels required. In the case where it is not possible to access any strong labels for a given problem, similar to prior work we can leverage the majority vote heuristic to initialize the parameters of our label model, which we find works well empirically.

3.3 Related work

Two-Stage Weak Learning

Two stage weak labeling methods separate the label model from end-model training [72, 1, 73, 75] . The primary advantage of this separation is that the cost of label denoising is paid only once and the change needed to training pipelines is minimal.

In our work it is necessary to alter the training objective by adding additional terms and one has to learn the parameters of the label model every time the end-model is changed. However, the additional computational cost of learning the label model $p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y})$ can be kept small by using a relatively smaller model compared to the end-model $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$. Our experiments provide evidence that this additional cost is rewarded by improved end-model performance, especially when some strong labels are present.

Joint Weak Learning

Most similar in spirit to our work are two end-to-end weak labeling methods that also jointly denoise and train: *WeaSel* [79] and *Denoise* [80]. These

methods differ both in how they parameterize the label model and in their training objective. The primary difference with our work is in the choice of training objective. Whereas we train using maximum-likelihood learning, Weasel uses a heuristic consistency constraint. Namely that the labels predicted by a denoising model and an end-model should agree. Training their heuristic objective can be unstable [79] and can result in degenerate solutions. In their paper the method is primarily justified by empirical performance but in our experiments it under-performed both Denoise and Integrated Weak Learning (see section 3.4). In contrast, our framework simply proposes an appropriate graphical model and trains via maximum likelihood with stochastic gradient descent.

The Denoise algorithm has an additional algorithmic component beyond weak supervision which incorporates self-supervision [83, 84] of its end-model. Confident predictions from the end-model are bootstrapped for learning where the labeling functions have low coverage. Based on their ablation studies, this significantly improves performance. Self-supervision could naturally be applied to our objective 3.7 to further bolster performance, which we leave for future work.

Learning with Noisy Labels

There are multiple different approaches to learning with noisy labels, including data cleaning [85, 86, 87], where useful information is potentially lost, and data re-weighting that weighs training data-points based based on different criteria [88, 89]. Most relevant to our line of work are those methods that attempt to correct noisy labels using a label model parameterized similarly to ours [90, 91, 92]. In the recent *Confident Learning* [90] they assume a probabilistic label model, treating the true underlying y as a latent, and do inference on a single $C \times C$ noise transition matrix in order to correct their noisy labels. Our approach differs in a few key ways. Firstly, our label model is integrated as part of the end-model training. Secondly, we are dealing with multiple transition matrices, one per each source of label noise, where the noisy label domain is

different than the true label domain. Lastly, our label model from figure 3.1b can learn different transitions for different $\boldsymbol{x}^n$, releasing the assumption that label noise needs to be constant across a dataset. In [92] they do in fact have a noise transition matrix that is dependent on $\boldsymbol{x}^n$, but they have an alternating training scheme for the end-model, instead of jointly training, and they too don't deal with multiple sources of noisy labels with different domains.

Crowdsourcing

Applications that deploy crowdsourcing to collect labeled data also face the challenge of how to aggregate multiple noisy sources of supervision. Each source being a human annotator as opposed to a labeling function. Major works in this area use the EM algorithm to jointly model the unknown true label and annotator skill [93, 94, 95]. Most similar to our work is [96], where the annotator skill model (equivalent to our label model) and end model are jointly optimized using a formulation similar to our basic model described in figure 3.1a. In contrast to [96], firstly we propose an objective that can incorporate ground truth data when available. Secondly, the domain of our noisy labels is different to the true label domain (labeling functions can choose to abstain but crowd workers cannot). This results in a non-square noise transition matrix, hindering the application of a trace regularizer as proposed in [96]. Lastly, we extend our probabilistic formulation (in figure 3.1b) to relax the assumption in [96] that the annotator skill is independent of $\boldsymbol{x}$, which we demonstrate leads to improved performance.

3.4 Experiments

The goal of our experiments is to provide a robust performance comparison between the variants of our iWL approach and the existing weak learning approaches discussed in section 3.3. In addition, we want to understand how the amount of strongly labeled data $\mathcal{L}$ impacts the performance of these methods for deep learning models, which may be of independent interest for practitioners. For our choice of datasets and the implementations of existing methods, we

leverage the recent comprehensive benchmark for weak supervision (WRENCH) [39]. Specifically, we use 6 of the classification problems that vary in dataset size as well as labeling function complexity - see table 3.1 for details.

This set of datasets provides some variance in the number of classes (from 3 up to 6) and number of labelling functions (from 5 up to 83). The number of classes directly impacts the number of parameters ϕ in our label and the number of strong labels required for inferring the latent effectively. An interesting future direction of research and ablation is to study the impact on our method of a much larger number of classes.

We compare to the two-stage weak learning approaches of Majority Vote and Snorkel [72], and to the end-to-end weak learning approaches of Denoise [80] and the more recent WeaSEL [79]. We refer to these as benchmark methods. We use the implementations of these methods available in the WRENCH benchmark.

Table 3.1: Attributes of the chosen classification datsets [39]

Dataset	#Classes	#LFs	#Train	#Validation	#Test
Census	2	83	10,083	5,561	16,281
IMDB	2	5	20,000	2,500	16,281
Yelp	2	8	30,400	3,800	3,800
SMS	2	73	4,751	500	500
AGNews	4	9	96,000	12,000	12000
TREC	6	68	4,965	500	500

3.4.1 Implementation details

Here we introduce our basic implementation details. Further details can be found in appendix A.2.

Discriminative model

We keep the end discriminative model $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ common across all methods in our comparisons and vary the label model accordingly. Specifically for $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ we use the distilled RoBerta transformer model [97] to provide a rich embedded representation for textual $\mathbf{x}$. This acts as input to a two-layer feed forward neural network model, with 100 hidden units in each layer and RelU

activation functions and a softmax final output. Across all experiments we use the Adam optimizer [66] with learning rate $1e^{-3}$ and mini-batch size of 128. As in WRENCH we do early stopping on the validation F1-score with a patience of 300 optimization iterations using the validation datasets provided. Specific to our proposed integrated approach: we set the hyperparameters λ_L, λ_W from equation 3.7 to 1 throughout - meaning we weight equally the contributions from the strong and weak labels in our objective. For the two-stage weak learning approaches of majority vote and snorkel, we use the probabilistic denoised labels, i.e. the softmax probability values (as opposed to the one-hot label values) and noise-aware loss objective as recommended in prior works as the highest performing configuration of these methods.

Label model

For our proposed approach we include both of the label model variants presented in figure 3.1 - including and excluding the dependency on $\mathbf{x}$. We refer to these as iWL and iWLD respectively in the results. For iWLD, the network $f_{\boldsymbol{\eta}}(\boldsymbol{x}) : \boldsymbol{x} \rightarrow \{\boldsymbol{\phi}_k, \ldots, \boldsymbol{\phi}_K\}$ uses the same architecture and hyperparameters as $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ as specified above, except for the structure of the final layer that instead outputs the linear transition matrix. In principle the label model can be of a much smaller network size than the end model; we leave further ablation around choice of model and impact on performance to future work. We initialize our label model parameters using the majority vote to help mitigate against the non-identifiability of y issue discussed in section 3.2.2. For the benchmark methods we use the default label model hyperparameters as provided by WRENCH except for the WeaSEL model. For the WeaSEL temperature hyperparameter we try values from the range $\{0.5, 1, 3, 5\}$ based on their recommendations and select the best performing value for each experiment configuration because we found this method to be sensitive to this parameter in our setup. In appendix A.3 we provide visualizations to illustrate how our label model is able to learn different transitions $\boldsymbol{\phi}_k^n$ for different datapoints.

Incorporating strong labels

We also evaluate all methods with different proportions of strong labels available at training time in addition to the weak labels. We believe this to be a realistic and important scenario for many real world applications. Specifically we evaluate the scenarios where 1%, 10%, 50% and 100% of the training data is strongly labeled (selected at random). The strong label log-likelihood term in our objective equation 3.5 means that our proposed approach can deal with this scenario by design. To ensure that the benchmark methods also benefit from these strong labels in our comparisons, we add an additional labeling function in these cases that outputs the strong label if available and abstains otherwise. For the two-stage weak learning approaches, we also include results for an alternative approach of leaving the labeling functions unchanged and instead replacing the resulting denoised label with the corresponding strong label if available when training the end-model $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ - see appendix A.1 for further details.

3.4.2 Results

We report the test F1-score as our main evaluation metric averaged over 5 random seeds alongside 1 standard deviation. In table 3.4.2 we summarize the performance results for the different methods, for different proportions of strongly labeled data, across all datasets considered. In figure 3.2, we plot the corresponding quartiles of the F1 scores to provide further insight into how the variability between seeds compares across the different methods. We also include a baseline where we train the end-model $p_{\boldsymbol{\theta}}(\mathrm{y}|\mathbf{x})$ only on the strong labels provided, ignoring any weak labels. We refer to this baseline as 'labels only' (LO).

We see from table 3.4.2 that our proposed approaches result in the best performing model (as measured by test F1 score averaged over 5 random seeds) in 19 out of the 30 cases and first or second best-performing in 28 out of the 30 cases. In particular, in the regime where you have a relatively small amount of strongly labeled data in addition to your weak labeling sources (i.e. the

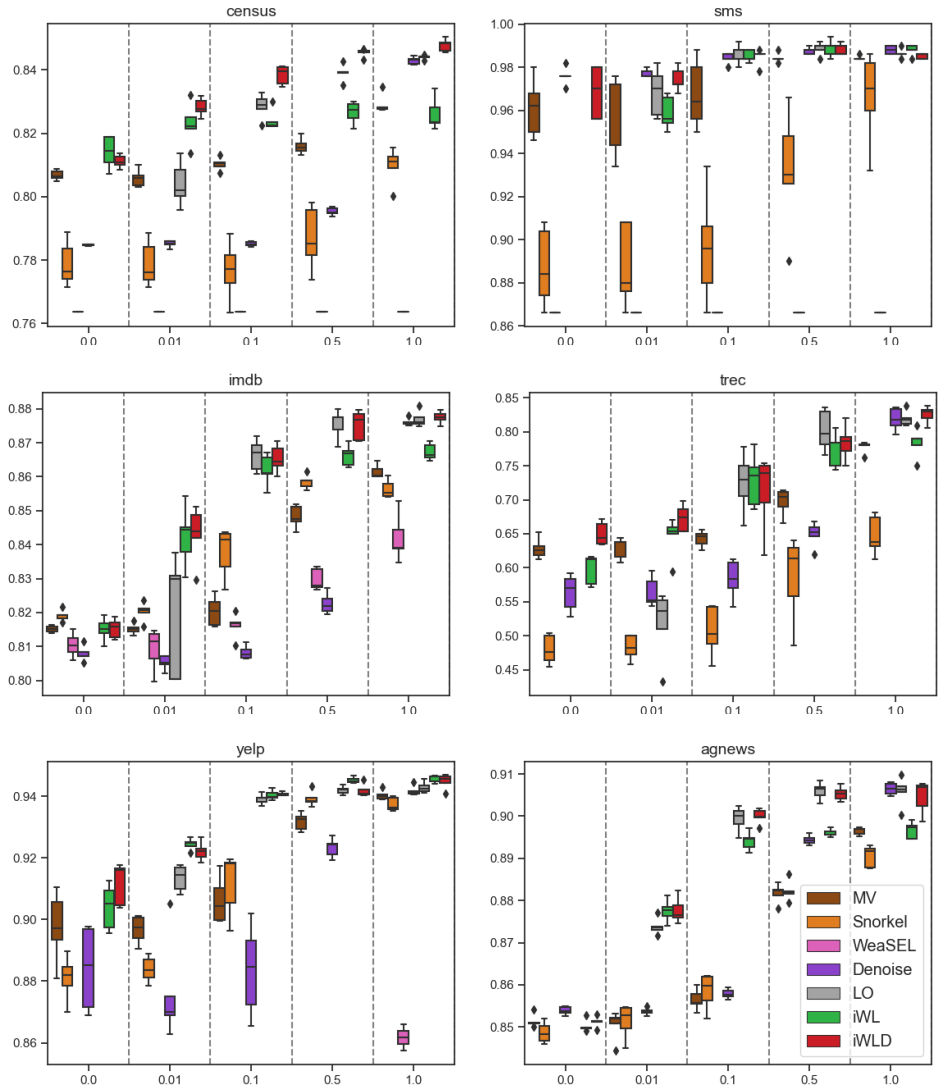


Figure 3.2: Box-plots of test F1 scores showing the quantiles across 5 random seeds on the y-axis. Each plot refers to a specific dataset. The models are grouped by the different fractions of strongly labeled data available on the x-axis (0%, 1%, 10%, 50%, 100%). In the majority of datasets and strong label splits, our proposed iWL and iWLD models are the best performing models and are robust to random seed changes shown by the relatively low variance. Methods that converge to significantly worse or degenerate solutions fall below a performance display threshold and are not shown. See table 3.4.2 for more detail.

1% row in table 3.4.2) our approach provides a conclusive improvement in all but one of the datasets. Here we consistently outperform the weak learning baselines by between 2 and 5 test-F1 points. In 5 of the 6 datasets we see that our integrated weak learning approach outperforms the LO baseline of

the end-model trained with 100% strong labels available. Furthermore, the boxplot quartiles in figure 3.2 illustrates that for a majority of the datsets and strong label splits, our iWLD model results in lower variance solutions than the benchmark methods.

Our results also provide some noteworthy insights related to the benchmark methods. Generating results across a range of different strong label proportions demonstrates that there is a tipping point at 10% over which LO becomes a competitive baseline. This can still represent a relatively significant amount of labeling effort (e.g. in AGNews this would be 9600 labels). It is likely that our use of the distilled RoBerta transformer as the feature extractor will be contributing to this performance, bringing some transfer learning benefits.

The joint approach Denoise is highlighted as the best performing approach in a small number of cases. This was unexpected because in the original WRENCH benchmark Denoise failed to outperform the other methods in any of these datasets. Furthermore, the more recent WeaSEL paper does not compare to Denoise as an end-to-end alternative. We note that Denoise, in addition to the weak and strong labels provided, also incorporates self-supervision signal into their training process. Our framework can naturally be extended with self-supervised labels which will likely further improve performance. This involves extending the dimensionality of our label model to consider another labelling function that takes predictions from the current state of the end model as the weak label. Finally, we experienced that WeaSEL failed with degenerate solutions in some of the experiments, for example for the TREC dataset, and performed for the most part worse than other methods. In an attempt to improve performance for WeaSEL, we tuned the temperature parameter as discussed in section 3.4.1.

3.5 Conclusion

This chapter introduced a new framework for improving the data efficiency of training supervised machine learning models that can principally integrate

both strong and weak supervision sources during training. It models the true underlying label y as a latent variable and jointly trains both the label model and end-model parameters using maximum likelihood. It is a generic framework that can be used in conjunction with existing supervised learning models to improve performance. We provide an empirical study across a range of classification benchmark problems of varying degrees of size and complexity and demonstrate that our approach consistently outperforms existing methods.

This work acts as a foundation for future work in many complimentary directions. Firstly, improving the label model to explicitly model correlations between weak supervision sources. We outline a provisional approach based on introducing an additional latent variable in appendix A.4. Secondly, exploring alternative parameterizations for our label model as presented in 3.2.1, including those that introduce constraints that remove the need to initialize with majority vote in the cases where no strong labels are available. Lastly, understanding how to combine our framework with related methods that attempt to mitigate the burden of gathering labeled data - namely self-supervised learning, active learning and transfer learning.

An application of particular interest is aligning large foundation models with human preferences (see chapter 6). It's common to get a wide array of slight different preferences for a given scenario. Weak learning approaches as outlined here could be an effective way to distil the wisdom of the crowd.

Table 3.2: Test F1 score averaged over 5 random seeds with 1 standard deviation in brackets across all datasets and all models considered: two stage weak learning approaches of Majority Vote (MV) and Snorkel. End-to-end weak learning approaches of WeaSEL and Denoise. The end-model $p_\theta(y|x)$ trained with the available strong labels only (labels only - LO). Our proposed integrated training approach with and without the x dependency in the label model (iWL and iWLD respectively). The results are grouped by the different proportions of strongly labeled data available (0%, 1%, 10%, 50%, 100%). Values highlighted in **red** indicates best performing and **blue** indicates second best. Our proposed approaches are the best performing models as measured by average test F1 score in 19 out of the 30 cases and the first or second best performing models in 28 out of the 30 cases.

Labels	**Dataset** / **Model**	AGNews	Census	IMDB	SMS	TREC	Yelp
0%	MV	**85.14** (0.16)	80.68 (0.16)	81.42 (0.10)	96.12 (1.38)	62.80 (1.54)	89.75 (1.16)
	Snorkel	83.15 (3.85)	77.89 (0.72)	80.91 (0.16)	88.72 (1.84)	47.96 (2.19)	88.09 (0.74)
	WeaSEL	66.32 (1.64)	76.38 (0.00)	81.05 (0.35)	86.60 (0.00)	27.60 (0.00)	54.38 (2.04)
	Denoise	**85.39** (0.10)	78.48 (0.02)	80.79 (0.24)	**97.60** (0.42)	56.32 (2.74)	88.41 (1.35)
	LO	-	-	-	-	-	-
	iWL	85.02 (0.14)	**81.40** (0.51)	**81.50** (0.34)	62.32 (2.04)	**59.80**(2.20)	**90.41** (0.74)
	iWLD	85.13 (0.13)	**81.11** (0.20)	**81.54** (0.29)	**96.84** (1.20)	**65.00** (1.71)	**91.17** (0.69)
1%	MV	85.05 (0.36)	80.59 (0.28)	81.52 (0.16)	95.40 (1.88)	62.88 (1.58)	89.67 (0.45)
	Snorkel	83.45 (3.78)	77.89 (0.73)	82.03 (0.28)	88.76 (1.93)	48.24 (1.82)	88.38 (0.43)
	WeaSEL	66.30 (2.10)	76.38 (0.00)	80.92 (0.63)	86.60 (0.00)	27.60 (0.00)	53.62 (1.34)
	Denoise	85.37 (0.09)	78.51 (0.11)	80.53 (0.21)	**97.72** (0.18)	56.44 (2.25)	87.64 (1.67)
	LO	87.39 (0.20)	80.41 (0.71)	81.98 (1.80)	96.84 (1.13)	51.76 (5.13)	91.34 (0.42)
	iWL	**87.75** (0.27)	**82.29** (0.67)	**84.25** (0.89)	95.88 (0.78)	**64.56** (2.98)	**92.43** (0.18)
	iWLD	**87.77** (0.31)	**82.82** (0.29)	**84.31** (0.84)	**97.44** (0.55)	**67.32** (1.95)	**92.21** (0.31)
10%	MV	85.65 (0.25)	81.03 (0.21)	82.05 (0.44)	96.76 (1.60)	64.28 (1.19)	90.63 (0.76)
	Snorkel	84.80 (2.36)	77.67 (0.94)	83.78 (0.73)	89.64 (2.60)	50.64 (3.73)	91.16 (1.03)
	WeaSEL	68.42 (0.49)	76.38 (0.00)	81.62 (0.36)	86.60 (0.00)	27.60 (0.00)	61.02 (8.33)
	Denoise	85.79 (0.12)	78.51 (0.07)	80.82 (0.20)	98.44 (0.26)	58.32 (2.88)	88.35 (1.48)
	LO	**89.94** (0.30)	**82.86** (0.39)	**86.63** (0.47)	**98.64** (0.46)	**72.52** (4.41)	93.91 (0.17)
	iWL	89.42 (0.22)	82.42 (0.33)	86.20 (0.47)	98.52 (0.27)	**72.92** (3.97)	**94.03** (0.16)
	iWLD	**90.00** (0.18)	**83.84** (0.30)	**86.53** (0.42)	**98.48** (0.39)	71.16 (5.72)	**94.06** (0.06)
50%	MV	88.17 (0.22)	81.59 (0.26)	84.82 (0.34)	98.44 (0.22)	69.72 (1.98)	93.18 (0.28)
	Snorkel	88.23 (0.24)	78.69 (1.01)	85.85 (0.21)	93.20 (2.84)	58.56 (6.41)	93.92 (0.25)
	WeaSEL	70.47 (2.17)	76.38 (0.00)	82.98 (0.32)	86.60 (0.00)	27.60 (0.00)	71.89 (4.14)
	Denoise	89.44 (0.11)	79.53 (0.13)	82.26 (0.31)	98.76 (0.17)	64.92 (1.83)	92.33 (0.32)
	LO	**90.60** (0.21)	**83.91** (0.27)	**87.56** (0.44)	**98.88** (0.30)	**80.24** (3.02)	**94.19** (0.13)
	iWL	89.60 (0.09)	82.66 (0.34)	86.62 (0.31)	**98.88** (0.39)	77.36 (2.60)	**94.51** (0.10)
	iWLD	**90.53** (0.17)	**84.55** (0.13)	**87.52** (0.43)	**98.80** (0.28)	**78.40**(2.58)	94.18 (0.21)
100%	MV	89.65 (0.09)	82.93 (0.30)	86.15 (0.21)	98.44 (0.09)	77.76 (0.89)	94.03 (0.16)
	Snorkel	89.05 (0.25)	80.97 (0.58)	85.64 (0.27)	96.60 (2.16)	64.76 (2.95)	93.74 (0.22)
	WeaSEL	72.72 (3.04)	76.38 (0.00)	84.20 (0.69)	86.60 (0.00)	27.60 (0.00)	84.43 (1.81)
	Denoise	**90.65** (0.14)	84.29 (0.11)	87.61 (0.11)	**98.80** (0.23)	81.88 (1.68)	94.17 (0.16)
	LO	**90.59** (0.34)	**84.40** (0.07)	**87.70** (0.24)	**98.64** (0.22)	**81.96** (1.11)	94.28 (0.18)
	iWL	89.67 (0.20)	82.61 (0.51)	86.73 (0.24)	**98.80** (0.24)	78.40 (2.19)	**94.56** (0.12)
	iWLD	90.46 (0.39)	**84.78** (0.20)	**87.74** (0.18)	98.52 (0.11)	**82.52** (1.25)	**94.47** (0.24)

Chapter 4

Generalization Gap in Amortized Inference

The ability of generative models trained with maximum likelihood to perform well on data unseen during training is central to the application of machine learning in many of the important problems we touched on in chapter 1 such as text and image generation. As discussed in section 2.1, we refer to this as the problem of generalization. One popular class of probabilistic generative model - the Variational Auto-Encoder (VAE) [98] - is an example of a latent variable model as introduced in chapter 2.1.1. As well as being an effective method to approximate the underlying generative process of datasets such as images and text, the lower dimensional representation of data encoded by the learned latent variables in VAEs are often used in the context of transfer learning. That is, the latent representations are often used as features for downstream tasks such as classification or compression. One reliable way to improve the generalization performance in this setting is to add more training data.

In this chapter, following the theme of data efficiency, we study the question of how to improve generalization performance for this class of model for a given training data budget?

We first study the problem of generalization for this class of probabilistic latent variable model. We decompose the generalization error into two generalization gaps that affect VAEs and demonstrate that over-fitting can be

dominated by the process of amortized inference using neural networks. Based on this observation, we propose a new training scheme that improves the generalization of amortized inference by bootstrapping samples from the model when training the inference network. We demonstrate how our method can improve performance in the context of image modelling and lossless compression.

4.1 Overfitting to training data

To recap briefly, for a finite dataset, a common concern in both supervised and unsupervised learning is that the probabilistic model may overfit to the training dataset $\mathcal{X}_{train}$, degrading generalization performance [99]. The generalization performance in the unsupervised setting can be measured by the test likelihood [100]: $\frac{1}{M}\sum_{n=1}^{M}\log p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}}^m)$, where $\mathcal{X}_{test} = \{\check{\boldsymbol{x}}^1, \ldots, \check{\boldsymbol{x}}^M\} \sim p_d(\mathbf{x})$ is the test dataset. A model that has overfit to the training dataset $\mathcal{X}_{train}$ generally results in a high training likelihood but a low test likelihood. As discussed in section 2.1 the use of neural networks to parameterise our distribution can amplify this problem given their capacity to model the noise and spurious correlations that may be present in the training data.

Although the test likelihood is a common evaluation criterion [101], the factors that affect the generalization of unsupervised probabilistic models are less well studied in comparison to supervised learning. We posit that this is because for common tasks, like sample generation or representation learning, good generalization in terms of the test likelihood is not a sufficient measure of performance. For example implicit models can generate sharp samples without having a likelihood function [102, 103, 48] and representations learned by latent variable models can be arbitrarily transformed without changing the likelihood [81]. In recent applications that use deep generative models for lossless compression [104, 105, 106, 100, 107], generalization in terms of the test likelihood directly indicates higher compression rate [100]. Specifically, given a probabilistic model $p_{\boldsymbol{\theta}}(\mathbf{x})$, a lossless compressor can be constructed to compress a test data point $\check{\boldsymbol{x}}$ to a bit string with length approximately equal

to $-\log_2 p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}})$. When $p_{\boldsymbol{\theta}}(\mathbf{x}) \to p_d(\mathbf{x})$, the average compression length attains the entropy of the data distribution $-\frac{1}{M}\sum_{m=1}^{M}\log_2 p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}}^m) \to \mathcal{H}_p(\mathbf{x})$, which is optimal under Shannon's source coding theorem [108], see chapter 4 of [51] for a detailed introduction. Therefore, a better test likelihood can lead directly to a greater saving in bits.

4.2 Variational auto-encoders

A popular type of probabilistic model is the Variational auto-encoder (VAE) [59, 60] which assumes a latent variable model $p_{\boldsymbol{\theta}}(x) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$. For a nonlinear parameterization of $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ (e.g. a deep neural network), the evaluation of $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ involves solving an intractable integration over $\mathbf{z}$. In this case, the evidence lower bound (ELBO) can be used to side-step the intractability as introduced in section 2.1.1

$$\mathbb{E}_{p_d(\mathbf{x})}\Big(\log p_{\boldsymbol{\theta}}(\mathbf{x})\Big) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})p_d(\mathbf{x})}\Big(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathrm{z}|\mathbf{x})\Big) \tag{4.1}$$

$$\equiv \mathbb{E}_{p_d(\mathbf{x})}\Big(\mathrm{ELBO}(\mathbf{x},\boldsymbol{\theta},\boldsymbol{\phi})\Big), \tag{4.2}$$

where $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is a variational posterior parameterized by a neural network with parameter $\boldsymbol{\phi}$. The use of an approximate posterior of the form $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ amortizes the cost of inference across the dataset. To better understand this objective, we can rewrite the expected ELBO as the following

$$\mathbb{E}_{p_d(\mathbf{x})}\Big(\mathrm{ELBO}(\mathbf{x},\boldsymbol{\theta},\boldsymbol{\phi})\Big) = \mathbb{E}_{p_d(\mathbf{x})}\Big(\log p_{\boldsymbol{\theta}}(\mathbf{x}) - \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\Big)\Big) \tag{4.3}$$

$$= -\underbrace{\mathcal{H}_{p_d}(\mathbf{x})}_{\text{constant}} - \underbrace{\mathrm{KL}(p_d(\mathbf{x}))||p_{\boldsymbol{\theta}}(\mathbf{x}))}_{\text{model learning}} \tag{4.4}$$

$$-\underbrace{\mathbb{E}_{p_d(\mathbf{x})}\Big(\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big)\Big)}_{\text{amortized inference}}, \tag{4.5}$$

We denote the posterior family of $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ as $\mathcal{Q}$ parameterized by $\boldsymbol{\phi}$ [109]. If $\mathcal{Q}$ is flexible enough such that the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}$, where $p_{\boldsymbol{\theta}}(\mathrm{z}|\mathrm{x}) \propto p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})(\mathbf{z})$, then at the optimum of equation 4.3, we have $\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big) = 0 \Rightarrow q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ and the ELBO will be equal to

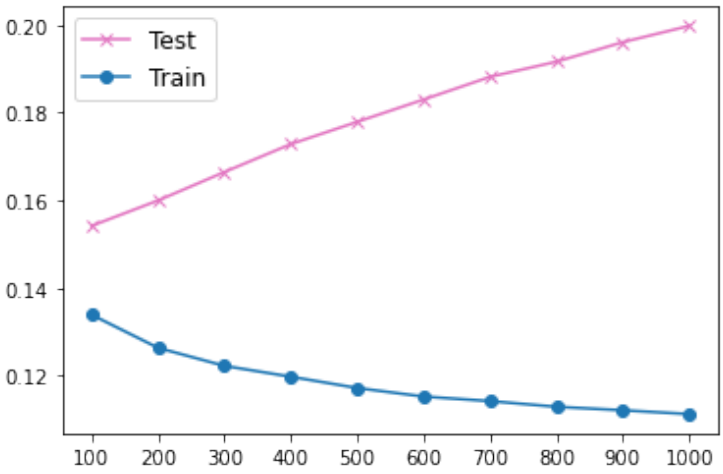


Figure 4.1: Bits per dimension (BDP) vs epochs. The training BPD decreases but the testing BPD increases during training, which indicates the VAE overfits to $\mathcal{X}_{train}$.

the log-likelihood $\text{ELBO}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \log p_{\boldsymbol{\theta}}(\mathbf{x})$ [59, 110]. Many methods have been developed to increase the flexibility of $\mathcal{Q}$ to obtain a tighter bound. For example adding auxiliary variables [111, 112], or flow-based methods [113, 114].

4.3 Generalization of VAEs

During training, we only have access to a finite dataset $\mathcal{X}_{train}$, which leads to the following Monte-Carlo approximation:

$$\mathbb{E}_{p_d(\mathbf{x})}\Big(\text{ELBO}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi})\Big) \approx \frac{1}{N}\sum_{n=1}^{N} \text{ELBO}(\boldsymbol{x}^n, \boldsymbol{\theta}, \boldsymbol{\phi}). \tag{4.6}$$

This empirical approximation to the true ELBO can lead to the VAE overfitting to the training data. To help illustrate this, we train a VAE on the binary MNIST dataset for 1000 epochs and plot the Bits-Per-Dimension (BPD) of both the training and testing dataset at every 100 epochs. In this case the BPD is defined as the the negative ELBO (with a base 2 logarithm) normalized by the data dimension. Lower BPD indicates higher ELBO. We provide more specifics on the training and model setup in section 4.5. Figure 4.1 plots the training and testing BPD during training, which demonstrates that the VAE model is overfitting to the training dataset. We now focus on understanding

which components of the model are contributing to this behaviour in order to adapt the training process to improve generalization performance for the same amount of training data. If we take the the decomposition in equation 4.5, we see there are two empirical approximations contributing to the ELBO.

Firstly, an empirical approximation related to the model term:

$$\mathrm{KL}(p_d(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) \approx \frac{1}{N}\sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^n) + \text{const.} \tag{4.7}$$

Secondly there is an empirical approximation related to the amortised inference term, which is the expected KL:

$$\mathbb{E}_{p_d(\mathbf{x})}\Big(\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big)\Big) \approx \frac{1}{N}\sum_{n=1}^{N} \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}^n)||p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}^n)\Big). \tag{4.8}$$

For flexible enough parameterizations of $p_{\boldsymbol{\theta}}(x)$ and $q_{\boldsymbol{\phi}}(z|x)$ (such as those parameterized by deep neural networks) there is the risk of overfitting to the training data for both elements. Focusing on the amortized inference term; we define $\hat{\boldsymbol{\phi}}$ to be the optimal parameter of the empirical variational inference objective

$$\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}} \frac{1}{N}\sum_{n=1}^{N} \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}^n)||p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}^n)\Big) \tag{4.9}$$

and we assume for any training data point $\boldsymbol{x}^n \in \mathcal{X}_{train}$

$$q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\boldsymbol{x}^n) = \arg\min_{q\in\mathcal{Q}} \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}^n)||p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}^n)\Big) \equiv q^*(\mathbf{z}|\boldsymbol{x}^n),$$

where $q^*(\mathbf{z}|\boldsymbol{x}^n)$ is the true optimal posterior within the $\mathcal{Q}$ family for datapoint $\boldsymbol{x}^n$. Here for simplicity we have assumed that, for a powerful enough inference neural network, there is no amortization gap [115], which means $q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\boldsymbol{x})$ can produce the optimal $q^*(\mathbf{z}|\boldsymbol{x}^n)$ for any training datum $\boldsymbol{x}^n \in \mathcal{X}_{train}$[1]. If $q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\boldsymbol{x}^n)$ overfits to a limited $\mathcal{X}_{train}$, $q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\breve{\boldsymbol{x}}^m)$ fails to be a good approximation to the

[1] See section 4.5.3 for empirical analysis of the tightness of the ELBO

true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\check{\boldsymbol{x}}^m)$ for test data $\check{\boldsymbol{x}}^m \in \mathcal{X}_{test}$.

We refer to the difference between the ELBO evaluated using $q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\mathbf{x})$ and the ELBO evaluated using $q^*(\mathbf{z}|\mathbf{x})$ as the Amortized Inference Generalization Gap (AIGG), defined as

$$\text{AIGG}(\mathbf{x}, \hat{\boldsymbol{\phi}}, q^*) = \mathbb{E}_{p_d(\mathbf{x})}\Big(\text{KL}\Big(q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big) - \text{KL}(q^*(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))\Big), \tag{4.10}$$

with the expectation w.r.t to the data distribution $p_d(x)$ and where $q_{\hat{\boldsymbol{\phi}}}$ is the optimal distribution for the training data. Equivalently, this gap can be written as the difference between two ELBOs calculated with different q

$$= \mathbb{E}_{p_d(\mathbf{x})}\Big(\underbrace{\mathbb{E}_{q^*(\mathbf{z}|\mathbf{x})}\Big(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q^*(\mathbf{z}|\mathbf{x})\Big)}_{\text{ELBO with optimal inference}} - \underbrace{\mathbb{E}_{q_{\hat{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}}\Big(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\mathbf{x})\Big)}_{\text{ELBO with amortized inference}}\Big). \tag{4.11}$$

The inference neural network introduced by amortization is the cause of this inference generalization gap. It is important to emphasize that this gap cannot be reduced by simply using a more flexible $\mathcal{Q}$. This would only make $\text{KL}\Big((q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}^n)||p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}^n)\Big)$ smaller for the training data $\boldsymbol{x}^n \in \mathcal{X}_{train}$ but would not explicitly encourage better generalization performance on test data [116].

To summarize, the generalization performance of a VAE depends on two factors:

- **Generative model generalization gap (GMGG):** defined in equation 4.7, caused by the generative model overfitting to the training data.

- **Amortized inference generalization gap (AIGG):** defined in equation 4.11, caused by the amortized inference model overfitting to the the training data.

4.3.1 Impact of the generalization gaps

Now we can consider how the two different generalization gaps outlined in the previous section contribute to the overall degradation in performance due to overfitting.

The generative model generalization gap (GMGG) estimate for the test dataset $\mathcal{X}_{test}$;

$$\mathrm{KL}(p_d(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) \approx -\frac{1}{M}\sum_{m=1}^{M}\log p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}}^m) + const, \tag{4.12}$$

cannot be calculated explicitly since we can only evaluate the lower bound $-\frac{1}{M}\sum_{m=1}^{M}\mathrm{ELBO}(\check{\boldsymbol{x}}^m, \boldsymbol{\theta}, \boldsymbol{\phi})$. Fortunately, using equation 4.3, if we know the optimal posterior for the test data $q^*(\mathbf{z}|\check{\boldsymbol{x}}^m) \equiv \arg\min_{q\in\mathcal{Q}}\mathrm{KL}(q(\mathbf{z}|\check{\boldsymbol{x}}^m)||p_{\boldsymbol{\theta}}(\mathbf{z}|\check{\boldsymbol{x}}^m))$, the log-likelihood can be approximated by the lower bound $\log p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}}^m) \approx \mathrm{ELBO}(\check{\boldsymbol{x}}^m, \boldsymbol{\theta}, \boldsymbol{\phi})$, which becomes an equality when $p_{\boldsymbol{\theta}}(\mathbf{z}|\check{\boldsymbol{x}}^m) \in \mathcal{Q}$ (assuming Q is flexible enough).

The AIGG can be estimated for the test dataset $\mathcal{X}_{test}$ by knowing the optimal posterior $q^*(\mathbf{z}|\check{\boldsymbol{x}}^m)$:

$$\mathrm{AIGG}(\mathbf{x}, \hat{\boldsymbol{\phi}}, q^*) \approx \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{q^*(\mathbf{z}|\check{\boldsymbol{x}}^m)}\Big(\log p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}}^m, \mathbf{z}) - \log q^*(\mathbf{z}|\check{\boldsymbol{x}}^m)\Big) \tag{4.13}$$

$$- \mathbb{E}_{q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\check{\boldsymbol{x}}^m)}\Big(\log p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}}^m, \boldsymbol{z}) - \log q_{\hat{\boldsymbol{\phi}}}(\mathbf{z}|\check{\boldsymbol{x}}^m)\Big). \tag{4.14}$$

We can estimate $q^*(\mathbf{z}|\check{\boldsymbol{x}}^m)$ by fixing $\boldsymbol{\theta}$ (which is learned on the training dataset) and learning $\boldsymbol{\phi}^*$ on the test dataset and further assume for simplicity that for an expressive enough inference network $q^*(\mathbf{z}|\check{\boldsymbol{x}}^m) = q_{\boldsymbol{\phi}^*}(\mathbf{z}|\check{\boldsymbol{x}}^m)$, where

$$\boldsymbol{\phi}^* = \min_{\boldsymbol{\phi}}\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\check{\boldsymbol{x}}^m)||p_{\boldsymbol{\theta}}(\mathbf{z}|\check{\boldsymbol{x}}^m)\Big) \tag{4.15}$$

$$= \max_{\boldsymbol{\phi}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\check{\boldsymbol{x}}^m)}\Big(\log p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}}^m, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\check{\boldsymbol{x}}^m)\Big). \tag{4.16}$$

We refer to this as the optimal inference strategy, given we are using the test dataset directly to learn the appropriate inference network. Using this optimal

inference strategy we can in principle eliminate the effect of the AIGG, allowing us to isolate the degree to which both the GMGG and AIGG are contributing to overfitting.

To help illustrate the contribution of GMGG and AIGG, we take the VAE trained on binary MNIST that was used to demonstrate overfitting in 4.1 from the previous section and we continue training using the optimal inference strategy as follows:

We train $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, keeping $\boldsymbol{\theta}$ fixed, for an additional 1000 epochs on the test data using equation 4.16 to obtain the test BPD for the optimal inference strategy. In figure 4.2 we overlay the test ELBO (BPD) using the optimal inference strategy at every 100 epochs during the subsequent training (green) with the existing test (pink) and train (blue) BPDs. For the optimal inference strategy the average likelihood $\frac{1}{M}\sum_{n=1}^{M}\log p_{\boldsymbol{\theta}}(\check{\boldsymbol{x}}^m)$ can be effectively approximated by the ELBO, therefore the difference between the two inference curves on the test set (test (pink) and optimal (green)) is the AIGG. We observe that after eliminating the AIGG using the optimal inference strategy, the test BPD (green) is stable with a marginal increase during training for this problem. This suggests that the generative model $p_{\boldsymbol{\theta}}(\mathbf{x})$ slightly overfitting to the data but that the overfitting is actually dominated by the amortized inference network.

Although the optimal inference strategy can help eliminate the inference generalization gap, training $q_{\boldsymbol{\phi}}$ on the test data is not practical in most applications of interest. Therefore, we now turn our attention to improving the generalization of amortized inference without access to the test data at training time and without leveraging any additional true training data.

4.4 Consistent amortized inference

We first propose an inference consistency requirement which, if satisfied, would result in optimal generalization performance for amortized variational inference, before then introducing a way to improve generalization performance by reducing AIGG.

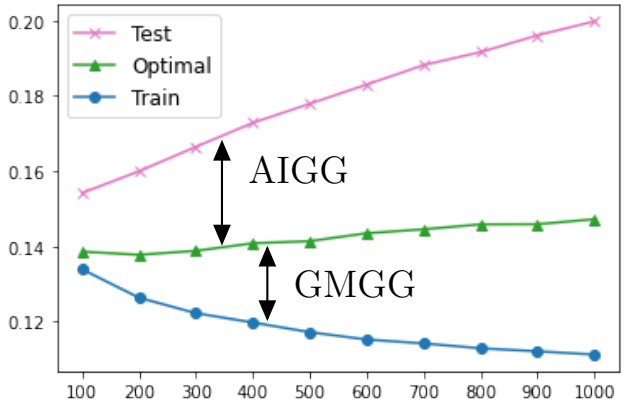


Figure 4.2: BDPs vs epochs. Visualization of the GMGG and AIGG.

When $p_{\boldsymbol{\theta}} \to p_d$, the amortized posterior should converge to the true posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \to p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ for every $\boldsymbol{x} \sim p_d(\mathbf{x})$ (if we assume that the true posterior belongs to the variational family $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}$). Although this requirement seems natural for variational inference, the classic amortized inference training used for VAEs does not satisfy it [59]. Recall the empirical ELBO objective

$$\frac{1}{N}\sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^n) - \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathrm{z}|\boldsymbol{x}^n)||p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}^n)\Big). \tag{4.17}$$

When the model converges to the true distribution $p_{\boldsymbol{\theta}^*} = p_d$ the training criterion for $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x})$

$$\min_{\boldsymbol{\phi}} -\frac{1}{N}\sum_{n=1}^{N} \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}^n)||p_{\boldsymbol{\theta}^*}(\mathbf{z}|\boldsymbol{x}^n)\Big) \tag{4.18}$$

can still result in the amortized posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x})$ overfitting to the training data. In principle, one could also limit the network capacity and/or add an explicit regularizer to the parameters [99] in an attempt to improve the generalization. However, this still will satisfy the consistency requirement in principle because it only uses the finite training dataset. Alternatively, there is another classic variational inference method, the wake-sleep training algorithm [117, 118], which does in fact satisfy the proposed consistency requirement.

4.4.1 Wake-sleep training

Defining $q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z}) = q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})p_d(\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})$, the two phases of wake-sleep training [117, 118] can be written as minimizing two different KL divergences in both $\mathbf{x}$ and $\mathbf{z}$ space.

- **Wake phase model learning:** $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is trained by minimizing the KL divergence

$$\min_{\boldsymbol{\theta}} \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})\Big) = \max_{\boldsymbol{\theta}} \mathbb{E}_{p_d(\mathbf{x})}\Big(\mathrm{ELBO}(\mathbf{x},\boldsymbol{\theta},\boldsymbol{\phi})\Big) + \mathrm{const}, \tag{4.19}$$

  where the expectation $\mathbb{E}_{p_d(x)}$ is approximated using the training data. This is referred to as the wake phase since the model is trained on experience from the so called real environment, i.e. it uses true data samples from $p_d(\mathbf{x})$.

- **Sleep phase amortized inference:** $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is trained by minimizing the KL divergence

$$\min_{\boldsymbol{\phi}} \mathrm{KL}\Big(p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})||q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})\Big) = \min_{\boldsymbol{\phi}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}\Big(\mathrm{KL}\Big(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})||q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\Big)\Big) + \mathrm{const}. \tag{4.20}$$

  Leaving out the terms that are irrelevant to $\boldsymbol{\phi}$, the objective can be estimated with Monte-Carlo

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}\Big(-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\check{\boldsymbol{x}}^m)\Big) \approx \frac{1}{K}\sum_{k=1}^{K} -\log q_{\boldsymbol{\phi}}(\boldsymbol{z}^k|\boldsymbol{x}^k), \tag{4.21}$$

  where $\boldsymbol{z}^k \sim p(\mathbf{z})$ and $\boldsymbol{x}^k \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}^k)$. This is referred to as the sleep phase because the samples from the model used to train $q_{\boldsymbol{\phi}}$ are interpreted as dreamed experience.

In contrast, the training criterion for the typical VAE amortized inference (equation 4.8) uses the true data samples from p_d to train $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, which we

refer to as wake phase amortized inference. If a perfect model $p_{\boldsymbol{\theta}^*}(\mathbf{x}) = p_d(\mathbf{x})$ were used for sleep phase amortized inference, then it is equivalent to minimizing

$$\mathbb{E}_{p_{\boldsymbol{\theta}^*}(\mathbf{x})}\Big(\mathrm{KL}\Big(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})||q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\Big)\Big) = \mathbb{E}_{p_d(\mathbf{x})}\mathrm{KL}\Big(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})||q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\Big)\Big). \tag{4.22}$$

In this set up, the inference network training satisfies the inference consistency requirement since we can access infinite training data from p_d by sampling from $p_{\boldsymbol{\theta}^*}$.

However, the wake-sleep algorithm presented lacks convergence guarantees [117] and minimizing $\mathrm{KL}\Big(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})||q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\Big)$ in the sleep phase doesn't necessarily encourage an improvement to the ELBO. Therefore, in the next section, we propose a different variational inference scheme: reverse sleep amortized inference, and demonstrate how it helps improve the generalization of the inference network in practice.

4.4.2 Reverse sleep amortized inference

We now propose to instead use the reverse KL divergence in the sleep phase when updating the inference network parameters. Specifically we consider the scheme where we fix $\boldsymbol{\theta}$ at the end of normal VAE training and then continue to train $\boldsymbol{\phi}$. This has the practical advantage of being a simple bolt on algorithm to the existing training process. Our objective is

$$\min_{\boldsymbol{\phi}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}\Big(\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big)\Big) = \max_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})p_{\boldsymbol{\theta}}(\mathbf{x})}\Big(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\Big), \tag{4.23}$$

where the integration $\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}$ is approximated by Monte-Carlo using samples from the generative model $p_{\boldsymbol{\theta}}(\mathbf{x})$. Unlike the vanilla wake-sleep protocol, this reverse KL objective encourages improvements to the ELBO. In the case of a perfect model $p_{\boldsymbol{\theta}^*}(\mathbf{x}) = p_d(\mathbf{x})$ the reverse sleep phase is equivalent to

$$\min_{\boldsymbol{\phi}} \mathbb{E}_{p_{\boldsymbol{\theta}^*}(\mathbf{x})}\Big(\mathrm{KL}\Big(p_{\boldsymbol{\theta}^*}(\mathbf{z}|\mathbf{x})||q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\Big)\Big) = \min_{\boldsymbol{\phi}} \mathbb{E}_{p_d(\mathbf{x})}\Big(\mathrm{KL}\Big(p_{\boldsymbol{\theta}^*}(\mathbf{z}|\mathbf{x})||q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\Big)\Big) \tag{4.24}$$

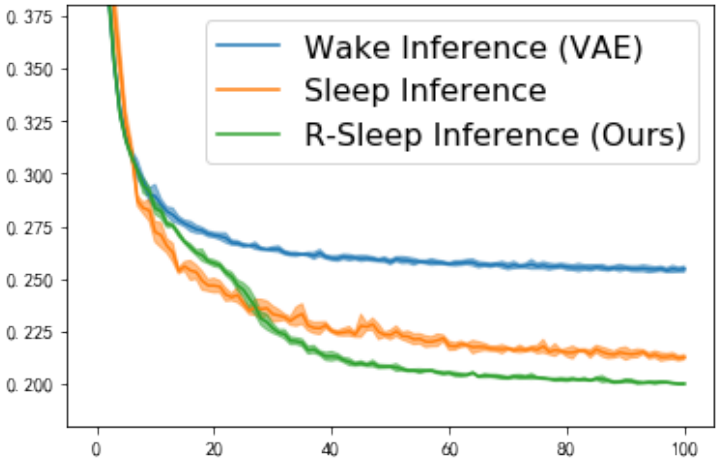


Figure 4.3: Test BPD vs epochs. We compare the consistency property between three amortized inference methods.

which indeed satisfies the inference consistency requirement.

We now illustrate the consistency requirement in the case of a perfect model. This can be achieved by using a pretrained VAE as the true data generation distribution $p_d(\mathbf{x})$ to emulate the perfect model. We first train a VAE to fit the binary MNIST problem. The VAE has the same structure as that used in section 4.3.1 and is trained for 100 epochs (see section 4.5 for more details on the model and training setup). After training, we treat the pre-trained $p_{\hat{\boldsymbol{\theta}}}(\mathbf{x}|\mathbf{z})$ as the true data distribution $p_d(x) \equiv \int p_{\hat{\boldsymbol{\theta}}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$. We sample 10000 data samples from p_d to form a training set $\mathcal{X}_{train}$ and 1000 samples to form a test set $\mathcal{X}_{test}$. We then train a new $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ with: wake phase inference (vanilla VAE training), sleep inference (from the normal wake-sleep setup and our reverse sleep inference and compare the results. Figure 4.3 shows the test BPD calculated after every training epoch. We can see sleep phase inference out-performs wake phase inference and that reverse sleep inference achieves the best BPD. Intuitively, given how we have constructed p_d, this is due to both the forward and reverse sleep inference using the true model to generate additional training data whereas the wake inference only has access to the finite training dataset X_{train}.

4.4.3 With imperfect models

In practice our model will not be perfect $p_{\boldsymbol{\theta}} \neq p_d$. Empirically we find that samples from even a well trained model $p_{\boldsymbol{\theta}}$ may not always be sufficiently like the samples from the true data distribution. This can lead to degradation in the performance of the inference network when using the reverse-sleep approach presented in the previous section. In practise, we propose to use a mixture distribution between the model and the empirical training data distribution as follows

$$\mathbb{E}_{m(\mathbf{x})}\Big(\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big)\Big) \quad \text{where} \quad m(\mathbf{x}) \equiv \alpha p_{\boldsymbol{\theta}}(\mathbf{x}) + (1-\alpha)\hat{p}_d(\mathbf{x}). \tag{4.25}$$

When $\alpha = 0$, it reduces to the standard approach used in VAE training. When $\alpha = 1$, we recover the reverse sleep method (equation 4.23). Although this introduces another hyper-parameter into the training scheme, we find empirically that a setting of $\alpha = 0.5$ works well in practice. This balances samples from the true underlying data distribution with samples from the model.

We refer to this method as reverse half-sleep since it uses both data and model samples to train the amortized posterior. We can rewrite the equation 4.25 as a sum of two positive terms

$$\alpha\mathbb{E}_{\hat{p}_d(\mathbf{x})}\Big(\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big)\Big) + (1-\alpha)\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}\Big(\mathrm{KL}\Big(\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big)\Big). \tag{4.26}$$

The optimum of this objective will make the first term 0, which is the same requirement as the classic amortized inference (equation 4.8). The second term, which is equivalent to the reverse sleep amortized inference (equation 4.23), encourages the inference consistency requirement: when $p_{\boldsymbol{\theta}} = p_d$, the optimum of the second term will set $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ for any $\boldsymbol{x} \sim p_d(\mathbf{x})$. When $p_{\boldsymbol{\theta}}$ is not perfect, the second term can be seen as a regularizer of the classic amortized inference objective that penalizes the hypothesis space of the amortization

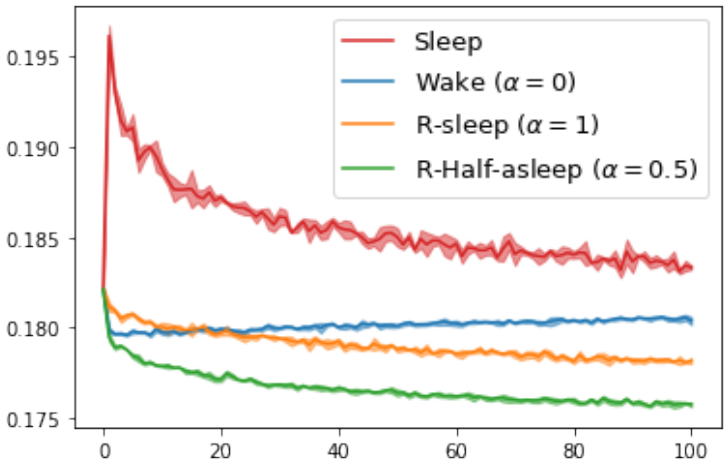


Figure 4.4: Test BPD comparisons of amortized inference with different α. We find the reverse half sleep method ($\alpha = 0.5$) achieves the best BPD. The mean and std are calculated with three random seeds.

neural network [99].

To illustrate the impact of different settings of α we again fit a VAE to the binary MNIST dataset, freeze $\boldsymbol{\theta}$, and then continue to train the amortized posterior for an additional 100 epochs using sleep inference (equation 4.20) and reverse half-sleep inference for three different α values. Figure 4.4 shows the test BPD comparison. We find the proposed reverse half-sleep method with $\alpha = 0.5$ outperforms the reversed sleep method ($\alpha = 1$), whereas the standard amortized inference training in VAE ($\alpha = 0$) leads to overfitting of the inference network. We also plot the sleep inference training curve, whose BPD is expected to be less competitive since it is not directly optimizing the ELBO.

4.5 Experiments

Up to this point we have included illustrations of the generalization gaps and our application of reverse sleep inference method using a simple running example of a basic VAE model trained on the binary MNIST dataset. The architecture for this model consists of feedback forward networks for the model and inference networks; each with 2 layers of 500 hidden units and a latent variable dimension of 16. We used ADAM [66] with a learning rate of 5×10^{-4}, which we found to be a stable choice across the different variations.

In this section we expand our empirical analysis to a broader range of problems to demonstrate the utility of our method for improving the generalization performance and data efficiency. We first replicate the VAE image modelling experiment introduced in the previous sections on more challenging datasets. Next we apply our method in a transfer learning setting and demonstrate how it benefits the performance of downstream classification. Finally we run some ablation experiments on the tightness of the ELBO, how the latent dimensionality impacts performance and the impact of applying our reverse sleep training procedure from the start of the original model training as opposed to only as a post-hoc training procedure. In appendix B we also present an application where VAEs are used as the generative model for lossless compression.

4.5.1 Image modelling

We apply the reverse half-sleep to improve the generalization of VAEs on three different datasets: binary MNIST, grey MNIST [119] and CIFAR10 [120].

Binary and Grey MNIST

We use a latent dimension of 16 and 32 respectively, and neural nets with 2 layers of 500 hidden units in both the model and inference neural networks. We use Bernoulli $p(\mathbf{x}|\mathbf{z})$ in binary MNIST and discretized logistic distribution for grey MNIST for the likelihood.

We first train the VAE with the usual amortized inference approach using Adam with $lr = 3\times10^{-4}$ for 1000 epochs and save the model every 100 epochs. We then use the saved models to train $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ on 1) the test data using the optimal inference and 2) using our reverse half-sleep method. This produces 2 different models we can evaluate against the usually trained VAE on the test data at each 100 epoch checkpoint.

For our reverse half-sleep training, we train $\boldsymbol{\phi}$ keeping $\boldsymbol{\theta}$ fixed for 100 epochs using the same optimization settings from the original VAE training. To sample from $p_{\boldsymbol{\theta}}(\mathbf{x})$, we first sample $\boldsymbol{z}' \sim p(\mathbf{z})$ and then sample $\boldsymbol{x}' \sim p(\mathbf{x}|\boldsymbol{z}')$.

For the optimal inference strategy, we train $\boldsymbol{\phi}$ keeping $\boldsymbol{\theta}$ fixed on the test

data set for an additional 500 epochs to ensure the same number of gradient steps are conducted (since the training set is 5 times as big as the test set). Figure 4.5a and 4.5b show that our approach does not require further training on the test data to improve generalization performance in terms of test BPD.

CIFAR10

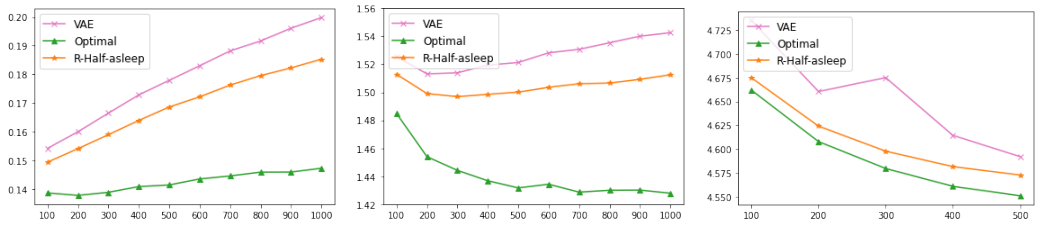
We use the convolutional ResNet architecture that we introduced in section 2.2 [8, 121] with 2 residual blocks and latent variable size of 128. The likelihood is a discretized logistic distribution with linear autoregressive parameterization within the channels. We train the original VAE for 500 epochs with Adam and $lr = 5\times10^{-4}$ and save the model every 100 epochs. This pre-trained VAE achieves 4.592 test BPD for CIFAR10[2].

Ideally, when the VAE model converges to the true distribution $p_{\boldsymbol{\theta}} \rightarrow p_d$, the aggregate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}) = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})p_d(\mathbf{x})d\mathbf{x}$ will match the prior $p(\mathbf{z})$. However, for a complex distribution like CIFAR10, a significant mismatch between $q_{\boldsymbol{\phi}}(\mathbf{z})$ and $p(\mathbf{z})$ is usually observed in practice [122, 123]. In this case, a sample $\boldsymbol{x'}$ that we generate using a latent sample from the prior $\boldsymbol{x'} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z'})$, where $\boldsymbol{z'} \sim p(\mathbf{z})$, may be blurry or invalid. A common solution is to train another model, e.g. a VAE [123] or a PixelCNN [124, 121] to approximate $q_{\boldsymbol{\phi}}(\mathbf{z})$. We decide to instead directly sample from $q_{\boldsymbol{\phi}}(\mathbf{z})$ rather than $p(\mathbf{z})$ to generate samples when updating $\boldsymbol{\phi}$ using our reverse sleep approach (equation 4.23), which can be done by first sampling $\boldsymbol{x'} \sim p_d(\mathbf{x})$ (from the training dataset) and then sample $\boldsymbol{z'} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x'})$. This scheme still results in a consistent training objective since $q_{\boldsymbol{\phi}^*}(\mathbf{z}) = p(\mathbf{z})$ for the optimal posterior $q_{\boldsymbol{\phi}^*}(\mathbf{z}|\mathbf{x})$. As before, for our reverse half-sleep training, we train $\boldsymbol{\phi}$ keeping $\boldsymbol{\theta}$ fixed for 100 epochs using the same optimization settings from the original VAE training and for the optimal inference strategy, we train on the test data set for an additional 500 epochs. In figure 4.5c we find the proposed reverse half-sleep approach (with sampling from $q_{\boldsymbol{\phi}}(\mathbf{z})$) consistently improves the generalization performance of

[2]This performance is comparable with other single latent VAE models reported in [121] - 4.51 BPD with a VAE with latent dimension 256 and 4.67 BPD with a discrete latent VAE (VQVAE)

the amortized posterior for the same budget of training data.

Since the model parameters $\boldsymbol{\theta}$ are shared and fixed in all comparisons, better test ELBO indicates the predicted $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ is closer to the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\mathbf{x}')$ under the KL divergence measure (see equation 4.3).



(a) Binary MNIST **(b)** Grey MNIST **(c)** CIFAR10

Figure 4.5: Test BPD comparisons among amortized inference (VAE), optimal inference strategy and the reverse half-sleep inference on three datasets. The x-axis represents the training epochs.

4.5.2 Down-stream classification tasks

One common use of VAEs is in transfer learning where we use the learned amortized posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ for down stream tasks. For example image classification where the samples $\boldsymbol{z}' \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ can be treated as a latent stochastic representation [125, 126] of a given data point $\boldsymbol{x}'$ that can be used as a lower dimensional feature representation. Given a labeled dataset $\{(\boldsymbol{x}^1, y^1), \cdots, (\boldsymbol{x}^N, y^N)\}$ and a pretrained amortized posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ we can train a classifier $p_{\boldsymbol{\eta}}(\mathrm{y}|\mathbf{z})$ that maps from the latent space $\mathbf{z}$ to the label y.

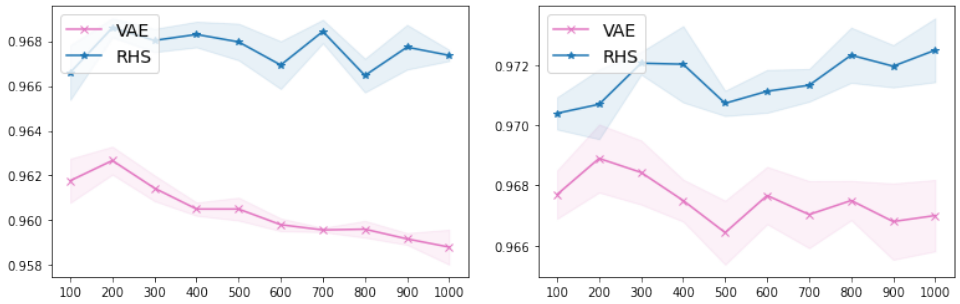
After training the classifier, for a given test set of unlabelled data $\{\check{\boldsymbol{x}}^1, \cdots, \check{\boldsymbol{x}}^M\}$, the predictive distribution can be written as

$$p(\mathrm{y}|\mathbf{x}) = \int p_{\boldsymbol{\eta}}(\mathrm{y}|\mathbf{z}) q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) d\mathbf{z} \tag{4.27}$$

and can be approximated using a Monte-Carlo estimate $p(\mathrm{y}|\mathbf{x}) \approx \frac{1}{K} \sum_{k=1}^{K} p(\mathrm{y}|\boldsymbol{z}^k)$, where $\boldsymbol{z}^k \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

We train a classifier with a 2 layer feed-forward neural network with hidden size 200, RelU activation and dropout with rate 0.1 on our binary and gret MINST datasets. The models are trained for 10 epochs with Adam optimizer

and learning rate 3×10^{-4}. During training, we randomly sample one $\boldsymbol{z}^k$ for each data point $\boldsymbol{x}$ and we use $k=100$ in the testing stage to estimate the predictive distribution. Figure 4.6 compares the posterior trained by the classic amortized inference approach and our proposed reverse half-sleep method. We find that our method consistently improves the classification accuracy performance for the same training data budget.



(a) Binary MNIST **(b)** Grey MNIST

Figure 4.6: Representation learning for down-stream classification. We train the VAE for 1000 epochs and evaluate the classification accuracy (y-axis, higher is better) on the down-stream classification task every 100 epochs (x-axis). The results are averaged over 3 random seeds and we also plot the standard deviation. VAE in the legend refers to typical VAE training, whereas RHS refers to our reverse-half-sleep method.

4.5.3 Ablation studies

Comparisons with regularization methods

Recent related work [116] proposed to alleviate the overfitting of amortized inference by optimizing a linear combination between the traditional amortized inference (equation 4.8) and a denoising objective

$$\alpha\mathbb{E}_{p(\epsilon)}\Big(\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}+\epsilon)||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big)\Big)+(1-\alpha)\mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big), \tag{4.28}$$

where $p(\epsilon)=\mathcal{N}(0,\sigma^2 I)$. We compare this regularizer to our method by training the pre-trained $\boldsymbol{\phi}$, keeping θ fixed, for an additional 100, 300 and 100 epochs on Binary, Grey MNSIT and CIFAR respectively using both methods.

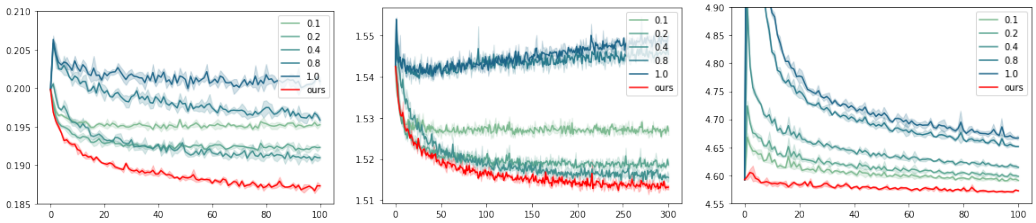
For the denoising regularizer, we use the same linear combination weight $\alpha=0.5$ as we use in equation 4.25 and vary $\sigma\in\{0.1,0.2,0.4,0.6,0.8,1.0\}$. See

table 4.1 for the comparisons. For MNIST, we find $\sigma \in \{0.1, 0.2, 0.4\}$ improves generalization performance, but larger noise levels hurts the performance.

For CIFAR10, only $\sigma = 0.1$ can slightly improve the generalization. In contrast, our method consistently achieves better generalization performance without tuning any hyper-parameters (given we are using a fixed value for $\alpha = 0.5$, which could in principle also be tuned). See figure 4.7 for the test BPD (evaluated every training epoch with the mean and standard deviation being calculated with 3 random seeds). One drawback of our method compared to the denoising regularizer approach is the requirement for model samples, which is more computational expensive during training.

Table 4.1: Average test BPD comparisons with Denoising Regularizer [116].

Methods	VAE	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.4$	$\sigma = 0.8$	$\sigma = 1.0$	Ours
Binary MNIST	0.200	0.195	0.192	0.191	0.196	0.201	**0.187**
Grey MNIST	1.543	1.527	1.519	1.515	1.545	1.550	**1.513**
CIFAR10	4.592	4.591	4.598	4.614	4.651	4.667	**4.572**



(a) Binary MNIST **(b)** Grey MNIST **(c)** CIFAR 10

Figure 4.7: Test BPD evaluated after every training epoch. We find, compared to the denoising regularizer, the proposed amortized inference training scheme consistently achieves better generalization performance in all tasks.

Tightness of the ELBO

In this section we want to verify the tightness of the ELBO as a lower bound of the log likelihood. Consider the likelihood for a single data point $\boldsymbol{x}'$, we have

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')}\Big(\log p_{\boldsymbol{\theta}}(\boldsymbol{x}'|\mathbf{z})\Big) - \mathrm{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')||p(\mathbf{z})\Big) \tag{4.29}$$

$$\equiv \mathrm{ELBO}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}). \tag{4.30}$$

To evaluate $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}')$ we can use an importance weighted estimation (IWAE [127]):

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}') = \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\Big(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}'|\mathbf{z})p(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\Big) \tag{4.31}$$

$$\approx \log \frac{1}{K}\sum_{k=1}^{K}\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}'|\boldsymbol{z}^k)p(\boldsymbol{z}^k)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}^k|\boldsymbol{x}')} \tag{4.32}$$

$$\equiv \text{IWAE}_k(\boldsymbol{x}', \boldsymbol{\theta}, \boldsymbol{\phi}), \tag{4.33}$$

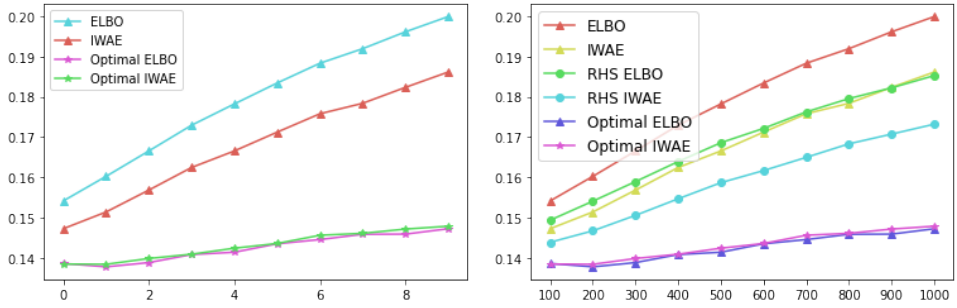
where $\boldsymbol{z}^k \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$. The accuracy of the importance sampling heavily depends on the proposal distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ and will be poor if $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ underestimates the high density regions of $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ [127]. For the ELBO computed using the optimal inference procedure, we can assume that the approximate posterior is close to the true posterior. Therefore if the lower bound is tight we will observe that the ELBO is approximately equal to the IWAE.

In figure 4.8 we compare the ELBO and IWAE using classic amortized inference and the optimal inference setup respectively. We use $k = 10$ in all cases. We find that the IWAE can improve the ELBO for the traditional amortized inference and is approximately equivalent to the ELBO using the optimal inference strategy. This is evdience to support that the ELBO using the optimal inference strategy is tight to $\log p_{\boldsymbol{\theta}}(\mathbf{x})$.

We also estimate the IWAE using the proposal posterior learned by our reverse half-sleep inference and find that our method can also improve the IWAE result; see figure 4.8 for details. In this case our method is providing a better proposal distribution for importance sampling.

Effects of latent space dimensionality

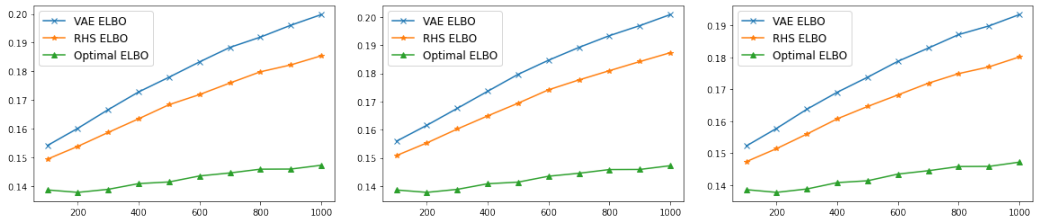
In this section we look at the effect of the dimensionality of $\mathbf{z}$ on the generalization performance. We use the VAE described in section 4.5 applied to binary Mnist with different sized latents: $[16, 64, 128]$. In figure 4.9 we find that the amortized inference network overfits in all cases, regardless of the latent size. We then apply our reverse half-asleep procedure to the checkpointed



(a) Tightness **(b)** All Comparisons

Figure 4.8: IWAE comparisons on binary MNIST. The x-axis indicates the training epoch and the y-axis is the Bits-per-dimension, which corresponds to the negative ELBO or IWAE with log 2 base and normalized by data dimension, lower is better. In figure a, we see that IWAE improves the ELBO when using classic amortized inference but is approximately equal to the ELBO when using optimal inference, which indicates the bound is tight. In figure b, we compare the IWAE with classic amortized inference, optimal inference and the proposed reverse half-asleep (RHS) inference. Here we find the proposed method can also improve the classic IWAE estimation without training on the test data.

model every 100 epochs and found it consistently improves the generalization performance across all latent sizes.



(a) $\text{Dim}(z) = 16$ **(b)** $\text{Dim}(z) = 64$ **(c)** $\text{Dim}(z) = 128$

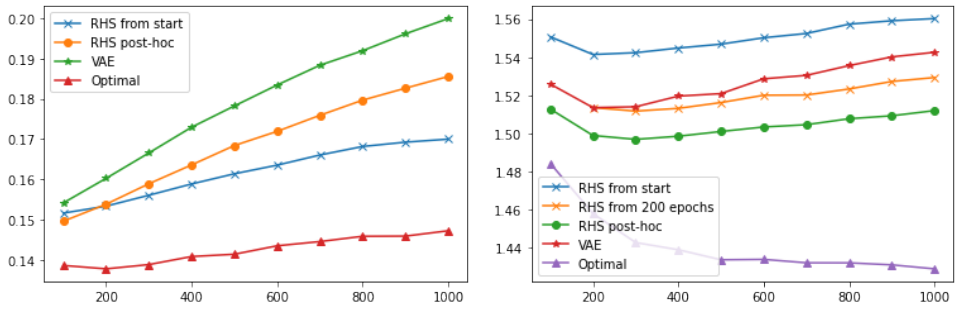
Figure 4.9: Effects of different latent dimension. The y-axis is the BPD and x-axis is the training epochs. We find the amortized inference generalization gap exits in all cases.

Reverse half-sleep from the beginning

Up to this point we have applied the reverse half-sleep training in a post-hoc fashion. This allowed us to extend the original VAE training with this additional step. It also allowed us to isolate the degree to which both the generative model and amortized inference generalization gaps are contributing to overfitting.

It has also been observed that a poor variational posterior in the early

stage of training can cause the M-step of the generative model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ to get trapped into a local minimum (see "Two problems with variational expectation maximization for time-series models" section in [128]). Here we want to check how our method behaves if applied from the beginning of training as opposed to just as a bolt on at the end with a fixed $\boldsymbol{\theta}$. In figure 4.10 we find that using the proposed reverse half-sleep from the beginning can lead to a better test ELBO compared to the classic VAE training, or our proposed post-hoc training, for a relatively simple dataset like Binary MNIST. However, we also find that for a more complex dataset like grey-scale MNIST, applying our method from the beginning performs worse than the classic VAE training. We hypothesize that for a complex dataset, the model in the beginning cannot generate valid images, which will lead to biased gradients. We also report the results of using the reverse half-sleep training starting from 200 epochs onwards and find it does in fact improve performance over classic VAE training, but is still worse than the post-hoc version. We leave further study of how to improve the generalization from the beginning of the training to future work.



(a) Binary MNIST **(b)** Grey MNIST

Figure 4.10: We compare different ways of using the proposed training objective (from the beginning or post hoc). We also plot the standard VAE training and the ELBO with optimal inference for reference.

4.6 Related work

A different perspective on generative models' generalization is proposed in [129] where the generalization is evaluated by testing if the model can generate novel combinations of features. In contrast, we measure generalization using

test likelihood, which is more relevant for certain applications like lossless compression. Recent work [100] first studies likelihood-based generalization in context of lossless compression. They focus on the scenario where the test and train data come from different distributions, whereas we assume they both follow the same underlying distribution. Additionally, their model has a tractable likelihood and they focus purely on the generative model generalization gap, whereas we focus on the amortized inference generalization gap in VAEs.

Previous work [115] studied the so called amortization gap in amortized inference. This is caused by using $q_{\boldsymbol{\phi}_*}(\mathbf{z}|\boldsymbol{x}^n)$ to generate posteriors for each input $\boldsymbol{x}^n$ rather than learning a posterior $q_*^n(\mathbf{z})$ for each $\boldsymbol{x}^n$ individually. This gap can be alleviated using a larger capacity inference network. This amortization gap is fundamentally different from the amortized inference generalization gap we discuss in this chapter since the latter focuses solely on test time generalization but the former problem also exists at training time.

Recent work [130] proposes a compression scheme based on the IWAE [127] bound, which is tighter than the ELBO and thus improves the compression rate. However, this method has to compress/decompress multiple latent samples, which requires extra time cost. On the other hand, we focus on improving the ELBO-based compression that only needs to compress one single latent sample. Nevertheless, similar to the K-step optimal inference strategy, our amortized training objective can also be used in the IWAE-based method, which gives a better proposal distribution for importance sampling.

Another example of related previous work [131] considers the following data generation process $\boldsymbol{x}^1 \sim p_d(\mathbf{x})$, $\boldsymbol{z}^1 \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}^1)$, $\boldsymbol{x}^2 \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}^1)$ and propose enforcing latent consistency between $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}^1)$ and $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}^2)$ for the pairs $(\boldsymbol{x}^1, \boldsymbol{x}^2)$ to encourage the learned representations to be more robust. This procedure is similar to the self-supervised contrasting learning procedure [132] where the augmented data is training data reconstructed using the VAE model. In this chapter, we want to encourage samples from the model $\boldsymbol{x}' \sim \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$ to have high ELBO under the model (equation 4.23) to improve the generalization

of the amortized inference network. We don't require any paired data in our method.

4.7 Conclusion

We have shown how the generalization of the VAE class of latent variable model is largely affected by the amortized inference network. We proposed an additional training stage for the inference network that uses generations from the model that provides better generalization for a given set of training data; as demonstrated in the applications to down stream classification and compression problems.

Improving the generalization performance of the decoding model in addition to the inference network was outside of the scope of this work; but is an obvious future consideration. As well as scaling our method up to larger generative models and adapting it for hierarchical latent variable models.

In particular we are interested in exploring how this method can benefit applications of VAE style models in molecular design, where generalization to unseen structures is paramount [133]. Furthermore extending our method to the challenges of hierarchical VAE models [134].

Chapter 5

Solipsistic Reinforcement Learning

In this chapter we present ideas for a latent variable model-based deep reinforcement learning framework that aims to tackle environments with high dimensional state spaces in a more data efficient way; in particular attempting to learn directly from pixels. By improved data efficiency in this RL setting, as discussed in chapter 1, we mean reducing the number of environment interactions required to achieve a target reward.

Specifically, we aim to address the research question as to whether it is possible to develop a framework that learns a low dimensional latent representation of the environment while avoiding the need to learn a generative model of the environment itself? And furthermore whether that representation can be effective for planning and reward modelling?

We refer to this as a solipsistic representation that we train to encode a belief that is consistent with the dynamics of the environment and is then exploited for effective planning. We explore choices of model and corresponding planning algorithms that can deal with both discrete and continuous state environments. We demonstrate empirically gains in data efficiency over existing model-free methods when learning directly from pixels and analyze the properties of our learned representations.

5.1 Reinforcement learning

The real world is complex and a learning agent must be able to recognize relevant signals to decide what actions to take towards reaching a goal. The focus of our work is to form environment representations for model-based planning and reward prediction without having to learn a generative model of the potentially complex environment.

For example, Pavlov's dog[135] learns to associate the sound of a bell with the eventual reward of food in spite of other potential sensory distractions. If we attempted to create a model for the dog's predictions of this reward it would be unnecessary to create a complex representation of the dog's complete sensory experience of the laboratory. Instead we should only need a solipsistic [1] representation that encodes the belief of the dog as to whether food will appear or not conditioned on the extracted signal of bell sounds alone. Thus, lessening the burden of the modeling problem. This intuition is the essence of what we are trying to formalize in this work under a model-based reinforcement learning (RL) framework. Our general principle of interest is representation learning that automatically encodes only the information in the environment needed to solve a given decision problem; ultimately improving the data efficiency of the agent. Practically speaking, we attempt to answer the question of whether it's possible to learn a useful low dimensional representation of an environment for planning and reward prediction without having to learn a generative model of the environment itself. We believe that any successful steps in this direction is progress towards more effective real world deployments of reinforcement learning.

To motivate our approach, Pavlov's dog [135] learns to associate the sound of a bell with the eventual reward of food, despite sensory distractions. There are classically two interpretations: (a) a model-free interpretation is that the dog learns a value $\mathcal{V}(\mathbf{x}_t)$ (expectation of eventual food reward) as a function of the environment state $\mathbf{x}_t$ at time t; (b) a standard model-based interpretation

[1] Solipsism is the philosophical idea that the external world may not exist and only an internal world may exist, whose representations are informed by the external world alone.

is that the dog models the environment and can use that to predict the future $p(\mathbf{x}_{t+k}|\mathbf{x}_t)$ and any eventual reward $p(\mathbf{r}_{t+k}|\mathbf{x}_{t+k})$. In contrast to these standard approaches, we posit an alternative model-based interpretation in which the dog forms an internal 'solipsistic'[2] low-dimensional representation $\mathbf{s}_t$ as a function of the external environment state $\mathbf{x}_t$ and forms a predictive model $p(\mathbf{s}_{t+k}|\mathbf{s}_t)$ of the representation, without learning a model of the environment itself. This representation is useful if the dog is able to accurately predict eventual reward using the solipsistic transition $p(\mathbf{s}_{t+k}|\mathbf{s}_t)$ and reward model $p(\mathbf{r}_{t+k}|\mathbf{s}_{t+k})$.

In the Reinforcement Learning (RL) setting an agent observes state $\boldsymbol{x}_t$ from the environment at time-step t, takes action $\boldsymbol{a}_t$, and subsequently observes $\boldsymbol{x}_{t+1}$ and reward $\boldsymbol{r}_{t+1}$. The goal of the agent is to learn (through interactions with the environment) how to take actions that result in favorable long-term rewards [35]. A standard RL assumption is that there is an underlying Markov Decision Process with transition $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$ [137, 138, 139, 140]; rewards are functions of the observed state and the goal is usually to take actions that maximize cumulative reward, see figure 5.1a[3]. The action $\boldsymbol{a}_t$ depends on the state $\boldsymbol{x}_{t-1}$, meaning that the state $\boldsymbol{x}_{t-1}$ is revealed before the action $\boldsymbol{a}_t$ is decided.

In model-based RL, we attempt to learn the model of the transition dynamics . Compared to model-free approaches, model-based RL can be significantly more sample efficient [137, 138, 139, 140]. However, for environments with high-dimensional states (such as an image pixels) the complexity and potential redundancy in the observations can make learning the environment dynamics using a model difficult and potentially unnecessary [142, 143].

A recent trend is to learn a lower dimensional representation $\mathbf{s}_t$ that is used to model relevant dynamics and reward prediction, such as PlaNet [143], World Model [142] and other variants [141, 142], see figure 5.1b. The usual strategy is to train a latent variable model $p(\mathbf{x}_t) = \int p(\mathbf{x}_t|\mathbf{s}_t)p(\mathbf{s}_t)d\mathbf{s}_t$ with an

[2]We use 'solipsism' to refer to the philosophy that only an internal representation of the world may exist [136]. In our context, the agent can plan on the basis of an internal dynamical representation of the external world.

[3]Alternatively one may use a higher-order Markov model $p(\mathbf{x}_{t+1}|\boldsymbol{h}_t)$ where $\boldsymbol{h}_t = \{\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}\}$ is the history of states and actions up to time point t [141].

encoding-decoding structure [98] like the VAE studied in section 4, while jointly training a dynamics model $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ in the lower dimensional latent space [143, 141, 142, 4]. The learned model infers a latent representation $\boldsymbol{s}_t$ given a state observation $\boldsymbol{x}_t$ (for example a sample from the posterior $p(\mathbf{s}_t|\mathbf{x}_t) \propto p(\mathbf{x}_t|\mathbf{s}_t)p(\mathbf{s}_t)$) that can be used by the dynamics model for efficient planning. Arguably, a limitation of these recent approaches is that they spend significant computational resources on learning a generative model of the high-dimensional state $\mathbf{x}_t$ – however, this generative model is not used directly during the planning phase. The learned representation $\mathbf{s}_t$ in these approaches is therefore likely encoding redundant information about the environment $\mathbf{x}_t$.

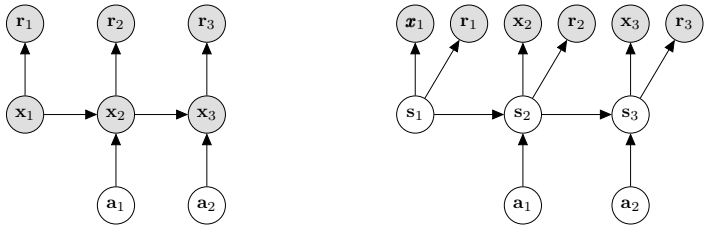
An alternative is to use model-free algorithms such as Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), amongst others [144, 145, 146, 147]. The upside is that these approaches avoid creating a generative model of the environment by learning a policy from pixels to state values. A potential downside is that they suffer from poor data efficiency compared to model based approaches; meaning they generally require much larger numbers of interactions with the environment.

The question we therefore study here, similar to other recent research work in this area [148], is whether it is possible to perform model-based RL without making a generative model of the environment. If this were possible, we could potentially reap the benefits of the data-efficiency of model-based RL, without the need to model complex high-dimensional observations.

5.2 Solipsistic representations

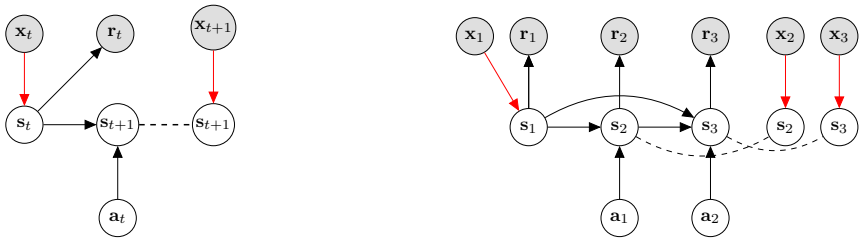
A good solipsistic representation $\boldsymbol{s}_t$ of an observation $\boldsymbol{x}_t$ is one that is consistent (the predicted next solipsistic state $\boldsymbol{s}_{t+1}$ given action $\boldsymbol{a}_t$ matches the observed next solipsistic state $\boldsymbol{s}_{t+1}$) and informative (one can predict the reward well using $\boldsymbol{s}_t$).

A solipsistic Markov model is depicted in figure 5.2a, where we remove the arrow from $\mathbf{s}$ to $\mathbf{x}$ in the latent variable model and instead introduce



(a) Observed Markov Model **(b)** Latent Variable Model.

Figure 5.1: Graphical models for model-based RL. Shaded nodes denote observed quantities. (a) Model dynamics and reward in the original space $\mathbf{x}$ and (b) in latent space $\mathbf{s}$.

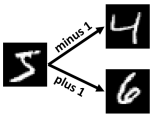


(a) Solipsistic Markov Model **(b)** Solipsistic Memory Model

Figure 5.2: Graphical models of the solipsistic Markov model (a) and memory model (b). We color the edge from the observation to the latent state to highlight that this is not a generative model of the environment. A dashed line indicates the consistent relationship between solipsistic state prediction and future state recognition, as discussed in section 5.2.

a recognition distribution $p(\mathbf{s}_t|\mathbf{x}_t)$, the purpose of which is to encode only the information in $\mathbf{x}_t$ that is needed to effectively learn the dynamics and the reward of the environment. We introduce a toy 'MNIST game' to help build intuition, see figure 5.3. Each observation $\boldsymbol{x}_t$ is a 28×28 MNIST image representing a digit from 0 to 9. The agent has two possible actions: 'minus 1' or 'plus 1'; the environment shows the resulting digit's image. The digit will stay the same when taking action 'minus 1' from digit 0 and 'plus 1' from digit 9. The game is initialized at digit 4. The reward r_t is 1 if the state is an image of digit 9; otherwise the reward $r_t = 0$. Whilst the observation $\boldsymbol{x}_t$ is a 784 dimensional image, clearly the underlying dynamics is representable by an integer $\boldsymbol{s}_t \in \{0, \ldots, 9\}$ (that we represent in practise by 1 hot vectors).

(a) States of the environment



(b) Transition dynamics

Figure 5.3: MNIST game. (a) The observation x_t is one of the 10 images. (b) Given a 'plus 1' action, the following image x_{t+1} is a higher digit and vice versa for 'minus 1'.

5.2.1 Learning objective

In what follows we describe three desiderata for our latent variable model that ultimately inform our learning objective - consistency in the predicted latent dynamics, consistency in the predicted rewards and the effective filtration of redundant observation state information.

Latent dynamics consistency

We wish to ensure that, for a given recognition distribution $p_{\boldsymbol{\phi}}(\mathbf{s}_t|\mathbf{x}_t)$, parameterised by a neural network with parameters $\boldsymbol{\phi}$, the dynamics of the solipsistic model are consistent with the dynamics of the true environment when training the model under sampled trajectories. In the setting of Pavlov's dog, if the model maps the current environment $\boldsymbol{x}_t$ to the internal state $\boldsymbol{x}_t \to \boldsymbol{s}_t$ that represents hearing the bell $\boldsymbol{s}_t = \mathsf{bell}$, and the dog predicts $\boldsymbol{s}_t \to \boldsymbol{s}_{t+1}$ from this that $\boldsymbol{s}_{t+1} = \mathsf{food}$, then we must observe that the next external state $\boldsymbol{x}_{t+1} \to \boldsymbol{s}_{t+1}$ indeed maps to food. This ensures that latent transitions are effective for planning.

Similarly, in the MNIST game, we need to force the solipsistic model to predict a digit that is consistent with the image that would appear in the next time step under the true environment transition. More specifically, we assume a Markov transition distribution $p_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, parameterised by a neural network with parameters $\boldsymbol{\theta}$, which takes the current solipsistic state and action as input and gives the distribution for the next solipsistic state $\mathbf{s}_{t+1}$.

The solipsistic state distribution at time $t+1$ can be formulated as

$$p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_t,\mathbf{a}_t) = \sum_{\mathbf{s}_t} p_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)p_{\boldsymbol{\phi}}(\mathbf{s}_t|\mathbf{x}_t). \tag{5.1}$$

We want this predicted distribution to be consistent with the recognition distribution $p_{\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_{t+1})$ from the next time step. To achieve this we introduce an agreement objective. We aim to leverage maximum likelihood, as introduced in section 2.1, against sampled trajectories for training, therefore the KL divergence is a natural choice:

$$\mathrm{KL}\Big(p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_t,\mathbf{a}_t)||p_{\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_{t+1})\Big). \tag{5.2}$$

Reward consistency

For effective planning, we want the solipsistic representation to be useful for reward prediction using a reward model we can learn from experience $p_{\boldsymbol{\eta}}(\mathbf{r}_t|\mathbf{s}_t)$. Given solipsistic state $\boldsymbol{s}_t$ and observed reward $\boldsymbol{r}_t$ pairs, we can again use maximum likelihood, to learn the reward model $p_{\boldsymbol{\eta}}(\mathbf{r}_t|\mathbf{s}_t)$. This is equivalent to minimizing the KL divergence $\mathrm{KL}(\tilde{p}(\mathbf{r}_t)||p_{\boldsymbol{\eta}}(\mathbf{r}_t|\mathbf{s}_t))$ between the empirical reward distribution $\tilde{p}(\mathbf{r}_t)$ that places all mass on the observed rewards and the model $p_{\boldsymbol{\eta}}(\mathbf{r}_t|\mathbf{s}_t)$. We assume that we are in a dense reward settings; that is we observe a reward at every time value.

Redundancy filtering

Finally, the intention of the recognition distribution $p_{\boldsymbol{\phi}}(\mathbf{s}_t|\mathbf{x}_t)$ is to filter out redundant information when producing the solipsistic representation. To achieve this, we need it to learn to distinguish if the state information is useful or not for both reward prediction and transition dynamics. For example in the MNIST game, the recognition distribution may just focus on the image backgrounds which are stationary over time and the solipsistic transition function could just learn the identity mapping. Although this forms a consistent solipsistic

representation, it would be useless for planning. Furthermore, having a reward objective is not sufficient to avoid this behavior. Features of the state which inform reward prediction may not be the same as those which inform a useful dynamics model. For example in the MNIST game, the recognition distribution may just keep the features which are relevant to distinguish if an image is 9 or not for predicting the reward, but ignore other useful information about the system dynamics.

One solution is to additionally encourage the agent's solipsistic dynamics to be inconsistent with trajectories which are not observed in reality. In other words, we want an objective that forces the predictive distribution $p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x},\mathbf{a}_t)$ to be different from $p_{\boldsymbol{\eta}}(\mathbf{s}_i|\mathbf{x}_i)$ for $i \neq t+1$ [4]. To achieve this we can maximize the expected KL divergence:

$$\mathbb{E}_{i\neq t+1}\mathrm{KL}\Big(p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_t,\mathbf{a}_t)||p_{\boldsymbol{\phi}}(\mathbf{s}_i|\mathbf{x}_i)\Big). \tag{5.3}$$

We refer to this as a solipsistic contrast term. It is similar in spirit to the contrastive loss used for representation learning or self-supervised learning [149, 150, 151]. Since the KL divergence is unbounded here, we use a positive constant cap m.

Overall objective

For a trajectory of length T our overall objective for jointly training the components introduced for our solipsistic model combines the three elements from the previous sections. We seek to minimize

$$\begin{aligned}&\frac{1}{T-1}\sum_{t=1}^{T-1}\mathrm{KL}\Big(p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_t,\mathbf{a}_t)||p_{\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_{t+1})\Big)+\frac{\lambda_r}{T}\sum_{t=1}^{T}\mathrm{KL}(\tilde{p}(\mathbf{r}_t)||p_{\boldsymbol{\eta}}(\mathbf{r}_t|\mathbf{s}_t))\\&+\frac{\lambda_s}{T-1}\sum_{t=1}^{T-1}\mathbb{E}_{i\neq t+1}\max\Big(0,m-\mathrm{KL}\Big(p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_t,\mathbf{a}_t)||p_{\boldsymbol{\phi}}(\mathbf{s}_i|\mathbf{x}_i)\Big)\Big)\end{aligned} \tag{5.4}$$

[4] More generally, in the case of multiple trajectories, we can sample $\boldsymbol{s}_i$ from other trajectories.

with respect to the parameters of the recognition distribution $\boldsymbol{\phi}$, transition distribution $\boldsymbol{\theta}$ and reward distribution $\boldsymbol{\eta}$; λ_s, λ_r and m are user chosen hyper-parameters. In our experience, the results are not particularly sensitive to the choice of these parameters, which we discuss further in section 5.4.

For tasks that require accurate long-term planning, prediction errors using the simple solipsistic Markov model may accumulate during roll-out[5]. In such cases we consider a variation of our model we refer to as the Solipsistic Memory Model (SMM). The SMM differs in that it's transition depends on all past solipsistic states and actions – see figure 5.2b for the graphical model representation. Within our overall objective 5.4, we replace $p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$ with $p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{s}_{t+1}|\mathbf{x}_t, \mathbf{h}_t)$ where $\mathbf{h}_t = \{\mathbf{s}_{1:t}, \mathbf{a}_{1:t}\}$ and use a recurrent neural network [64, 65] to learn the transition dynamics of the solipsistic model.

5.3 Acting and planning

For a given solipsistic model, we take a sequence of actions using a re-planning procedure: we observe $\boldsymbol{x}_1$ and determine the first solipsistic state distribution using the recognition process $p_{\boldsymbol{\phi}}(\mathbf{s}_1|\boldsymbol{x}_1)$. We then determine $\boldsymbol{a}_1^*$ using a planning procedure (e.g. dynamic programming, or sampling from a trained parameterized policy, as we discuss further below) and take this action in the environment, observing the resulting $\boldsymbol{x}_2$. We repeat this recognition and planning procedure until time T, observing $\boldsymbol{x}_t$ at each step and then planning the best next action. This process of re-planning at every time step helps prevent the accumulation of prediction errors from our model and is efficient since our solipsistic representation is relatively low dimensional. How planning and acting in the environment is folded into the overall RL process of model and policy training is described in detail in algorithm 1.

[5]Whilst the underlying physical dynamics of a problem might be Markovian, any pixel based representation will result in some discretization error. For long sequences, small discretization errors can accumulate, resulting in poor long term prediction unless a longer term history is used.

5.3.1 Markov model

Given the observation $\boldsymbol{x}_1$, we would like to predict the expected rewards we would obtain by taking a sequence of subsequent actions $\boldsymbol{a}_{1:T-1}$. The recognition distribution enables us to determine the distribution for the first solipsistic state $p_{\boldsymbol{\theta}}(\mathbf{s}_1|\boldsymbol{x}_1)$. The state-action trajectory can be described by the distribution

$$p(\mathbf{s}_{1:T}|\boldsymbol{x}_1, \boldsymbol{a}_{1:T-1}) = p(\mathbf{s}_1|\boldsymbol{x}_1) \prod_{t=1}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \boldsymbol{a}_t). \tag{5.5}$$

For planning, the goal is to maximize the cumulative reward by choosing a sequence of actions. For a discrete solipsistic state space, the objective is

$$\sum_{\mathbf{s}_{1:T}} \left(\sum_{t=1}^{T} \mathcal{R}(\mathbf{s}_t) \right) p(\mathbf{s}_1|\boldsymbol{x}_1) \prod_{t=2}^{T} p(\mathbf{s}_t|\mathbf{s}_{t-1}, \boldsymbol{a}_{t-1}), \tag{5.6}$$

where we use the reward function defined as $\mathcal{R}(\mathbf{s}_t) \equiv \mathbb{E}_{p_{\boldsymbol{\eta}}(\mathbf{r}_t|\mathbf{s}_t)}[\mathbf{r}_t]$. If we interpret the solipsistic model as a Markov Decision Process (MDP) then the optimal action sequence has value

$$\max_{\boldsymbol{a}_1} \sum_{\mathbf{s}_1} p(\mathbf{s}_1|\boldsymbol{x}_1, \boldsymbol{a}_1) \cdots \max_{\boldsymbol{a}_{T-2}} \sum_{\mathbf{s}_{T-1}} p(\mathbf{s}_{T-1}|\mathbf{s}_{T-2}, \boldsymbol{a}_{T-1}) \max_{\boldsymbol{a}_{T-1}} \sum_{\mathbf{s}_T} p(\mathbf{s}_T|\mathbf{s}_{T-1}, \boldsymbol{a}_{T-1}) \sum_{t=1}^{T} \mathcal{R}(\mathbf{s}_t). \tag{5.7}$$

This is readily solved by dynamic programming.

Value Estimation

We let $\mathcal{V}(\mathbf{s}_T) = \mathcal{R}(\mathbf{s}_T)$. For $t = T-1, \ldots, 2$ and each state of $\mathbf{s}_t$ we calculate

$$\mathcal{V}(\mathbf{s}_t) = \mathcal{R}(\mathbf{s}_t) + \max_{\boldsymbol{a}_t} \sum_{\mathbf{s}_{t+1}} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \boldsymbol{a}_t) V(\mathbf{s}_{t+1}). \tag{5.8}$$

The first optimal action can be computed using the value function

$$\boldsymbol{a}_1^* = \arg\max_{\boldsymbol{a}_1} \sum_{\mathbf{s}_1} p(\mathbf{s}_1|\boldsymbol{x}_1) \sum_{\mathbf{s}_2} p(\mathbf{s}_2|\mathbf{s}_1, \boldsymbol{a}_1) \mathcal{V}(\mathbf{s}_2). \tag{5.9}$$

We then take the first optimal action $\boldsymbol{a}_1^*$ in the real environment to get observation $\boldsymbol{x}_2$, and do re-planning based on the new solipsistic distribution $p(\mathbf{s}_2|\boldsymbol{x}_2)$. The general procedure is to repeat: (1) take the action $\boldsymbol{a}_t^*$ in the environment and get the new observation $\boldsymbol{x}_{t+1}$, (2) use the recognition function to compute $p(\mathbf{s}_{t+1}|\boldsymbol{x}_{t+1})$, (3) compute the next action $\boldsymbol{a}_{t+1}^*$ using

$$\boldsymbol{a}_{t+1}^* = \arg\max_{\boldsymbol{a}_{t+1}} \sum_{\mathbf{s}_{t+1}} p(\mathbf{s}_{t+1}|\boldsymbol{x}_{t+1}) \sum_{\mathbf{s}_{t+2}} p(\mathbf{s}_{t+2}|\mathbf{s}_{t+1}, \boldsymbol{a}_{t+1}) \mathcal{V}(\mathbf{s}_{t+2}). \tag{5.10}$$

By using this re-planning scheme, we can sequentially decide the optimal action sequence under our model, which we demonstrate in our MNIST game in section 5.4.1. For continuous latent states and non-linear transition dynamics, exact dynamic programming is usually not available. See [152] for alternative approximate dynamic programming techniques.

5.3.2 Memory model

In the SMM dynamic programming becomes intractable and we instead learn a policy $p_{\boldsymbol{w}}(\mathbf{a}_t|\mathbf{s}_t)$, parameterized by a neural network with paramters $\boldsymbol{w}$. Here we assume that the observed state $\boldsymbol{s}_t$ contains sufficient information to determine the best action; however, in order to accurately track long term behavior we need to use the SMM dynamics to track changes in state. This is typically required in the case of pixel-based planning in which the best action is readily determinable from the current state; however, keeping track of the long-term consequences of a sequence of actions requires using a history of states because of the discretization errors in observing images only.

The resulting objective to maximize is

$$\mathcal{E}(\boldsymbol{w}) \equiv \int \left(\sum_{t=1}^{T} \mathcal{R}(\mathbf{s}_t) \right) p(\mathbf{s}_1|\boldsymbol{x}_1) \prod_{t=2}^{T} p(\mathbf{s}_t|\mathbf{h}_{t-1}) p_{\boldsymbol{w}}(\mathbf{a}_{t-1}|\mathbf{s}_{t-1}) d\mathbf{s}_{1:T} d\mathbf{a}_{1:T-1}. \tag{5.11}$$

where, for discrete actions, the integral over $a_{1:T-1}$ can be replaced by summation.

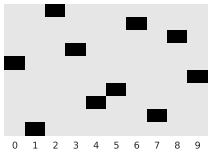
Previous work has demonstrated that Variational Optimization (VO) style algorithms [153, 154] have an advantage over policy gradients for environments with long time horizons. We therefore use a standard VO algorithm to learn the policy parameters $\boldsymbol{w}$, see appendix(C.1).

In the RL setting, we iterate through cycles of the following steps; trajectory collection from the environment (using the latest best policy and some action exploration), model training, policy training (which are separate steps - full details provided in algorithm 1) and policy evaluation. We provide specific parameter settings and architecture choices in section 5.4.2. We re-initialise the parameters of our policy $\boldsymbol{w}$ to a random initialisation after every round of sampling a new batch of trajectories and updating our transition, recognition and reward models offline. This is so that our policy learning process using VO does not over-fit to older trajectories and can appropriately exploit the latest information about the environment.

In the case of discrete actions, which we encounter in section 5.4.2, we choose at each step of the environment interaction a random action with probability ϵ and an action based on the current best policy with probability $(1-\epsilon)$. In the case of continuous actions, small Gaussian noise can be added to the action chosen based on the current best policy.

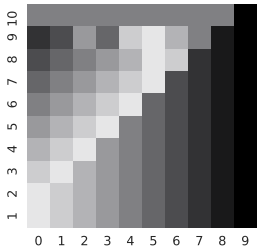
Algorithm 1 Overall learning procedure in practise - bringing together the previous methodology, where we assume the use of the memory model and a parameterized policy.

Set constants I, N_{model}, N_{policy}, $T_{collect}$, T_{model}, T_{policy}, B, ϵ, J
Initialize $\mathcal{M} \leftarrow \emptyset$
Initialize Parameters of $\boldsymbol{\theta}_0$, $\boldsymbol{\phi}_0$, $\boldsymbol{\eta}_0$, $\boldsymbol{w}_0$
for $i = 0, \ldots, I$ **do**
    for $c = 0, \ldots, C$ **do**
        Reset environment
        for $t = 1, \ldots, T_{collect}$ **do**
            Observe $\boldsymbol{x}_t, \boldsymbol{r}_t$ and sample $\boldsymbol{s}_t \sim p_{\boldsymbol{\phi}_i}(\mathbf{s}_t|\boldsymbol{x}_t)$
            Sample $\hat{\boldsymbol{a}}_t$ with exploration strategy using $p_{\boldsymbol{w}_i}(\mathbf{a}_t|\boldsymbol{s}_t)$, ϵ
            Take action $\hat{\boldsymbol{a}}_t$ in environment
        end for
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(\boldsymbol{x}_t, \boldsymbol{s}_t, \hat{\boldsymbol{a}}_t, \boldsymbol{r}_t)\}_{t=1}^{T_{collect}}$
    end for
    for $n = 0, \ldots, N_{model}$ **do**
        Sample trajectories $\{\{(\boldsymbol{x}_t, \boldsymbol{s}_t, \hat{\boldsymbol{a}}_t, \boldsymbol{r}_t)_b\}_{t=1}^{T_{model}}\}_{b=1}^{B} \sim \mathcal{M}$
        Update $\boldsymbol{\theta}_i$, $\boldsymbol{\phi}_i$, $\boldsymbol{\eta}_i$ using equation 5.4 and ADAM
    end for
    Re-initialize $\boldsymbol{w}_i$ to a random policy
    Initialise VO parameter $\boldsymbol{\mu}_0 \leftarrow \boldsymbol{w}_i$
    for $n = 0, \ldots, N_{policy}$ **do**
        for $j = 0, \ldots, J$ **do**
            Initialize $\boldsymbol{h}_1 \leftarrow \emptyset$
            Sample $\boldsymbol{x}_1$ from memory $\mathcal{M}$
            Sample $\boldsymbol{s}_1 \sim p_{\boldsymbol{\phi}}(\mathbf{s}_1|\boldsymbol{x}_1)$
            Sample $\boldsymbol{w}^j \sim \mathcal{N}(\boldsymbol{\mu}_n, \sigma = 0.2)$
            for $t = 0, \ldots, T_{policy}$ **do**
                Sample $\boldsymbol{a}_t \sim p_{\boldsymbol{w}_i^j}(\mathbf{a}_t|\boldsymbol{s}_t)$
                $\boldsymbol{h}_t \leftarrow \boldsymbol{h}_t \cup (\boldsymbol{s}_t, \boldsymbol{a}_t)$
                Predict $\mathcal{R}(\boldsymbol{s}_t)$
                Sample $\boldsymbol{s}_{t+1} \sim p_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}|\boldsymbol{h}_t)$
            end for
            Compute $\mathcal{E}(\boldsymbol{w}^j) = \sum_{t=1}^{T} \mathcal{R}(\boldsymbol{s}_t)$
        end for
        Compute $\boldsymbol{\mu}_{n+1}$ with $\{\mathcal{E}(\boldsymbol{w}^j)\}_j^J$ using VO equation C.2 and ADAM
    end for
    Update policy parameters $\boldsymbol{w}_{i+1} \leftarrow \boldsymbol{\mu}_{n+1}$
end for



(a) Recog. $p(\mathbf{s}_t|\mathbf{x}_t)$.

(b) Reward $p(\mathbf{r}_t|\mathbf{x}_t)$.



(c) $\mathcal{V}(\mathbf{s}_t)$ for t from 1 to 10.

Figure 5.4: The learned solipsistic model for the MNIST game. The pixel images $\boldsymbol{x}_t$ are represented by their corresponding number on the x-axis. (a) The y-axis is the solipsistic state $\boldsymbol{s}_t$. The model learns to associate an image $\boldsymbol{x}_t$ with a unique solipsistic state $\boldsymbol{s}_t$, with $p(\mathbf{s}_t|\mathbf{x}_t)$ being almost deterministic. (b) The learned model predicts the instantaneous reward correctly. (c) The y-axis is the time step from 1 to 10. The value is normalized within each time-step, the darker patches indicate higher values. Given any state at time step 1, the optimal action is always 'plus 1'.

5.4 Experiments

Rather than showing state-of-the-art across a range of RL challenges, the goal of the experiments is to confirm our hypothesis that model-based RL can be achieved without requiring a generative model of the observations $\boldsymbol{x}_t$ and to go deeper into understanding qualitatively how the learned representations are behaving. We discuss details of the simple MNIST game introduced in section 5.2. We apply our proposed method to a variation of the Cartpole benchmark from OpenAI gym [155]. If using the low dimensional cartpole state provided by this environment, this is an easy problem for control. We instead try to learn directly from pixels using continuous $\boldsymbol{s}_t$ to demonstrate our recognition distribution works as expected. Further details of the models and training are given in appendix(C.2).

5.4.1 MNIST Game

For the solipsistic state we assume we know the true number of states $\boldsymbol{s}_t \in \{1, \ldots, 10\}$ (represented by 1 hot vectors). The recognition distribution $p_{\boldsymbol{\phi}}(\mathbf{s}_t|\mathbf{x}_t)$ is parameterized by a convolutional network. The output of the neural

network is a softmax function, which gives the probability $p_{\boldsymbol{\phi}}(\mathbf{s}_t = i|\mathbf{x}_t)$. We parameterize the transition $p_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathrm{a}_t)$ with two normalized 10×10 matrices $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathrm{a}_t = 0)$ and $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathrm{a}_t = 1)$ (for the actions of minus 1 and plus 1 to the current digit - see the description in section 5.2). The reward distribution $p_{\boldsymbol{\eta}}(\mathrm{r}_t|\mathbf{s}_t)$ is parameterized by a normalized 10×2 matrix. At each episode we select random actions $a_{1:14}$ and collect a single trajectory $\boldsymbol{x}_{1:15}$ to add to our memory[6]. Then during model training, we iteratively sample batches of consecutive states from memory with batch size 64 and update the model using equation 5.4 and ADAM [66]. This process then repeats at each episode.

We can see from figure 5.4a that the 10 different images are assigned to 10 different solipsistic states with high probability. In figure 5.4b, we see successful reward predictions, with the image state 9 having high probability of reward 1. At test time, we randomly initialize the image state and set the horizon for planing to $T = 10$, since for any given state at $t = 1$, the agent can reach the goal state within 10 steps. In figure 5.4c we plot the value of each state from $t = 1, \ldots, 10$. The optimal action at time 1 is 'plus 1' for each initial state; similarly the subsequent optimal action is always 'plus 1'; the learned model has correctly solved the problem.

5.4.2 Cartpole control from pixels

In cartpole, a pole is attached by an un-actuated joint to a cart that moves along a friction-less track, which can be controlled by applying a force of $+1$ or -1 to the cart at each time step. A reward of 1 is received at every time step that the pole remains upright and the episode terminates when it falls over or the cart moves too far from the center. A maximum horizon of 200 time steps is generally used.

Instead of using the low-dimensional states provided by the OpenAI gym, we use the rendered image frames as the state observations. We first gray scale each video frame and down-sample to produce a 64×64 frame $\boldsymbol{f}_t$ at

[6]The MNIST game is simple enough that, during model training, random exploration is sufficient.

time t; three consecutive frames are then stacked to represent each state observation, $\boldsymbol{x}_t = \{\boldsymbol{f}_t, \boldsymbol{f}_{t-1}, \boldsymbol{f}_{t-2}\}$ for $t \geq 3$. This provides higher-order dynamics information like speed, which are hidden in a single frame. For simplicity we assume a stationary initial position (initial pole position is upright) and define $\boldsymbol{x}_1 = \{\boldsymbol{f}_1, \boldsymbol{f}_1, \boldsymbol{f}_1\}$ and $\boldsymbol{x}_2 = \{\boldsymbol{f}_2, \boldsymbol{f}_1, \boldsymbol{f}_1\}$. We use the SMM since this gives better long-term prediction, despite errors from the pixel rendering of the true continuous underlying dynamics.

We follow algorithm 1 to interact with the environment and learn the policy and model. We set $\mathbf{s}_t \in \mathbb{R}^{16}$ and choose $p_{\boldsymbol{\phi}}(\mathbf{s}_t|\mathbf{x}_t) = \delta(\mathbf{s}_t - g_{\boldsymbol{\phi}}(\mathbf{x}_t))$, $p_\theta(\mathbf{s}_{t+1}|\mathbf{h}_t) = \delta(\mathbf{s}_{t+1} - f_{\boldsymbol{\theta}}(\mathbf{h}_t))$. The recognition function $g_{\boldsymbol{\phi}}$ and transition function $f_{\boldsymbol{\theta}}$ are a convolutional neural network and recurrent network respectively. The reward distribution $p_{\boldsymbol{\eta}}(\mathrm{r}_t|\mathbf{s}_t)$ is a Bernoulli distribution with the probability parameterized using a small neural network with a sigmoid output. All model parameters are trained jointly using the ADAM optimizer. Since the KL divergence between two delta distributions is not formally defined for equation 5.4, we use the spread KL divergence [156] with fixed Gaussian spread noise that has variance 0.5, resulting in a square loss objective.

The policy, section 5.3.2, $p_{\boldsymbol{w}}(\mathrm{a}_t|\mathbf{s}_t)$ is a Bernoulli distribution with the probability parameterized using a small feed-forward network with a sigmoid output and trained using variational optimization - see appendix(C.2). When evaluating the trained policy in the real environment we take the most likely action at each re-planning step.

We compare to DQN and PPO [144, 145] following their standard open source implementations adapted for acting directly in pixel space – see appendix(C.2.3) for details. In figure 5.4.2 we report the average reward over 5 different runs for all methods, with each run carried out using different random parameter seeds, but the same fixed initial position of the environment.

Compared to PPO and DQN our solipsistic model-based approach is significantly more data efficient, learning to balance the pole for over 150 time steps, after seeing only 300 trajectories. This demonstrates that our solipsistic

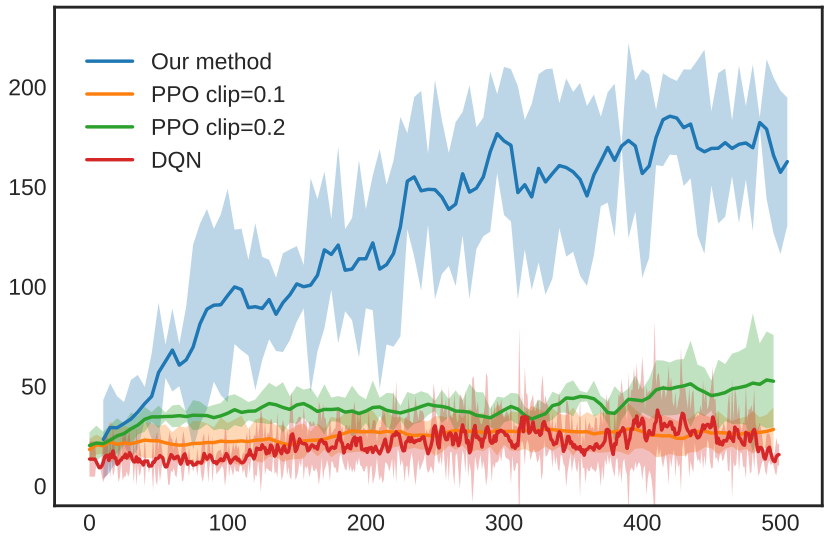


Figure 5.5: Evaluating learned policies. x-axis is number of trajectories sampled from the environment. y-axis is cumulative reward received, averaged over 5 training runs. The data is smoothed using a moving average with window size 3.

approach is accurately modeling both the relevant environment dynamics and the reward. For this setting, we found PPO and DQN struggled to balance the pole for over 50 time steps on average after 2500 sampled trajectories from the environment - see figure C.1 in appendix(C.2.3).

Learned representations

Of interest for a qualitative analysis on the recognition function and learned solipsistic representations are: (1) has the recognition function learned to filter out redundant information? (2) are the solipsistic trajectories consistent with their corresponding observation trajectories? To answer (1) we extract filters from the first layer of the CNN recognition function, which act as attention maps over the pixels. In figure 5.6 we see that different filters are attending to different physical attributes, for example pole position, pole speed and cart position. The right half of the figure shows the activations of the CNN's first layer in the recognition network. We select 4 filters' activations (there are 8 filters in the first layer of recognition network) and use a sigmoid function to create these grey-scale images.

The first filter appears to encode spatial information of the cart and pole (the activated pixels are consistent with the position of both throughout). The second and third filters we believe represent velocity information for the pole (given the activated pixels can be interpreted as providing a finite difference type estimate of velocity by encoding the position of the pole in the first and last frames), while the fourth can be interpreted as encoding information about the velocity of the cart (given more pixels become activated in the region of the cart the faster it goes).

In contrast, no clear physical interpretation was apparent for samples from PPO's convolutional policy and value networks, which we discuss further in appendix(C.2.4). In support of (2), in figure 5.7d we illustrate that the solipsistic trajectories have smooth transitions and are disentangled in accordance with their corresponding trajectories in pixel space.

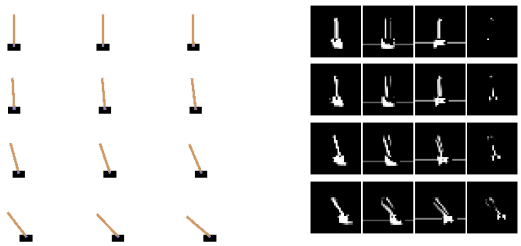
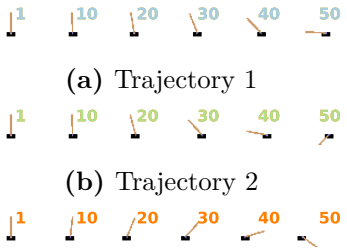


Figure 5.6: Visualization of the CNN filters' activations. We take action 'push cart to the right' for 15 steps, so that the velocity of the cart and pole are monotonically increasing. The left half of the figure shows four states (i.e. the 4 rows) $\{\boldsymbol{x}_1, \boldsymbol{x}_5, \boldsymbol{x}_{10}, \boldsymbol{x}_{15}\}$ where each state is the stack of 3 successive frames (i.e. the 3 columns) $\boldsymbol{x}_t = \{\boldsymbol{f}_t, \boldsymbol{f}_{t-1}, \boldsymbol{f}_{t-2}\}$ This illustrates that different filters are attending to different physical attributes, for example pole position, pole speed and cart position. The right half of the figure shows the activations of the CNN's first layer in the recognition network.

5.5 Related work

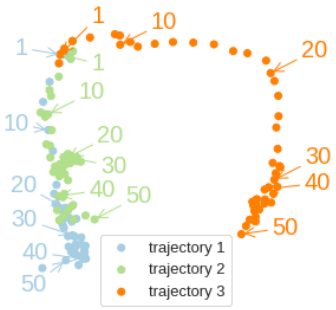
A traditional approach to reduce the complexity of the state is to apply state-aggregation methods, such as non-parametric dimensionality reduction techniques [157], or hand-coded features, to obtain lower-dimensional state representations – see [158] and [159] for a review. These methods require



(a) Trajectory 1

(b) Trajectory 2

(c) Trajectory 3



(d) Solipsistic state trajectories

Figure 5.7: (a,b,c) show three test trajectories, where we plot the frames $\{\boldsymbol{f}_1, \boldsymbol{f}_{10}, \ldots, \boldsymbol{f}_{50}\}$. (d) We construct $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{50}$ by stacking the successive frames within each trajectory, and then produce $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{50}$. We use PCA to project the solipsistic states to 2D for visualization. Trajectories 1 and 2 are similar (the pole falls to the left) and have similar solipsistic trajectories. Trajectory 3 shows the object fall to the right and the solipsistic representation is far away from that of trajectory 1 and 2.

strong prior knowledge about the environment and are not generally useful in situations with complex state spaces. Further the representations are learned separately from the modeling process, which can hinder overall performance when utilized for planning [160]. In contrast, our solipsistic approach jointly learns the dynamics model and representations.

Recent work [161, 162] also propose to use deep neural networks to learn state representations without reconstructing the original state. However, they only demonstrate the benefits of this approach in model-free learning whereas we show how to do planning using the learned representation and the model dynamics.

Contrastive learning is widely used in the field of representation learning [163]. The aim is to encourage similar datapoints to have similar representations [149, 150, 151]. In RL, this idea has been used to improve the data efficiency of model-free algorithms by treating contrastive learning as an auxiliary task [164, 165, 166]. The contrastive objective we use (section 5.2) instead aims to prevent the recognition function from learning a trivial solution to ensure the solipsistic representation is useful for planning.

Most closely related to our work is [148] which uses a Recurrent State Space

Model without a decoder component, akin to our Solipistic Markov Model setup. They use the variational information bottleneck (VIB) principle [167, 168] to derive the following regularizing term for their learning objective:

$$\log p(\mathbf{s}_t|\mathbf{x}_t) - \log \sum_i p(\mathbf{s}_t|\boldsymbol{x}_i), \tag{5.12}$$

where the summation is over the observations in the current sequence batch. The paper shows that this term keeps $\boldsymbol{s}_t$ predictable from the current image, whilst also keeping the latent representations diverse. Although this work also achieves model based RL in the representation space without reconstructing the original image state, their regularized VIB objective is different to our solipsistic consistent-contrast objective (equation 5.4). We leave detailed comparisons in both theory and practice to future work.

5.6 Conclusion

We introduced Solipsistic Reinforcement Learning, a model-based reinforcement learning framework that learns useful latent representations of the environment for planning and reward prediction, without constructing a generative model of the environment. Our work is consistent with the recent general trend away from modeling the dynamics of high dimensional spaces and towards learning models that more directly solve the task at hand.

In our experiments on learning from pixels we assume a stationary initial position for our episodes. Without this simplifying constraint, the temporal consistency of our learned latent representations suffered, which negatively impacts overall performance. This phenomenon may restrict the ability for the model to solve larger and more complex problems. Resolving this practical limitation should be explored in future work.

Whilst model-based reinforcement learning is arguably preferable to model-free alternatives, previous approaches do not learn in an end-to-end fashion and also require an explicit model of the environment. We hope therefore that we have shown that there is scope to solve reinforcement learning problems in

a model-based way, but without the downsides of requiring complex models of the environment.

Chapter 6

Active Preference Learning

Arguably one of the more impactful practical breakthroughs in deep learning research over the last decade has been the scaling up of foundation models like large language models (LLMs) in terms of training data and model sizes. The scaling laws [27] have resulted in emergent properties of reasoning and these models are fast replacing our traditional search engines as a go to source of societal knowledge. This is particularly apparent in the case of ChatGPT by OpenAI, which has reportedly become one of the fastest growing consumer software products ever [169].

Such large language models are based on variants of the transformer architecture as introduced in section 2 trained using the back-propagation procedure as discussed in section 2.3. These auto-regressive models are first trained in an unsupervised manner to do next token prediction on vast troves of data predominately collected on the internet. Fundamentally this objective produces compression pressure on a system with limited capacity; therefore in order to get good at predicting the next token for arbitrary sequences of tokens on the internet, the model is forced to encode underlying latent concepts and reasoning procedures. This results in unprecedented capabilities in zero-shot and few-shot learning [26, 170]. Transfer learning is then adopted to make these base models more practical; they are generally subsequently fine-tuned on human preference data.

As large language models (LLMs) become more capable, fine-tuning tech-

niques for aligning with human intent are increasingly important. A key consideration for aligning these models is how to most effectively use human resources, or model resources in the case where LLMs themselves are used as oracles. Our research question is how can we more effectively use these resources during fine-tuning in a way that is simple and practical to implement?

Reinforcement learning from Human or AI preferences (RLHF/RLAIF) is the most prominent example of a fine-tuning technique [171], but is complex and often unstable. Direct Preference Optimization (DPO) is simpler and more stable alternative to RLHF [172]. In this chapter, we develop this idea further with a simple active learning strategy to make better use of preference labels. We propose simple acquisition functions for prompt/completion pairs based on the predictive entropy of the language model and a measure of certainty of the implicit preference model optimized by DPO. We demonstrate how our approach improves both the rate of learning and final performance of fine-tuning. In our experiments using open source models with up to ≈ 1 billion parameters, we see this approach improves final performance of the fine-tuned model by 3-7% on average over a random baseline.

6.1 RL from Human Feedback

The process of fine-tuning form human preferences is an important component to producing highly capable and generally helpful systems like ChatGPT. The most prominent class of fine-tuning technique is reinforcement learning from human feedback (RLHF) [171]. RLHF consists of a two stage process to adapt the pretrained autoregressive LLM $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$.

First, a reward model $r_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{y})$ is trained in a supervised manner on ranked pairwise preference data. For a given prompt $\boldsymbol{x}$, two completions are sampled from the model $(\boldsymbol{y}_0, \boldsymbol{y}_1) \sim p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ and an oracle rater judges which they prefer. We denote $\boldsymbol{y}_w$ as the preferred completion and $\boldsymbol{y}_l$ as the other. Typically the rater is a human participant, however the use of LLMs to instead provide feedback has also shown great promise [173]. This process is repeated over N

prompts resulting in the pairwise preference dataset $\mathcal{X}_P = \{\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l\}^N$. Second, a reinforcement learning (RL) algorithm such as Proximal Policy Optimization (PPO) [174] is used to fine-tune the parameters of the language model $\boldsymbol{\theta}$ by maximising the expected reward of completions as measured by $r_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{y})$. RL is used here because of the non-differentiability of sampling from $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$.

During the reward modelling phase in RLHF, the preference data is assumed to follow the Bradley-Terry (BT) model [175]. The objective for training the reward model can be framed as a binary classification task with a cross entropy objective:

$$\mathcal{L}_{\boldsymbol{\phi}}(\mathcal{X}_P) = -\mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}_w, \mathbf{y}_l) \sim \mathcal{X}_P}[\log \sigma(r_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{y}_w) - r_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{y}_l))]. \tag{6.1}$$

During the subsequent RL fine-tuning phase, the trained reward model is then used to score completions. PPO is used to update the parameters $\boldsymbol{\theta}$ by maximising the objective:

$$\mathbb{E}_{\boldsymbol{x} \sim \mathcal{X}, \boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}[r_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y})] - \beta \mathrm{KL}\Big(p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) || p_{\boldsymbol{\theta}_0}(\boldsymbol{y}|\boldsymbol{x})\Big). \tag{6.2}$$

The second term here regularises the fine-tuned model using the KL-divergence to stay close to the state of the LLM before fine-tuning $p_{\boldsymbol{\theta}_0}(\mathbf{y}|\mathbf{x})$. The main rationale provided for this is to prevent the model from deviating too far from the distribution on which the reward model is accurate. In practise the following reward function is used with PPO to update $\boldsymbol{\theta}$ [176, 177]:

$$r_{ppo}(\boldsymbol{x}, \boldsymbol{y}) = r_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{y}) - \beta(\log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) - \log p_{\boldsymbol{\theta}_0}(\boldsymbol{y}|\boldsymbol{x})). \tag{6.3}$$

A downside of RLHF is it's complexity; PPO introduces separate reward and value models that may be comparable in size to $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$, which are typically kept in memory during training. Furthermore PPO is found to have high variance and be sensitive to choices of hyper-parameters.

6.2 Direct Preference Optimization

Recently Direct Preference Optimization (DPO) has been proposed as a simpler and more stable alternative to RLHF [172]. DPO also depends on the collection of pairwise preference data, but crucially does not require first training an explicit reward model or the subsequent use of RL. Instead it relies on a straight forward binary cross entropy objective that increases the likelihood $\boldsymbol{y}_w$ and decreases the likelihood of $\boldsymbol{y}_l$. This approach implicitly optimizes the same objective as RLHF, without the added complexity.

DPO is originally derived from the optimal solution to 6.2; providing a maximum likelihood objective analogous to equation 6.1, but parameterised by $\boldsymbol{\theta}$ instead of $\boldsymbol{\phi}$ [172];

$$\mathcal{L}_{\boldsymbol{\theta}}(\mathcal{X}_P) = -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l)\sim\mathcal{X}_P}\left[\log\sigma\left(\beta\log\frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}_w|\boldsymbol{x})}{p_{\boldsymbol{\theta}_0}(\boldsymbol{y}_w|\boldsymbol{x})} - \beta\log\frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}_l|\boldsymbol{x})}{p_{\boldsymbol{\theta}_0}(\boldsymbol{y}_l|\boldsymbol{x})}\right)\right]. \quad (6.4)$$

We refer to this objective as having an implicit reward model:

$$\hat{r}(\mathbf{x},\mathbf{y}) = \beta\frac{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{p_{\boldsymbol{\theta}_0}(\mathbf{y}|\mathbf{x})}. \quad (6.5)$$

This formulation has the distinct advantage of not requiring the explicit reward modeling step and avoids the need to perform any reinforcement learning. Furthermore it has been shown to outperform RLHF across a range of benchmarks [172]. In practise the construction of $\mathcal{X}_P$ for DPO, including the preference labelling, is done upfront and stochastic gradient descent (SGD) is then used to fine-tune $\boldsymbol{\theta}$ according to equation 6.4 as introduced in section 2.3.

6.3 Active Preference Learning

Fine-tuning state-of-the-art LLMs using both of the aforementioned methods can require highly skilled domain experts, or expensive LLMs in the case of AI feedback, to produce the required preference data. In this section, we focus on how best to utilize the available preference labelling budget for training.

Instead of randomly sampling prompts during fine-tuning, we explore a more active sampling approach, which we refer to as Active Preference Optimization (APO). Informally, active learning is a paradigm in machine learning that aims to iteratively select the most useful datapoints during training using the current state of the model. Specifically we are interested in the setting of pool-based active learning which involves selecting a subset of observations from a closed pool of unlabeled data [178].

We use straight forward acquisition functions for prompt/completion pairs that leverage the predictive entropy of $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ and a simple proxy for the certainty of DPOs implicit preference model. Our APO training algorithm consists of iterations of the following scheme: randomly sample a large batch of prompts; generate pairs of completions for each prompt according to the latest version of the fine-tuned $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$; rank the prompt/completion pairs according to our acquisition function; select the highest ranking subset as a mini-batch of preference pairs for training; get preference labels on this actively selected mini-batch and, finally, fine-tune $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ using the labelled data before repeating the process until some preference labelling budget has been reached. We first outline in more detail our general active learning training procedure before introducing our acquisition functions for data selection.

This approach requires us to augment the existing DPO fine-tuning loop, which randomly samples mini-batches from a fixed preference labeled dataset, with an outer data acquisition loop. We compute the number of data acquisition steps K based on the acquisition batch size M and the overall labelling budget B. At each step S we randomly sample (without replacement) S prompts, generate completions using the current state of the model if required, then score the sampled datapoints using an acquisition function, where $M < S < N$ (where N is the total number of available prompts). We then select the highest ranking M datapoints to add to $\mathcal{X}_P$ before updating $\boldsymbol{\theta}$ with a round of fine-tuning. We specify our full process in algorithm 2.

Unlike the typical application of active learning in supervised learning

Algorithm 2 Active Learning Training Procedure

1: $\mathcal{X} \leftarrow \{\boldsymbol{x}\}^N$ ▷ initialise dataset of prompts
2: $\mathcal{X}_P \leftarrow \{\dots\}$ ▷ initialise empty preference labelled dataset
3: $K \leftarrow \lfloor \frac{B}{M} \rfloor$ ▷ compute number of acquisition steps
4: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$ ▷ initialise model weights
5: **for** $i = 1 \dots K$ **do** ▷ iterate through acquisition steps
6: $\mathcal{X}_S \coloneqq \{\boldsymbol{x}\}^S \sim \mathcal{X}$ ▷ randomly sample prompts
7: $\mathcal{X}_S \coloneqq \{\boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{x}\}^S \leftarrow \text{Generate}(\boldsymbol{\theta}, \mathcal{X}_S)$ ▷ generate completions
8: $\mathcal{X}_S \coloneqq \{\boldsymbol{s}, \boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{x}\}^S \leftarrow \text{Score}(\boldsymbol{\theta}, \mathcal{X}_S)$ ▷ score data using acquisition function
9: $\mathcal{X}_M \coloneqq \{\boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{x}\}^M \leftarrow \text{Subset}(\mathcal{X}_S)$ ▷ subset to highest scoring pairs
10: $\mathcal{X}_M \coloneqq \{\boldsymbol{y}_w, \boldsymbol{y}_l, \boldsymbol{x}\}^M \leftarrow \text{Oracle}(\mathcal{X}_M)$ ▷ get preference labels from orcale
11: $\mathcal{X}_p \leftarrow \mathcal{X}_p + \mathcal{X}_M$ ▷ expand preference dataset
12: $\boldsymbol{\theta} \leftarrow \text{Finetune}(\boldsymbol{\theta}_0, \boldsymbol{\theta}, \mathcal{X}_P)$ ▷ train using DPO until some stop critiera
13: EvaluateUsingOracle$(\boldsymbol{\theta}, \boldsymbol{\theta}_0)$ ▷ evaluate model on some held out test dataset
14: **end for**

settings, where at each acquisition step only the scoring of observations $\boldsymbol{x}$ is required, we have an additional step of also generating the corresponding completions for the acquired data. This is indicated by step 7 in our training procedure 2. This is required here if our choice of acquisition function requires access to completions, which we will discuss further in section 6.3.1. If this were not the case, it would still be required after step 9 when a subset of the data has been selected because our oracle needs access to completions for providing preference labels.

Although the choice of acquisition function is the primary concern of this study, there are some additional important design choices to consider for the practical implementation of this procedure. For example, how many fine-tuning training iterations to do at each acquisition? How many prompts to randomly sample and how many completions to generate? We will discuss these design choices further in appendix D.

6.3.1 Acquisition functions

In selecting scoring methods (step 8 in 2) we aim for options that are straightforward to implement and do not require modifications to the model architectures or the fine-tuning procedure itself. This allows for a drop in addition to existing implementations. As a result, we propose using the predictive entropy of

$p_{\boldsymbol{\theta}_t}(\mathbf{y}|\mathbf{x})$ as well as a measure of certainty under the Bradley-Terry preference model, which leverages the implicit reward model in DPO.

Entropy of the language model

Prior work has shown the predictive entropy (PE) to be a well calibrated measure of uncertainty in LLMs [179]. Therefore, if used as an acquisition function it will bias the fine-tuning process towards prompts the model is more uncertain about. The model represents a conditional distribution over possible completions. The predictive entropy is defined as:

$$\mathcal{H}_{p_{\boldsymbol{\theta}}}(\mathbf{y}|\mathbf{x}) = -\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})], \tag{6.6}$$

where this intractable integral can be approximated with Monte-Carlo samples in practise

$$\begin{aligned}\mathcal{H}_{p_{\boldsymbol{\theta}}}(\mathbf{y}|\mathbf{x}) &= -\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})] &(6.7)\\ &\approx -\frac{1}{N}\sum_{n=1}^{N}\log p_{\boldsymbol{\theta}}(\boldsymbol{y}^n|\mathbf{x}), &(6.8)\end{aligned}$$

where we calculate $\log p_{\boldsymbol{\theta}}(\boldsymbol{y}^n|\mathbf{x})$ by summing the log probability of each token in the completion.

Preference model certainty

The predictive entropy alone does not capture the extent to which the model accurately reflects oracle preferences, which is the ultimate goal of the fine-tuning process in this setting. To address this, we turn to characteristics of the Bradley-Terry model. We define a function we refer to as the certainty of the implicit preference model using $\boldsymbol{y}_1, \boldsymbol{y}_2 \sim p_{\boldsymbol{\theta}_t}(\mathbf{y}|\mathbf{x})$ that is maximised when the difference between the implicit rewards (see equation 6.5) for $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ is large and minimised when it's small. Specifically, during our scoring process (step 8 in algorithm 2) we determine the difference in our model's predicted

rankings for two different completions corresponding to the same input as

$$|\hat{r}(\boldsymbol{x}^i, \boldsymbol{y}_1^i) - \hat{r}(\boldsymbol{x}^i, \boldsymbol{y}_2^i)|. \tag{6.9}$$

We prioritize prompt/completion pairs with higher differences during the selection of data points for fine-tuning. Our hypothesis is that data points with high values provide valuable learning opportunities. Should the model's implicit preference predictions diverge from the oracle's evaluation, especially with high certainty, prioritising these discrepancies when fine-tuning can enhance model performance.

This choice is well motivated by the behaviour of the DPO training objective (equation 6.4). Consider the gradient update with respect to the parameters $\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}} = -\beta \mathbb{E}_{\mathcal{X}_P} \left[\boldsymbol{w} (\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_w | \boldsymbol{x}) - \nabla_{\boldsymbol{\theta}} \log p_{\theta}(\boldsymbol{y}_l | \boldsymbol{x})) \right], \tag{6.10}$$

where $\boldsymbol{w} = \sigma(\hat{r}(\boldsymbol{x}, \boldsymbol{y}_l) - \hat{r}(\boldsymbol{x}, \boldsymbol{y}_w))$ weights each sample $(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l) \sim \mathcal{X}_P$. This gradient update can be interpreted as weighting examples by how incorrectly the implicit reward model is while accounting for the strength of the KL constraint. Early in fine-tuning, when the implicit preference model is still likely to be wrong often, our proposed acquisition strategy prioritises examples that result in substantial gradient updates, which we find to accelerate learning progress and lead to an improvement in the final performance in our experiments in section 6.5.

A hybrid approach

In practise we can combine both entropy and preference certainty as complimentary metrics for scoring data to exploit the strengths of both. Our hypothesis is that higher entropy prompts are more likely to give incorrect predictions from the implicit preference model. In our experiments for this hybrid approach, we first select a relatively large batch of prompts and rank them by the entropy. We then take the top subset of prompts ranked by entropy and generate the required completion pairs before scoring and ranking according to preference

certainty. Finally, we take the top subset of prompt/completion pairs ranked by preference certainty and add them to our preference dataset for fine-tuning.

6.3.2 Choice of oracle

In step 10 and 13 of our algorithm 2 we require access to an oracle to provide preference judgements on pairs of completions; in step 10 to produce training data and step 13 to evaluate against a held out test data set. Given our desired setup of generating completions from the latest version of the model at each data acquisition step, we can't easily leverage prelabelled datasets here. Furthermore, given the need to re-run experiments multiple times with multiple different dataset, model, acquisition functions and seed (to get some measure of statistical significance) combinations, using humans to provide judgements is not feasible.

As we discuss in the experiments section 6.5, we do use relatively large open source language models to empirically test our hypotheses; however, the absolute performance of these models falls far short of the state-of-the-art models that have been trained for months with millions of dollars of compute by commercial entities. These models are only accessible to make inferences via black box APIs. This presents an opportunity to leverage these far superior models to provide the labels we need for the rather simple preference judgements. The questions then become; are these models good enough? Which model should we choose? How should we prompt?

We can look to recent research to answer the first question in the affirmative. Recent work has suggested that LLMs are superior oracles than existing metrics [180]. Of particular relevance is the LLM as an evaluator study carried out in [172] for the summarization task we also use in our experiments; they convincingly demonstrate that judgements form OpenAI's GPT-4, appropriately prompted, correlate strongly with humans. Furthermore, GPT-4 and human agreement is typically similar or higher than inter-human annotator agreement.

With this choice of oracle, our evaluation approach (on held out test prompts) is to use head-to-head win-rate comparisons versus completions sampled from

the model at the start of training $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$. In the cases where suitable human-provided completions are already provided on the hold out test data, we can instead use these to evaluate the models performance. We don't evaluate every acquisition step, because of how expensive in cost and time using GPT-4 as an oracle is. Depending on the experiment we evaluate at multiple appropriate way-points during training, or just at the end.

Oracle prompt

As LLMs get more broadly capable, the amount of so-called prompt engineering research is on the rise. Although not the focus of this work, we need to provide a prompt to our LLM based oracle. We follow the guidelines outlined in [172] given their robust results. We make one additional change, which is also asking the model to provide a rationale for it's reference. Qualitatively we found this to provide superior results on a handful of testcases; there is also good evidence to support that chain of thought-like prompting is an effective strategy to improve zero shot performance [181].

In our experiments we require two distinct oracle prompts: one for sentiment analysis and the other for summarization - see figure 6.1. In order to mitigate against any potential bias due to the ordering of results presented, we randomly change the ordering of the model completions presented to the oracle during evaluation and fine-tuning.

Choice of base model

A major downside of using GPT-4 as our oracle model is the expensive cost and high latency. In particular the cost can restrict the number of seeds we can afford to run and similarly the number of ablation studies. A far more economical choice would be to use older versions of models such as GPT-3.5. We therefore tested to see whether this would be sufficient.

We ran a simple test where we generated preference labels twice for both GPT-3 and GPT-4 on a set of 50 prompts and completions sampled sampled from the fine-tuning from human preferences dataset [176]. Unfortunately we found that GPT-4 was far more consistent (>90%) than GPT-3.5-turbo (only

```
// SENTIMENT ORACLE PROMPT

<SYSTEM>
You are a helpful assistant that evaluates the quality and
positive sentiment of movie reviews
</SYSTEM>

<USER>
Which of the following movie reviews is better? The best one
will be the one with the most positive sentiment, which also
is grammatically correct, consistent, and avoids repetition.

Review A:
{{PROMPT}} {{COMPLETION-A}}

Review B:
{{PROMPT}} {{COMPLETION-B}}

First, provide a one-sentence comparison of the two reviews,
explaining which is better and why. Second, on a new line,
state only "A" or "B" to indicate your choice.

You must choose A or B for the preferred answer even if
neither review is very good.

Your response should use the format:
Comparison: <one-sentence comparison and explanation>
Preferred: <"A" or "B">
<\USER>
```

```
// SUMMARIZATION ORACLE PROMPT

<SYSTEM>
You are a helpful assistant that evaluates the quality of
summaries for internet posts.
</SYSTEM>

<USER>
Which of the following summaries does a better job of
summarizing the most important points in the given
forum post, without including unimportant or irrelevant
details?

Post:
{PROMPT}

Summary A:
{COMPLETION_A}

Summary B:
{COMPLETION_B}

First, provide a one-sentence comparison of the two
summaries, which you prefer and why. Second, on a new line,
state only "A" or "B" to indicate your choice.

You must choose A or B for the preferred answer even
if neither summary is very good.

Your response should use the format:

Comparison: <one-sentence comparison and explanation>
Preferred: <"A" or "B">
<\USER>
```

Figure 6.1: GPT-4 oracle prompts for sentiment and summarization tasks

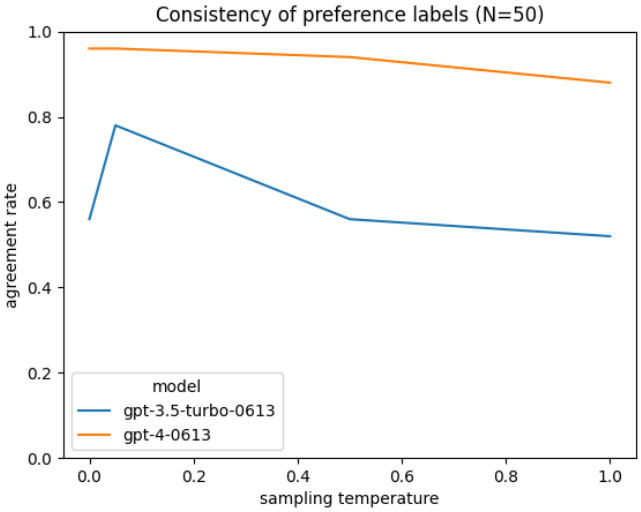


Figure 6.2: Self-consistency of preference labels provided by GPT-3 and GPT-4 across 50 prompt completion pairs. Each model provided two preference labels for each prompt completion pair.

$\sim$60%) at a range of sampling temperatures - see figure 6.2. We therefore chose to use GPT-4 as the oracle for our experiments and adjusted our budget of evaluations appropriately to contend with the cost.

Completion sampling

Another important aspect of evaluating language models is the strategy used for getting completions from the model given auto-regressive probabilistic model; often referred to as the decoding strategy. The choice of decoding method can impact the quality and characteristics of the model's output. Common decoding strategies include greedy, beam search, top-k sampling, nucleus sampling, and temperature-scaled sampling, among others. Greedy decoding just selects the most probable next token at each step, which can lead to repetitive and deterministic outputs. Beam search maintains multiple hypotheses at each step and can produce more diverse and coherent outputs, but may still suffer from a lack of diversity and can be computationally expensive. Top-k sampling restricts the next token to be sampled from the k most probable tokens, introducing randomness into the output. Nucleus sampling, or top-p sampling, takes this further by sampling from the smallest set of tokens whose cumulative probability exceeds a threshold p, allowing for a dynamic number of tokens to be considered based on the model's confidence.

In our experiments we leverage temperature-scaled sampling that adjusts the probability distribution over the next token by scaling the logits before applying the softmax function. A temperature parameter T controls the degree of scaling. A low temperature ($T < 1$) sharpens the distribution, making the model more confident and conservative in its predictions, often leading to less diverse outputs. A high temperature ($T > 1$) flattens the distribution, increasing diversity in the output by making less probable tokens more likely to be chosen. A temperature of zero ($T = 0$) effectively turns the sampling into greedy decoding. In our experiments to follow we use $T = 0.7$ for $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ during training, $T = 0.2$ during testing (to encourage lower variance) and $T = 0.05$ for the GPT-4 oracle to promote deterministic oracle judgements.

Fine-tuning details

Here we discuss in more detail the implementation details for the fine-tuning step (12) in algorithm 2. We adopt the most straight-forward implementation,

which is to re-initialise θ_t to θ_0 at each time step t and fine-tune to convergence, sampling uniformly from all previously acquired preference data $\mathcal{X}_p$. This is consistent with previous work on deep active learning [182] and relies on the assumption that the cost (in time and/or money) of acquiring oracle labels outweighs the cost of fine-tuning again on all acquired data after each new batch of labels is acquired. The focus of our main experiments in section 6.5 is to isolate the differences in performance caused by the different acquisition vs randomly acquiring data. In Appendix D.4, we discuss adapting our approach for online learning and present some provisional results.

6.4 Related Work

Our work is closely related to Direct Preference Optimization [172] which we leverage as our fine-tuning algorithm of choice. We augment the training process with an additional data acquisition and fine-tuning loop as outlined in algorithm 2. The random baseline in our experiments is equivalent to the DPO procedure.

There are numerous recent research efforts in exploring how a more active learning setup can improve fine-tuning LLMs, but don't use DPO as a basis. The Reward rAnked FineTuning (RaFT) technique [183] introduces an online training procedure that ranks, using an oracle reward model, multiple completions for each prompt; selecting the top performers to use in a traditional supervised fine-tuning process. That is; maximising the likelihood of the best performing completions for each prompt. Once training is complete, they randomly sample a new batch of data, then re-generate completions form the latest version of the trained model and repeat the ranking/filtering and training step. Like DPO, this approach does not require the use of reinforcement learning for updating the parameters of the model. Unlike our approach, RaFT consults the oracle on every data point before filtering for the subset that will be used during training; therefore is not trying to make better use of the oracle resource.

Another orthogonal application of active learning in the setting of improving

pre-trained LLM performance is the active sampling of few shot examples for prompt stuffing [184]. In this work, the authors use acquisition functions based on different uncertainty, diversity and similarity scores of the language model across datasets of few-shot examples to determine which examples are best to reference in the prompt to improve performance. Although similar in spirit to our work, they don't consider updating the parameters of the model using preference-labelled data.

An alternative active learning approach is data pruning. In [185], pruning heuristics are applied to filter the data used in the first stage of unsupervised LLM pre-training. This leads to improved performance on downstream tasks versus the LLMs pre-trained on the full dataset. Over 50% of the data can be pruned while still leading to improvements. This work does not directly consider the impact of such pruning techniques for the preference fine-tuning stage, but some of their perplexity based heuristics could represent viable alternatives to our acquisition strategies.

Finally, a research theme adjacent to active learning that can also reduce the amount of preference labels required is that of self-play fine-tuning [186, 187]. These works focus on how to bootstrap $p_{\boldsymbol{\theta}_t}(\mathbf{y}|\mathbf{x})$ during fine-tuning to provide preference labels, or to act as a reward model, as opposed to trying to make better use of oracle resources. This in principle could be combined with our active preference learning approach and so we consider it complimentary.

6.5 Experiments

The focus of our experiments is to determine if more active sampling during the fine-tuning process can bring us gains in data efficiency when dealing with limited labelling budgets; in terms of the rate of learning and the final performance achieved. We compare four different acquisition configurations: random, entropy, certainty and entropy + certainty (as discussed in section 6.3.1). We evaluate across two different open source large language models and two different datasets used in recent related work. We also gather some

qualitative findings about the characteristics of the datapoints being acquired under the different schemes, which we discuss further in 6.5.5.

6.5.1 Datasets

In line with recent work [176, 172] we focus on two distinct datasets for our experiments; IMDB and TLDR. IMDB is a dataset of movie reviews where the task is to complete a positive review given the start of a review. TLDR is a dataset of Reddit posts where the task is to provide a summary of the post. Table 6.1 provides a summary of the dataset details.

Table 6.1: Preference learning datasets summary

	IMDB	**TLDR**
Train size	25,000	117,000
Test size	25,000	6,550
Task	Complete reviews according to preference	Generate summaries according to preferences
Data source	`https://huggingface.co/datasets/imdb`	`https://huggingface.co/datasets/CarperAI/openai_summarize_tldr`

The choice of oracle for providing labels and evaluating on the test data is GPT-4; details of the prompts are provided in the previous section 6.3.2. Our prompts specify a task specific preference, but also consider grammatical correctness and consistency. We provide further details on dataset pre-processing in appendix D.

6.5.2 Models

For both IMDB and TLDR we use relatively large transformer based architectures. See table 6.2 for a summary of the models and main hyper-parameters used in both cases. For IMDB, the GPT-2 base transformer model provided by Hugging Face[1] was pre-trained on the WebText corpus [188] and has 12 layers with 768 dimensions, with 12 attention heads. It was also further trained in an unsupervised way on the full IMDB dataset. For TLDR, we use the Pythia[2]

[1]Downloaded pre-trained base model from `https://huggingface.co/edbeeching/gpt2-large-imdb`

[2]Pre-trained base model from `https://huggingface.co/pvduy/pythia-1B-sft-summarize-tldr`

class of transformer model [189] that has an architecture similar to GPT-3, with 805M parameters, 16 layers with 2048 dimensions and 8 attention heads. We ran our fine-tuning on single 40GB RAM A100 and 48GB 600 ADAs GPUs throughout our experiments.

6.5.3 Acquisition sampling

Given we follow a pool-based active learning approach we assume access to an abundant supply of prompts to choose from during fine-tuning. In practise we have two steps to consider for filtering the data - after the initial selection of prompts (step 6 in algorithm 2) and after completions have been generated (step 7). In the latter case, more information is available, but require potentially expensive completions.

In our experiments we first randomly sample $S = 4000$ for IMDB and $S = 2048$ for TLDR for our entropy only and preference certainty only acquisition runs. When doing entropy + preference certainty, we first randomly sample $J \times S$ prompts, rank them by entropy and take the top S prompts to generate completions before further scoring and ranking by preference certainty. We use $J = 8$ for IMDB and $J = 4$ for TLDR. We use $N = 8$ samples when approximating the entropy. For all experiments we set the final acquisition batch size to $M = 128$.

6.5.4 Evaluation

We use GPT-4 as the oracle for providing labels and evaluating the test data. Details of the prompts are provided in section 6.3.2. Our prompts specify a task-specific preference but also consider grammatical correctness and consistency. Our evaluation approach on held-out test prompts uses head-to-head win-rate comparisons versus completions sampled from the pre-trained model from the start of training $p_{\boldsymbol{\theta}_0}(\mathbf{y}|\mathbf{x})$ for IMDB. For TLDR, we replaced the pre-trained model completions with the human-provided completions that were available on the hold-out test data. Due to the significant cost of using GPT-4 as the oracle for evaluation, we don't evaluate after every single data acquisition step.

Each evaluation is done against 1024 test prompts.

Table 6.2: Preference learning models summary

	IMDB	**TLDR**	**Comment**
Model used	Pre-trained GPT-2 [10]	Pre-trained Pythia [189]	
Parameter size	774M	805M	
Optimizer	ADAM lr: 1e-06	ADAM lr: 1e-06	as per DPO [172]
Finetuning Epochs	50	70	See appendix D.2
Mini-batch size	64	64	For fine-tuning
Prompt batch size (S)	4000	2048	
Acquisition batch size (M)	64	64	Out of 512 prompts scored
β	0.2	0.2	as per DPO [172]
Model source	`https://huggingface.co/edbeeching/gpt2-large-imdb`	`https://huggingface.co/pvduy/pythia-1B-sft-summarize-tldr`	

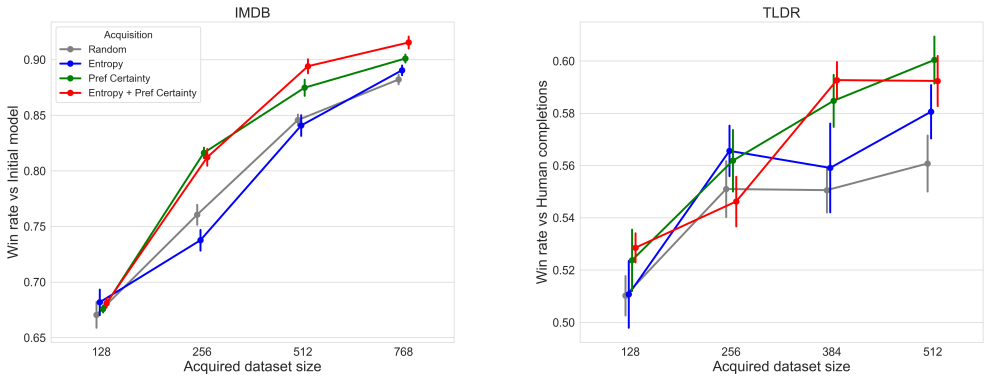
6.5.5 Results

We run our active learning procedure (algorithm 2) to fine-tune the models discussed in the previous section against IMDB and TLDR. The overall data acquisition, fine-tuning and evaluation processes are repeated for 9 different random seeds. Figure 6.3 and table 6.3 contain the detailed win-rate results of each configuration. The cost associated to evaluating using GPT-4 limited the number of data acquisition steps we could practically carry out, therefore we focused on doing more seeds on fewer numbers of data acquisition steps to aid in drawing conclusions.

Overall we find that our certainty acquisition function outperforms random and entropy, improving win-rate performance by between 1-6% on average. This provides evidence in favour of our hypothesis discussed in 6.3.1 that prompts with higher differences in the implicit rewards corresponding to their completions provide valuable learning opportunities. We find that combining preference certainty with entropy gives a small improvement for the larger

acquisition batch sizes (512, 768) on IMDB, but this result is not consistent across both datasets. Given these results and the additional complexity due to the Monte Carlo estimation of the entropy, we recommend the preference certainty acquisition as a simple acquisition strategy to use in practise.

For the first fine-tuning step ($M = 128$), there is no discernible difference between the strategies. This makes sense when using the preference certainty acquisition because the initial pre-trained model is used to rank the data and it doesn't yet know anything about the oracle's preferences. In Appendix D.3 we provide examples of typical prompt and completion pairs, alongside the oracle preference and rationale provided by our GPT-4 oracle, before and after the fine-tuning process.



(a) IMDB Win-Rate **(b)** TLDR Win-Rate

Figure 6.3: Win-rate at evaluation waypoints. (a) IMDB is win-rate vs the initial model.(b) TLDR is win-rate vs human provided summaries on the test prompts (b). The x-axis is the size of the acquired dataset used for fine-tuning at the point of evaluation. Each model and dataset pair was trained with 9 random seeds and we plot means with standard errors. Preference certainty and entropy + preference certainty outperform the random baseline.

Analysing acquired data

In section 6.3.1 we motivate why the preference certainty acquisition strategy may provide an advantage versus a random baseline when fine-tuning with DPO. This focused on whether it would surface examples where the implicit

Table 6.3: Active preference learning results: the mean to 2 d.p. and standard errors to 3 d.p. of the win-rates. For IMDB, we calculate the win-rate vs the completions generated by the initial pre-trained model. For TLDR we calculate the win-rate vs the human completions available on the test set. The size column represents the size of the acquired dataset used for fine-tuning at the point of evaluation.

Dataset	Size	Random	Entropy	Pref certainty	Pref + Ent
IMDB	**128**	0.67 ± 0.012	0.68 ± 0.011	$\mathbf{0.68} \pm 0.003$	$\mathbf{0.68} \pm 0.004$
	256	0.76 ± 0.008	0.74 ± 0.009	$\mathbf{0.82} \pm 0.005$	0.81 ± 0.007
	512	0.84 ± 0.004	0.84 ± 0.009	0.87 ± 0.007	$\mathbf{0.89} \pm 0.006$
	768	0.88 ± 0.004	0.89 ± 0.004	0.90 ± 0.004	$\mathbf{0.92} \pm 0.005$
TLDR	**128**	0.51 ± 0.008	0.51 ± 0.013	0.52 ± 0.012	$\mathbf{0.53} \pm 0.006$
	256	0.55 ± 0.01	$\mathbf{0.57} \pm 0.01$	0.56 ± 0.012	0.55 ± 0.01
	384	0.55 ± 0.009	0.56 ± 0.017	0.58 ± 0.01	$\mathbf{0.59} \pm 0.007$
	512	0.56 ± 0.012	0.58 ± 0.01	$\mathbf{0.60} \pm 0.009$	0.59 ± 0.01

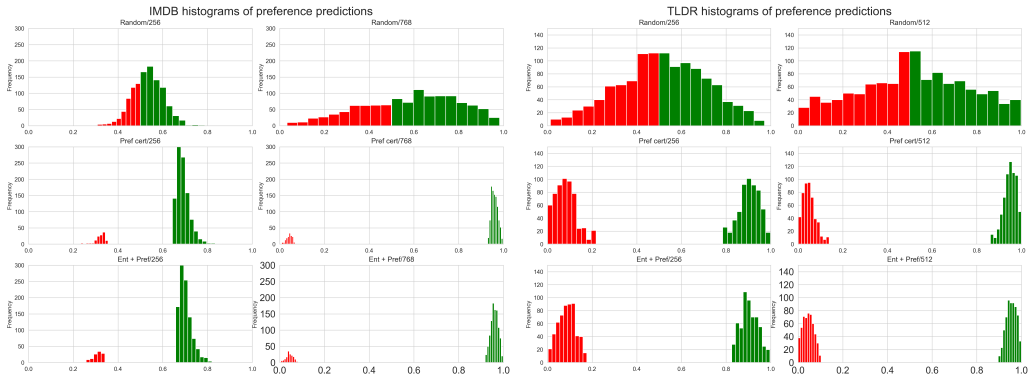


Figure 6.4: Histograms of probabilities from our implicit Bradley Terry preference model across a batch of acquired data; grouped by incorrect (red) and correct (green) preferences according to the oracle. This assumes a decision threshold of 0.5. Our preference certainty acquisition function surfaces confidently with wrong examples.

preference model provided an incorrect prediction, with certainty. We carry out a post hoc analysis of the data acquired during our experiments to better understand the characteristics of the acquired examples. In particular, what differs between the different acquisition strategies and how they change as fine-tuning phases progress. The approach we take is to look at how the implicit preference predictions from the model correlate with the true oracle preferences.

We construct a classifier using the Bradley Terry (BT) model - equation 6

in [172] - that gives us $p(\mathbf{y}_1 \succ \mathbf{y}_0|\mathbf{x}) \in [0, 1]$ under our implicit reward model (equation 6.5). Using the probabilities provided, we construct histograms in figure 6.4 for the batches of M acquired datapoints across all 9 seeds. We map the data in such a way that the bucket at 0.9 will contain examples where the BT model was most confidently correct according to it's probability, and 0.1 will contain the most confidently wrong. The red $0.0 \rightarrow 0.5$ contains all the incorrect predictions bucketed into 10 bins according to their probability. The green $0.5 \rightarrow 1$ contains all correct predictions. To determine correctness, we use a 0.5 decision threshold on our BT model and compare the result to the ranking provided by the oracle.

We can see from these histograms that the random acquisition selects quite uniform examples according to the implicit preference model predictions. The preference certainty-based acquisition on the other hand surfaces a lot of confidently incorrect examples which ultimately aid with improving fine-tuning performance when using DPO.

6.6 Conclusion

We've demonstrated a simple and effective way to improve the use of an oracle labelling budget for preference fine-tuning LLMs. Our active learning setup builds upon DPO and uses the implicit preference model to determine which data points to get oracle judgements for during training.

There are a couple of noteworthy limitations to address in future work:

- **Reducing the model training budget:** Although we are focused here on making better use of the oracle labelling budget, as opposed to the overall training computation budget, it is worth also considering for practical reasons how the latter may be improved. Currently after each data acquisition step, we re-initialise the parameters of the model to the unsupervised pre-trained LLM parameters and do a full fine-tuning process before re-acquiring data. This will get prohibitively slow and expensive the larger the models and datasets become.

  One potential option to explore to reduce the training computation budget is to not reset the parameters of the model at each data acquisition step and only do a much smaller number of fine-tuning epochs. In Appendix D.4, we discuss adapting our approach for online learning and present some provisional results. Considerations for how to balance the amount the model trains on previously acquired datapoints will be important. Ideally a Bayesian formulation could be considered; using an online update to the posterior over the parameters in the face of newly acquired preference labels. Such that we don't need to consider previously acquired data points. The challenge here will be how to effectively compute the posterior distribution and updates for a model of this size. Tactics like only considering a subset of parameters (say last layer) may be helpful.

  Another potential option is to continue to reset the parameters after each data acquisition step, but only consider fine-tuning a subset of model parameters so that the fine-tuning iterations are much faster and less computationally burdensome. Combining our approach with parameter

efficient fine-tuning methods like LORA [190] and Q-LORA [191] are reasonable options.

- **Hyper-parameter calibration:** Due to the significant cost of evaluation (using GPT-4) and computation time for a full cycle of training (on A100 GPUs) we were limited in our ability to do ablation studies on hyper-parameters in this study. Instead relying on default values from existing work. For instance, future work with additional budget should consider ablation on optimizer choice, learning rate, fine-tuning stopping criteria and more random seeds for improving the statistical robustness of the results.

- **More scale:** To prove out that these techniques will have material impact on training SOTA LLMs, we would need to move into the multi-billion parameter regime. This would require distributed training across multiple GPU nodes and more conservative choice of training and evaluation configurations given the cost.

In addition, there are also multiple avenues of potential future work we are exploring:

- **More sophisticated acquisition strategies:** Our preference model certainty acquisition function is simple and our results provide evidence that it is effective. Combining the merits of multiple strategies, such as entropy and certainty and adding additional criteria around diversity of prompts and completions could are also worth exploring to further improve efficiency. Furthermore, more explicit treatment of the epistemic uncertainty of the model when acquiring prompts may also be useful.

- **Surrogate preference model and self-supervision:** One potential avenue to further save on oracle labels is to bootstrap the predictions of our implicit preference modoel, or an explicit surrogate preference model we train on the labels acquired to date. An interesting challenge here is

to determine an acquisition strategy that selections data points where we should trust the model predictions.

- **Broader applications:** Our proposed techniques are applicable to any generative model that can be fine-tuned using a DPO like objective where preference pairs are used in a MLE objective. Moving beyond the text modality to images, sound and multi-modal applications are promising avenues for future work.

- **Beyond binary feedback:** As part of our model-based evaluation framework discussed in section 6.3.2 we are generating both a preference label and a natural language rationale. The primary purpose of getting the oracle GPT-4 model to also generate a rationale is that this has been shown to improve the performance of the model at providing preferences. An interesting idea is to update our MLE objective to jointly maximise the likelihood of the rationale alongside maximising the likelihood of the preferred objective and minimising the likelihood of the less preferred completion for a given prompt. This may provide more information about the latent preferences to the model being fine-tuned and allows us to get away with less preference labels. It may also help regularise the model into still producing valid language when preference fine-tuning, potentially allowing us to remove the KL regularisation to the initial pre-trained model in equation 6.4; saving on memory and computation during fine-tuning.

Chapter 7

Conclusions

The overarching goal of this thesis was to improve on the data efficiency of deep learning systems, which are becoming increasingly data and resource hungry [27]. Within this broad theme we presented methodological contributions across the different learning paradigms of supervised learning (chapter 3), unsupervised learning (chapter 4), reinforcement learning (chapter 5) and preference learning (chapter 6). As discussed with examples in chapter 1, informally data efficiency has a different goal in each of these learning categories. In supervised and preference learning data efficiency is mainly concerned with minimizing the number of queries to an oracle to label data. Implicit in this is the assumption that the oracle labels (either human generated, or other model generated) are expensive in time and or cost. In unsupervised learning data efficiency is mainly concerned with minimizing the amount of unlabelled training data required. In reinforcement learning, the goal is the minimize the number of interactions required with the environment.

In chapter 3 we introduced a weak learning framework that jointly trains a label model and end-model. We illustrated that this can significantly improve performance versus existing 2-stage methods [72, 1, 73, 75] across a range of natural language classification benchmark tasks [39]. We also presented a parameterization of the label model that can produce similar mappings between the latent and true label (the transitions $\boldsymbol{\phi}_k$) for similar datapoints $\boldsymbol{x}$ and amortises the cost, using a neural network, of computing this for each

datapoint. There are many promising avenues of future work in this direction. We can naturally combine our weak learning framework with active learning to further improve data efficiency. This would entail actively sampling mini-batches of data (like in chapter 6) in order to reduce the epistemic uncertainty w.r.t both the label model and end model parameters; considering both weak and strong labels in the process. Another promising direction is to combine this weak learning approach with the work in chapter 6 on preference fine-tuning LLMs; to aggregate preference data from different sources - both Human and AI based preference data of different qualities.

In chapter 4 we focused on arguably a relatively under-studied problem of generalization in VAE style models. VAEs have been shown to be effective in many important problems wherein acquiring training data is challenging; such as in chemistry and biology research, when models are trained on existing molecular structures [33, 34]. We demonstrated that this type of model does suffer significantly from over-fitting. We framed this in context of two distinct generalization gaps: the generative model and the amortized inference generalization gaps, where the over-fitting was dominated by the amortized inference neural network. We proposed an inference consistency requirement, which, if satisfied, would result in optimal generalization performance for this class of model. We demonstrated how the default training objective does not satisfy this, but that the wake-sleep training algorithm [118] does. We then introduced an alternative sleep-type training step for updating the parameters of the inference neural network after normal training that, unlike the traditional wake-sleep setup, directly improves the ELBO. This approach bootstraps predictions from the generative model to improve the generalization performance of the inference network. We provided empirical evidence on image modelling and compression tasks that our approach is a useful addition to existing VAE training pipelines to improve generalization performance. Natural next steps for future would involve applying our method to other applications where VAEs have shown promise; such as molecular design [133], combining it with methods

to also reduce the generative model generalization gap [129] and extending it to cater for more hierarchical latent variable structures [134].

In chapter 5 we then turned our attention to the problem of reinforcement learning; in particular model-based reinforcement learning. We present ideas for a model that learns a low-dimensional representation of the target environment while avoiding the need to learn a full generative model of the environment, which prior works have required. We explore different choices of model and planning algorithms. On the model front, we introduce a memory model that aids the ability of the model to accurately track long term behaviour in it's latent dynamics. We build up an overall learning objective for our model using three desiderata around consistency in predicted dynamics, consistency in predicted rewards and the effective filtration of redundant observation state information. Future research in this area should first explore scaling up the proposed learning framework to larger and more complex learning from pixel RL problems and addressing any practical limitations around the initialization of the learned representations.

Finally in chapter 6 we tackle the problem of making more efficient, in terms of oracle usage, aligning large pre-trained language models to human preferences. We first discussed direct preference optimization as an alternative to RLHF [171, 172] before then presenting an active learning process that requires generating completions with the latest version of the model for every data acquisition step. Of independent interest is the use of LLM based oracles for both generating training labels and for evaluation; without which our experiments would not have been tractable. We demonstrated that using our certainty metric of the implicit reward model as a simple choice of acquisition function improved win-rate performance by $3-7\%$ in our experiments over a random baseline. We discussed multiple promising avenues of future work. To improve the practical utility, it's important to explore approaches to minimise the computation budget required during the active fine-tuning process; for example by combining our approach with parameter efficient fine-tuning methods like LORA and Q-

LORA [190, 191], or taking more of an online Bayesian learning perspective. Of particular interest is moving beyond just binary and categorical feedback types to also leverage the rationale produced by our oracle; we believe this learning from more natural feedback to be an important next step in AI alignment.

Our hope is that the methods introduced in chapter 3 and 4 can help practitioners improve the data efficiency of their existing deep learning systems in practise as well as inform future research, and that chapters 5 and 6 represent effective stepping-off points for further research in this area.

Appendix A

Integrated Weak Learning

A.1 Further ablation

In this section we provide results for the ablation on how to include strong labels when available in the two-stage baseline methods, namely Snorkel and Majority Vote (MV). The standard approach is to add an additional labeling function that outputs the strong label if available and abstains otherwise. We refer to this as the voting variant (-V). This is what we use in the results reported in section 3.4. An alternative approach we also consider is to leave the labeling functions unchanged and instead replace the resulting denoised label with the corresponding strong label if available when training the end-model $p_{\boldsymbol{\theta}}(y|\mathbf{x})$. We refer to this as the strong variant (-S). Figure A.1 and the corresponding table A.1 compare Snorkel-V, Snorkel-S, MV-V and MV-S to our model variant iWLD across the same 6 datasets for different splits of available strongly labeled data. We can see that the -S variants in fact consistently outperform the -V variants. Furthermore our iWLD approach is the best performing method in 28 out of the 30 cases.

A.2 Further experiment details

All experiments were run on a GPU cluster with access to 10 V100 GPU processors. Each experiment run was executed on a single GPU instance. The code we have made available for producing our experimental results is implemented using PyTorch [192].

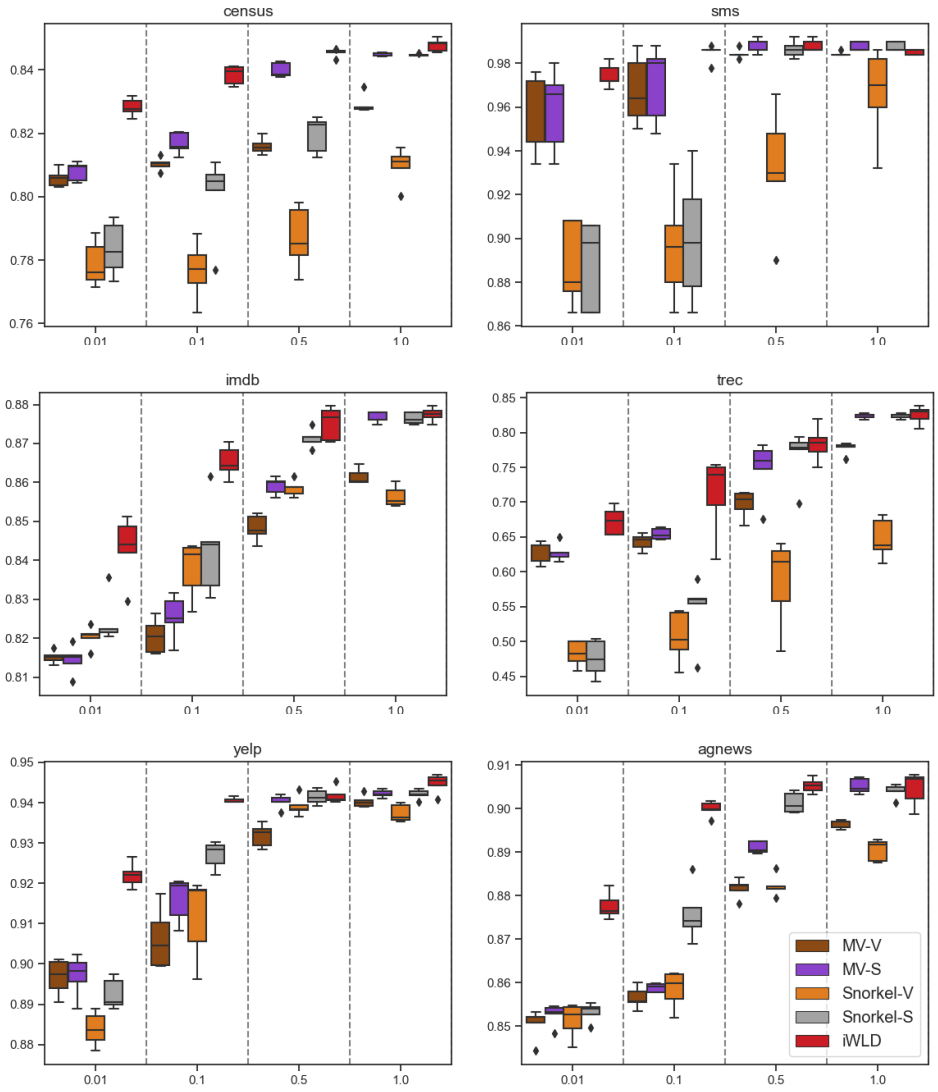


Figure A.1: Box-plots of test F1 scores showing the quantiles across 5 random seeds on the y-axis. Each plot refers to a specific dataset. The model variants discussed in section A.1 are grouped by the different fractions of strongly labeled data available on the x-axis (1%, 10%, 50%, 100%). See table A.1 for more detail.

The specific train, validation and test data splits used across all datasets are available in a standardized schema at the following URL: `https://drive.google.com/drive/folders/1VFJeVCvckD5-qAd5Sdln4k4zJoryiEun`. This is provided as part of the WRENCH benchmark [39]. Further information on how this data was gathered, the original source, and the relevant attributes is available in WRENCH [39].

Table A.1: Test F1 score averaged over 5 random seeds with 1 standard deviation in brackets across all datasets and model variants discussed in section A.1. The results are grouped by the different proportions of strongly labeled data available (1%, 10%, 50%, 100%). Values highlighted with **bold** indicate the best performing method.

Labels	Model	Agnews	Census	IMDB	SMS	TREC	Yelp
1%	MV-V	85.05 (0.36)	80.59 (0.28)	81.52 (0.16)	95.40 (1.88)	62.88 (1.58)	89.67 (0.45)
	MV-S	85.27 (0.25)	80.80 (0.30)	81.45 (0.38)	95.88 (1.91)	62.84 (1.34)	89.71 (0.52)
	SKL-V	83.45 (3.78)	77.89 (0.73)	82.03 (0.28)	88.76 (1.93)	48.24 (1.82)	88.38 (0.43)
	SKL-S	84.23 (2.47)	78.36 (0.85)	82.43 (0.63)	88.84 (2.07)	47.56 (2.67)	89.25 (0.38)
	iWLD	**87.77** (0.31)	**82.82** (0.29)	**84.31** (0.84)	**97.44** (0.55)	**67.32** (1.95)	**92.21** (0.31)
10%	MV-V	85.65 (0.25)	81.03 (0.21)	82.05 (0.44)	96.76 (1.60)	64.28 (1.19)	90.63 (0.76)
	MV-S	85.89 (0.10)	81.68 (0.34)	82.54 (0.57)	97.08 (1.76)	65.44 (0.82)	91.61 (0.56)
	SKL-V	84.80 (2.36)	77.67 (0.94)	83.78 (0.73)	89.64 (2.60)	50.64 (3.73)	91.16 (1.03)
	SKL-S	87.59 (0.64)	80.03 (1.34)	84.29 (1.22)	90.00 (2.99)	54.56 (4.88)	92.71 (0.34)
	iWLD	**90.00** (0.18)	**83.84** (0.30)	**86.53** (0.42)	**98.48** (0.39)	**71.16** (5.72)	**94.06** (0.06)
50%	MV-V	88.17 (0.22)	81.59 (0.26)	84.82 (0.34)	98.44 (0.22)	69.72 (1.98)	93.18 (0.28)
	MV-S	89.11 (0.14)	83.99 (0.23)	85.91 (0.23)	98.76 (0.33)	74.80 (4.23)	94.03 (0.17)
	SKL-V	88.23 (0.24)	78.69 (1.01)	85.85 (0.21)	93.20 (2.84)	58.56 (6.41)	93.92 (0.25)
	SKL-S	90.14 (0.24)	81.96 (0.57)	87.14 (0.23)	98.64 (0.38)	76.64 (3.89)	94.15 (0.18)
	iWLD	**90.53** (0.17)	**84.55** (0.13)	**87.52** (0.43)	**98.80** (0.28)	**78.40** (2.58)	**94.18** (0.21)
100%	MV-V	89.65 (0.09)	82.93 (0.30)	86.15 (0.21)	98.44 (0.09)	77.76 (0.89)	94.03 (0.16)
	MV-S	**90.53** (0.18)	84.50 (0.06)	87.66 (0.14)	**98.76** (0.22)	82.40 (0.40)	94.23 (0.09)
	SKL-V	89.05 (0.25)	80.97 (0.58)	85.64 (0.27)	96.60 (2.16)	64.76 (2.95)	93.74 (0.22)
	SKL-S	90.40 (0.16)	84.48 (0.04)	87.64 (0.15)	**98.76** (0.22)	82.40 (0.40)	94.21 (0.12)
	iWLD	90.46 (0.39)	**84.78** (0.20)	**87.74** (0.18)	98.52 (0.11)	**82.52** (1.25)	**94.47** (0.24)

A.3 Visualizing the label model

In section 3.2.1 we proposed a label model that included a dependency on $\mathbf{x}$ (see figure 3.1b). Figure A.2 plots an array of different transitions $\boldsymbol{\phi}_k^n$, after training, for different labeling functions k and different datapoints n. These are sampled from the TREC dataset (number of class labels $C = 6$ and number of labeling functions $K = 68$). This helps illustrate that our model has in fact learned different transitions for different datapoints.

A.4 Dependent labeling functions

There is a straightforward extension to our model that may be well suited to where we have multiple dependent labels $\tilde{y}_1, \ldots, \tilde{y}_K$. In principle, the labeling functions are conditionally independent given $\boldsymbol{x}$. To take information from the weak labels, our assumption in section 3.2 is that we can explain the weak

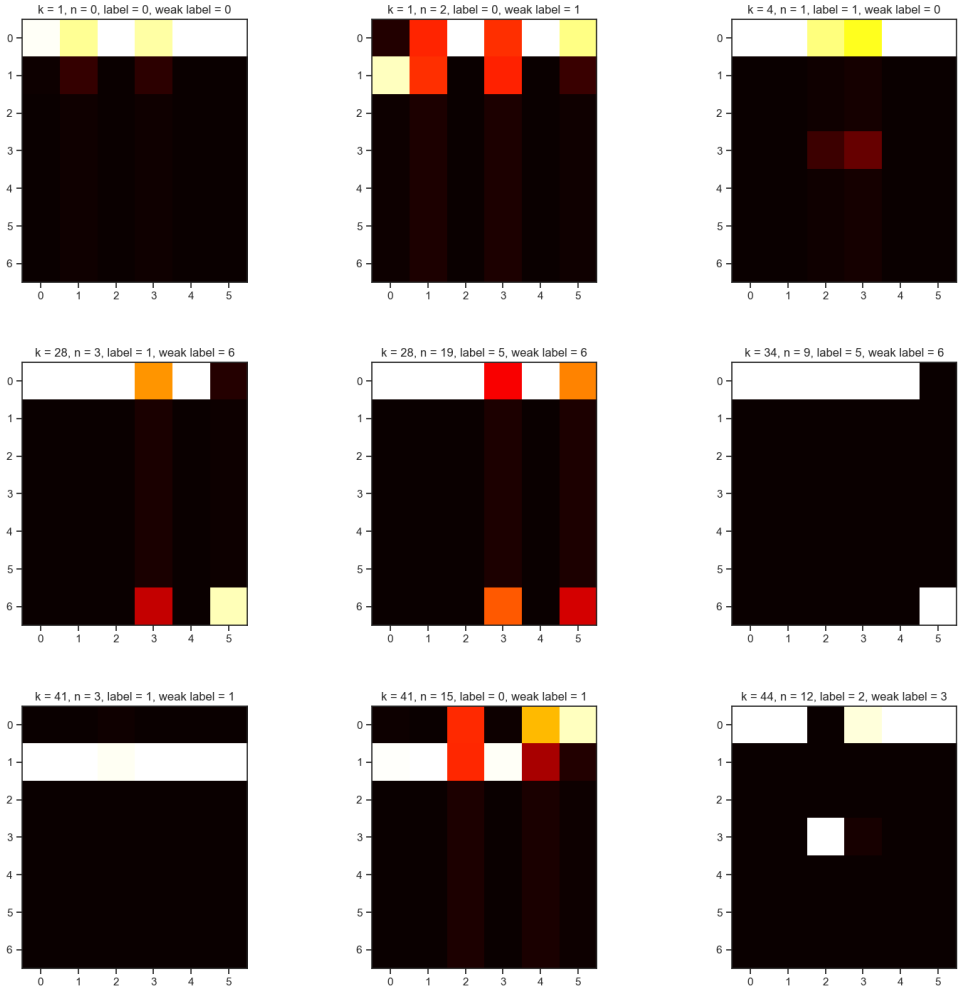


Figure A.2: Heat plots of the transitions $\boldsymbol{\phi}_k^n$ at convergence after training. For an array of different labeling functions k and different datapoints n, sampled from the TREC dataset (number of class labels $C = 6$ and number of labeling functions $K = 68$). This illustrates that our model is learning different transitions for different datapoints.

labels based only on the true label y, that is

$$p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y}, \mathbf{x}) = \prod_{k=1}^{K} p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}_k|\mathrm{y}). \tag{A.1}$$

A simple alternative choice to consider dependent weak labels is to include an additional latent $\mathbf{h}$

$$p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}|\mathrm{y}, \mathbf{x}) = \sum_{\mathbf{h}} p_{\boldsymbol{\phi}}(\mathbf{h}|\mathbf{x}) \prod_{k=1}^{K} p_{\boldsymbol{\phi}}(\tilde{\mathrm{y}}_k|\mathrm{y}, \mathbf{h}). \tag{A.2}$$

For a discrete $\mathbf{h}$ we can use the EM algorithm for learning. In principle, we can also include a dependency $p_{\boldsymbol{\phi}}(\mathbf{h}|\mathrm{y})$. We leave further investigation of this variant to future work.

Appendix B

Generalization Gap In Amortized Inference

B.1 Application to lossless compression

Lossless compression is an important application of VAEs where improved generalization translates directly to compression rate. Given a VAE with $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$, a compressor can be efficiently implemented using the Bits Back algorithm [193, 104] with the ANS coder [194].

In algorithm 3, we summarize the Bits Back procedure with amortized inference to compress/decompress a test data point $\boldsymbol{x}'$ to a stack that contains bit string of messages. The resulting code length for data $\boldsymbol{x}'$ is approximately equal to the negative ELBO:

$$-\log_2 p_{\boldsymbol{\theta}}(\boldsymbol{x}'|\boldsymbol{z}') - \log_2 p(\boldsymbol{z}') + \log_2 q_{\boldsymbol{\phi}}(\boldsymbol{z}'|\boldsymbol{x}'). \tag{B.1}$$

Algorithm 3 Bits Back with Amortized Inference

1: Share $\{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}), q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}), p(\mathbf{z})\}$ with sender and receiver.
2: **Compression:**
3: Draw sample $\boldsymbol{z}' \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ from the stack.
4: Encode $\boldsymbol{x}' \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}')$ onto the stack.
5: Encode $\boldsymbol{z}' \sim p(\mathbf{z})$ onto the stack.
6: **Decompression:**
7: Decode $\boldsymbol{z}' \sim p(\mathbf{z})$ from the stack.
8: Decode $\boldsymbol{x}' \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}')$ from the stack.
9: Encode $\boldsymbol{z}' \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ onto the stack.

We demonstrated in chapter 4 that $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ may overfit to the training data, directly degrading compression performance in this setting by producing a worse ELBO on test data where it matters. To improve the compression BPD, the optimal inference strategy outlined in section 4.3 can be applied within the Bits Back algorithm. In the compression stage, we can train $\boldsymbol{\phi}$ using the single datapoint:

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \text{ELBO}(\boldsymbol{x}', \boldsymbol{\theta}, \boldsymbol{\phi}). \tag{B.2}$$

When the $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ is parameterized to be a Gaussian, we can take $\boldsymbol{\phi}$ to be the mean and standard deviation $\mathcal{N}(\boldsymbol{\phi}_{\boldsymbol{\mu}}, \boldsymbol{\phi}^2_{\boldsymbol{\sigma}})$.

In the decompression stage the compressed data $\boldsymbol{x}'$ is recovered before the $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ is used to encode $\boldsymbol{z}'$. Therefore, we can also train $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}')$ using the recovered $\boldsymbol{x}'$ to maximize the test ELBO.

If the optimization procedure is the same as that used at the compression stage, we will get the same $q_{\boldsymbol{\phi}^*}(\mathbf{z}|\boldsymbol{x}')$. In practice, we need to prespecify the number of gradient descent steps K to train $\boldsymbol{\phi}$. When K is large, we approximately recover the optimal inference strategy and the code length is approximately

$$-\log_2 p_{\boldsymbol{\theta}}(\boldsymbol{x}'|\boldsymbol{z}') - \log_2 p(\boldsymbol{z}') + \log_2 q_{\boldsymbol{\phi}^*}(\boldsymbol{z}'|\boldsymbol{x}'). \tag{B.3}$$

Algorithm 4 Bits Back with K-step Optimal Inference

1: Share $\{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}), q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}), p(\mathbf{z})\}$ and the optimization procedure B.2 between the sender and receiver.
2: **Compression:**
3: Take K gradient steps $\boldsymbol{\phi} \rightarrow \boldsymbol{\phi}^K$ using equation B.2.
4: Draw sample $\boldsymbol{z}' \sim q_{\boldsymbol{\phi}^K}(\mathbf{z}|\boldsymbol{x}')$ from the stack.
5: Encode $\boldsymbol{x}' \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}')$ onto the stack.
6: Encode $\boldsymbol{z}' \sim p(\mathbf{z})$ onto the stack.
7: **Decompression:**
8: Decode $\boldsymbol{z}' \sim p(\mathbf{z})$ from the stack.
9: Decode $\boldsymbol{x}' \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}')$ from the stack.
10: Take K gradient steps $\boldsymbol{\phi} \rightarrow \boldsymbol{\phi}^K$ using equation B.2.
11: Encode $\boldsymbol{z}' \sim q_{\boldsymbol{\phi}^K}(\mathbf{z}|\boldsymbol{x}')$ onto the stack.

This procedure of using optimal inference was first proposed in [195] in the context of lossy compression and then applied to lossless compression with Bits Back coding in [130]. By varying the optimization steps K in the optimal inference we can trade off between the speed of decompression and the compression rate. We summarise the Bits Back algorithm with K-step optimal inference in algorithm 4.

The added run-time of additional training steps when doing the actual compression is a major downside of this optimal inference approach. In contrast, our proposed reverse half-sleep inference scheme can improve the compression rate without adding any additional training burden at compression time.

Furthermore, our method can be combined with this optimal inference strategy to provide a better model initialization to reduce the amount of subsequent compression time training needed to reach the same compression rate.

We implement Bits Back with ANS [194] and compare the compression among four inference methods:

1. Baseline: This is the classic VAE-based compression introduced by [104]. For binary and grey MNIST, both the encoder and decoder contain 2 fully connected layers with 500 hidden units and latent dimension 10. The observation distributions are Bernoulli and discretized logistic distribution respectively. For CIFAR10, we use fully convolutional ResNets [8] with 3

residual blocks in the encoder/decoder, latent dimension 128 and discreteized logistic distribution with a channel-wise autogressive linear layer[196] as the observation distribution. We train both the amortized posterior and the decoder by maximizing the ELBO (equation 4.2) using ADAM with $lr = 3\times10^{-4}$ for 100, 100 and 500 epochs (for binary MNIST, grey MNIST and CIFAR10 respectively), and then apply algorithm 3 at compression time.

2. Reversed half-sleep: We use our method from equation 4.25 to further train our inference network before compression for an additional 100 and 300 epochs using ADAM with ($lr = 3\times10^{-4}$) for binary and grey MNIST respectively, and $lr = 1\times10^{-5}$ for 100 epochs for CIFAR10. Other training details are the same as the baseline method.

3. Optimal inference: we take the amortized posterior and decoder model from the baseline and apply the K-step optimal inference strategy described in algorithm 4 to do compression. We use ADAM and vary the K from 1 to 10 to achieve a trade-off curve between compression rate and speed. We actively choose the highest learning rate that makes the BPD consistently improve as we inrecase K: $lr = 5\times10^{-3}$ for binary and grey MNIST and $lr = 1\times10^{-3}$ for CIFAR10.

4. Reversed half-asleep to improve initialisation for optimal inference: we take the inference networks trained in option 2 above and the decoder model from the baseline and conduct K-step optimal inference. All other training details are as per method 3.

	Baseline	Ours	K=7
BPD	0.185	0.179	0.179
Com. Time	0.006	0.006	0.013
Dec. Time	0.006	0.006	0.013
Time Cost	-	0%	116.7%

Table B.1: Demonstrates the compression and decompression time comparisons for MNIST

In tables B.1, B.2, we detail test BPD comparisons for the different methods outlined. We can see if optimization is not allowed at compression time, the

	Baseline	Ours	K=8
BPD	4.602	4.585	4.585
Com. Time	0.27	0.27	0.38
Dec. Time	0.26	0.26	0.38
Time Cost	-	0%	46.2%

Table B.2: Demonstrates the compression and decompression time comparisons for CIFAR10

use of our reverse-half-sleep method achieves better compression rate with no additional computational cost. If we allow K-step optimization during compression, for a given computational budget, the amortized posterior initialized using our reverse-half-sleep method also achieves lower BPD, which leads to a better trade-off between the run-time and compression rate.

Appendix C

Solipsistic Reinforcement Learning

C.1 Variational optimization

With VO we model the parameters of the policy with a Gaussian distribution $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\sigma})$ to form a differentiable upper bound that we can minimise w.r.t $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

$$U(\boldsymbol{\mu}, \boldsymbol{\sigma}) = \mathbb{E}_{p(\boldsymbol{w}|\boldsymbol{\mu},\boldsymbol{\sigma})} - \mathcal{E}(\boldsymbol{w}), \tag{C.1}$$

where $\mathcal{E}(\boldsymbol{w})$ is our memory model planning objective from equation 5.11. After optimising this bound with respect to $\boldsymbol{\mu}, \boldsymbol{\sigma}$, we take the final $\boldsymbol{\mu}$ as our value for $\boldsymbol{w}$. We compute gradients with the usual log-derivative trick, using J Monte Carlo samples to approximate the resulting expectation as follows. In practice, we find that learning $\boldsymbol{\sigma}$ does not improve performance, hence we fix it to an isotropic value of 0.2 throughout and minimize w.r.t $\boldsymbol{\mu}$,

$$\frac{\partial U}{\partial \boldsymbol{\mu}} \approx \frac{1}{J} \sum_{j=1}^{J} \frac{\partial}{\partial \boldsymbol{\mu}} \log p(\boldsymbol{w}^j|\boldsymbol{\mu}, \boldsymbol{\sigma})(-\mathcal{E}(\boldsymbol{w}^j)), \tag{C.2}$$

where $\boldsymbol{w}^j$ refers to the j^{th} Monte Carlo sample of the policy parameters - see section policy training in algorithm 1 for further details. In practice, we also apply fitness shaping [154] to $\mathcal{E}(\boldsymbol{w}^j)$ in equation C.2 to make the VO update invariant w.r.t. order-preserving transformations of the reward value.

C.2 Experiment details

C.2.1 MNIST Game

In this section we discuss the details of the architectures and hyper-parameter settings used for our toy MNIST game experiment from section 5.4.1.

Model architecture The recognition network is composed of a two-layer convolution neural network, followed by a two-layer feed-forward network. The two convolution layers have 10 and 20 filters respectively with stride 2 and kernel size 5. We use max pooling after convolutional layer. We use ReLU activation functions for both the convolutional and feed-forward layers. The forward network has 100 hidden units in each layer and the output size is equal to the number of solipsistic states (in this case 10). We use a softmax function to create the probabilities of the categorical distribution for determining the solipsistic state assignments.

Model training For our model objective equation 5.4 we set the hyper parameters as $\lambda_s = 1$, $\lambda_r = 2$ and $m = 5$. We use ADAM [66] as the optimizer with learning rate 10^{-4} to train the model for $N_{model} = 3000$ iterations with batch size $B = 64$.

Redundancy filtering For computational efficiency, we approximate equation 5.3 using the following Monte Carlo approximation

$$\mathbb{E}_{i \neq t+1} \mathrm{KL}(p(\mathbf{s}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)||p(\mathbf{s}_i|\mathbf{x}_i)) \approx \frac{1}{B-1} \sum_{\boldsymbol{s}_j} \mathrm{KL}(p(\mathbf{s}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)||p(\boldsymbol{s}_j|\boldsymbol{x}_j)), \tag{C.3}$$

where $\boldsymbol{s}_j \in \mathcal{M}(\boldsymbol{s}_{t+1}) \backslash \boldsymbol{s}_{t+1}$ and $\mathcal{M}(\boldsymbol{s}_{t+1})$ is the mini-batch set that $\boldsymbol{s}_{t+1}$ belongs to and the set $\mathcal{M}(\boldsymbol{s}_{t+1}) \backslash \boldsymbol{s}_{t+1}$ has size $B-1$.

C.2.2 Gym Control from pixels

In this section we discuss the details of the architectures and hyper-parameter settings used for our Cartpole experiment from section 5.4.2. For model and policy training we follow algorithm 1.

Model architecture The recognition network is a four-layer convolution

neural network with Batch Norm [197] and ReLU activation functions followed by a feed-forward layer with output size equal to the size of the solipsistic representation $dim(S) = 16$. We set the kernel size to 3 for the first three convolutional layers and 5 for the last convolutional layer. We set the channel size to 8 and stride to 1 for the first convolutional layer and 16 with stride 2 for the other three convolutional layers. The policy network is a two layers feed forward neural network with 50 hidden units in each layer, which maps from the solipsistic state to a sigmoid function that parameterizes the probability of a Bernoulli distribution. The RNN we used is a single layer Gated Recurrent Unit (GRU) [65] with memory in the first time-step initialized as the first solipsistic state s_1. In each recurrent step, the GRU cell takes one action as input and outputs the prediction of the solipsistic state for the next time step. Therefore, the size of the hidden memory of GRU is equal to the size of the solipsistic state.

Model training We use ADAM with learning rate 10^{-4}, on batches of size $B = 10$ of sampled environment trajectories of length $T_{model} = 50$, with $\lambda_s = 1$, $\lambda_r = 2$ and $m = 5$.

Policy training We use VO (section C.1), with $J = 50$ parameter perturbations, under model prediction roll-outs of length $T_{policy} = 200$ and we train for $N_{policy} = 50$ iterations, using ADAM with learning rate 10^{-2}. We found that re-initializing the policy parameters w after each model update helped overall performance.

Exploration strategy At each iteration i of algorithm 1 during the trajectory collection phase we collect $C = 5$ trajectories and add them to our memory for subsequent model training. We use an exploration strategy (see line 9 of 1) during trajectory collection as follows: for $c = 1$ we follow the latest policy $p_{\boldsymbol{w}_i}(\mathbf{a}_t|\mathbf{s}_t)$ with re-planning (section 5.3) to collect a full trajectory. Then, for $c = 2, \ldots, 5$, we instead follow an ϵ-policy, where we take the action sampled from $p_{\boldsymbol{w}_i}(\mathbf{a}_t|\mathbf{s}_t)$ with probability ϵ and we take a random action with probability $1 - \epsilon$, where $\epsilon = 0.5$.

Redundancy filtering We use $\mathbf{x}_t^m$ to denote the state at time t of the mth trajectory. We approximate equation 5.3 using the following Monte Carlo approximation

$$\mathbb{E}_{i \neq t+1} \mathrm{KL}\Big(p(\mathbf{s}_{t+1}^m | \mathbf{x}_t^m, \mathbf{a}_t^m) || p(\mathbf{s}_i | \mathbf{x}_i)\Big) \approx \mathrm{KL}\Big(p(\mathbf{s}_{t+1}^m | \mathbf{x}_t^m, \mathbf{a}_t^m) || p(\mathbf{s}_{t+1}^n | \mathbf{x}_{t+1}^n)\Big), \tag{C.4}$$

where the nth ($n \neq m$) trajectory is sampled from memory. We found this one-sample Monte Carlo approximation works well in practice.

C.2.3 Baseline model-free methods

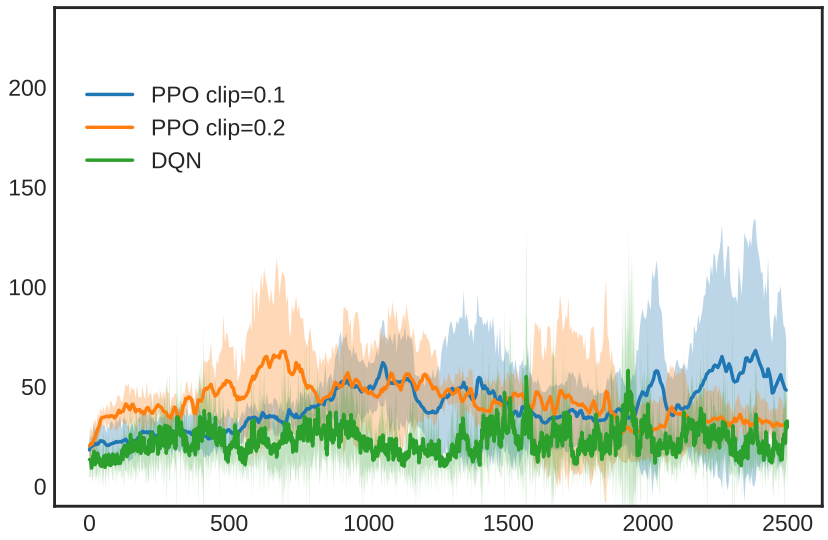


Figure C.1: Evaluating the baseline policies trained using 2500 trajectories. We plot the average over 5 runs using different random seeds for our parameter initialization. We smooth the curves using a moving average with window size 3. We find the curves for PPO have high variance, meaning that PPO can occasionally get reasonable performance for this number of trajectories (e.g. balancing the pole for over 100 time steps successfully) but is not very stable during training and across across different random seeds.

C.2.3.1 PPO

We implemented the PPO method [145] as a baseline to evaluate our methods. We run PPO using two different clip ratios 0.1 and 0.2. The policy and the value networks share similar architecture to our recognition convolutional

neural network described in section C.2.2 with the only difference being the dimensionality of the output layers. For the Cartpole experiment, the output size of the policy network is 2 and we use a softmax activation function to parameterize the probabilities of choosing from two actions. The output of the value network is a linear layer with output dimension 1. For each training episode, we sample 5 trajectories from the true environment with a maximum length of 200. We then train the model for 10 epochs using ADAM where we take the hyper-parameters provided by the OpenAI Baselines implementation [198], where learning is 3^{-4}, $\gamma = 0.99$, $\lambda = 0.95$, the weight of the value term is 0.5 and the weight of the entropy term is 0.01. We report results for two different clip ratios of 0.1 and 0.2.

C.2.3.2 DQN

The other model free baseline we compared to is DQN [144]. Like for the case of PPO, we keep the architecture of the Q-network and the target network similar to our recognition network, except for the final output layer. In our Cartpole experiment, the Q-network outputs the Q-value for the two possible actions given the states. The action with the larger Q-value is chosen during control. We train DQN for 2500 episodes (as shown in figure C.1). For each episode, we sample one trajectory with the maximum length of 200. After a limited grid search of hyper-parmeters, we find that the hyper-parameters from the PyTorch DQN tutorial [199, 192].

C.2.4 Activation maps for PPO

In this section, we visualize the activations of the first convolution layer of the policy network and the value network for PPO. We plot all 8 filters' activations and use a sigmoid function to create the following grey-scale images. Each column represents a filter. Each row represents the state of a trajectory as the cart is pushed to the right with monotonically increasing velocity (as in figure 5.6). In both figure C.2 and figure C.3, none of the activation maps have a change in activation that can clearly be interpreted as relating to the increase

in velocity (unlike the case of the solipsistic recognition model illustrated in figure 5.6); only the positional information of the cart and pole is obviously captured.

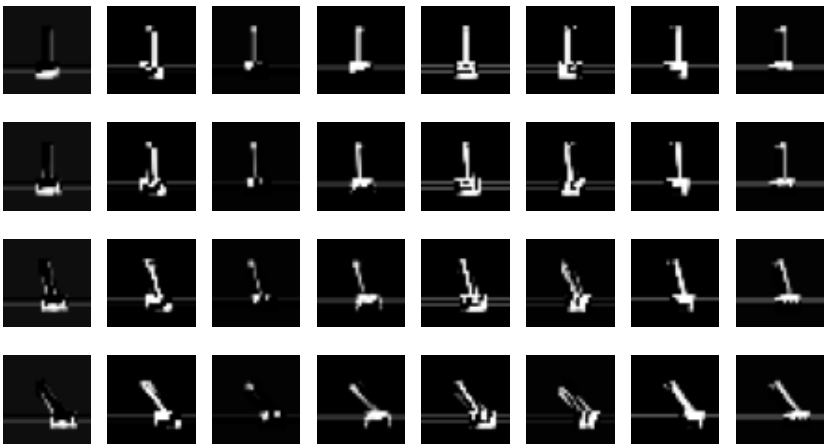


Figure C.2: Visualization of the activations in PPO's policy network.

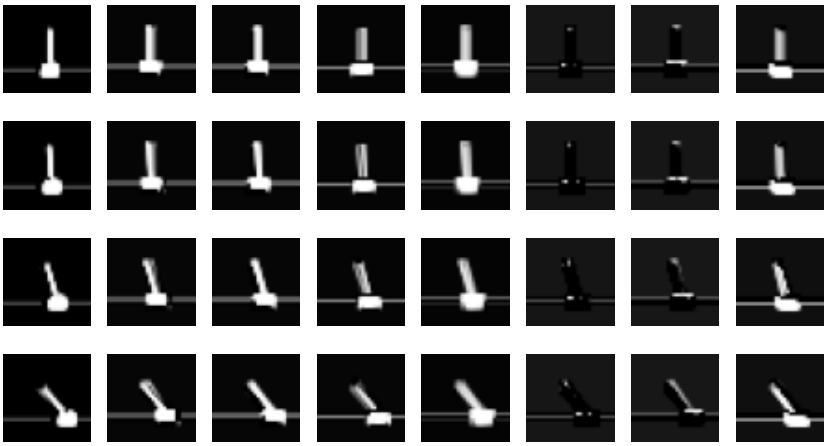


Figure C.3: Visualization of the activations in PPOs value network.

Appendix D

Active Preference Learning

D.1 Data preprocessing

For IMDB, each sample $\boldsymbol{x}$ is randomly drawn beginning of a review. The only processing we do here is to randomly truncate $\boldsymbol{x}$ to a number of tokens randomly drawn from the range 8-16 tokens. See table D.1 for some truncated examples that we feed to the model to complete a positive review for:

Truncated movie review prompt samples
I very much looked forward to this movie. Its a good family ...
Really, I can't believe that I spent $5 on this movie. I am a huge zombie ...
I have read all of the Love Come Softly books....
I've seen all four of the movies in this series. Each one strays further ...

Table D.1: IMDB data from `https://huggingface.co/datasets/imdb`; randomly truncated to produce a prompt for training data generation and evaluation.

For TLDR, we filtered the Reddit posts between 200-1000 characters. This was mainly due to memory contraints of the GPUs used to train the models. We also filtered whole broad categories of Reddit posts out, such as r/offmychest and r/tifu, because they had high likelihood of containing explicit content. Finally we removed trailing space tokens. See table D.2 for examples.

D.2 Fine-tuning iterations

In order to determine how many fine-tuning epochs to carry out after each new data acquisition step, we took a simple approach of defining a fixed number

Prompt	SUBREDDIT: r/cats TITLE: Acquired cat! Now a question.. POST: So, I just got a lovely little cat named Luna. She's about a year, a year and a half and pretty tiny. I live in an apartment located on the 5th floor of my building. My apartment doesn't have AC (I'm in NYC) and I usually like to leave the windows open for ventilation. They've got child bars, but because Luna is so small she can easily fit through them–and did a few moments ago. Nearly gave me a heart attack watching her slip through them and walked out onto a very narrow ledge 5 floors above a concrete sidewalk. She came right back in, but now I'm concerned about having a dead cat on my hands (or more accurately, on my sidewalk). So my question is, should I trust her cat instincts and leave the windows open? Or shall I sit in a stuffy apartment with the windows sealed? TL;DR:
Human Summary	I live on the 5th floor and my cat just walked out on my window ledge and came back in. Should I be nervous she's going to explore too far out and fall to her kitty death?
Prompt	SUBREDDIT: r/AskReddit TITLE: I want to throw a great party, Reddit. What are some special things that I can do to make this happen? POST: Alright, so my 19th birthday is next Friday. I live in a college town, renting a house alone for the summer, and I want to throw a great party. I have invited friends from both school (which means they are coming from all over the state) and from home. However, since it's summer, I know that a lot of people won't show. Gas is expensive and for some, it's a far drive, so I understand. I'm thinking music, card playing, beer pong, etc... But what else can I do to make sure that everyone mingles, it isn't lame, and that things stay under control while everyone still has a good time? TL;DR:
Human Summary	How do I throw a kick ass party with a bunch of people who don't really know each other and my house is still standing in the morning?

Table D.2: Samples of TLDR data from `https://huggingface.co/datasets/CarperAI/openai_summarize_tldr`

of epochs. The parameters of the model are re-initialized after each data acquisition step. We based the choice of how many epochs to fine-tune for on the number of epochs it took on average for the model to converge at different dataset sizes. We analysed loss and win-rate curves for the different model and dataset combinations and decided upon 50 epochs for IMDB and 70 for TLDR - see figure D.1 for a sample of convergence behaviour.

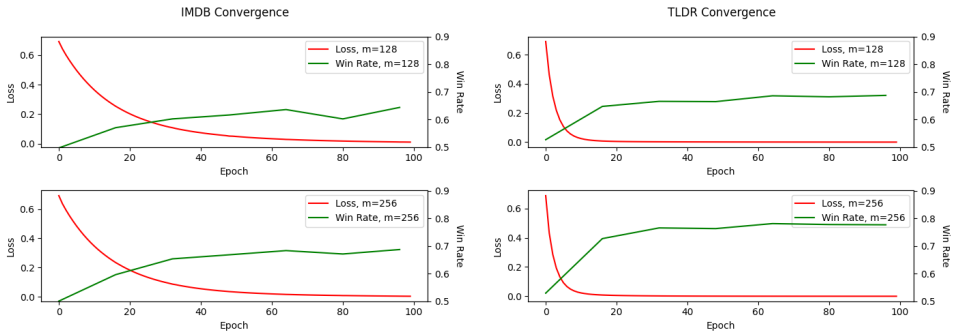


Figure D.1: Illustrates a sample of how the convergence of the loss relates to the win-rate. Used for emprically inferring the number of fine-tuning epochs to apply after each data acquisition step.

Future work will explore different early stopping criteria and techniques for

not resetting the model parameters after each data acquisition step to make the method more computationally efficient and practical for scaling up.

D.3 Example responses

In tables D.3, D.4 we provide examples of prompt completion pairs for our models discussed in section 6.5.2 after the fine-tuning process alongside the preferred GPT-4 oracle choice.

Prompt	The comments of the previous user are harsh indeed
A: Fine-tuned completion	. This is a great film with a great story and a great cast. Highly recommended. I saw this movie on Sund
B: Original completion	. I am a fan of the original series and I have watched the first two seasons of the series. I have also
Preference	A
Rationale	Review A is better as it expresses a clear positive sentiment towards the film, while Review B does not provide any opinion on the movie
Prompt	I was looking forward to this so much, being a
Fine-tuned completion	huge Harry Potter fan. I was very impressed with the story and the acting. I highly recommend it.
Original completion	huge fan of the original series. I was so disappointed. I was expecting a lot more from this.
Preference	A
Rationale	Review A is better because it provides a positive sentiment about the movie, is grammatically correct, and does not contain repetition like Review B.

Table D.3: Sampled completions, and GPT-4 oracle evaluation from an IMDB fine-tuned with certainty acquisition.

D.4 Online variation

In section 6.5 we note that a practical limitation to scaling algorithm 2 to significantly larger models is that the parameters $\boldsymbol{\theta}_t$ are reset to $\boldsymbol{\theta}_0$ in step 12 and the model is fine-tuned on all the data acquired so far each step. Here we motivate an online variant where we instead don't reset the parameters and do a single gradient update with respect to the most recently acquired data during fine-tuning. This significantly reduces the time spent fine-tuning during the active learning approach. This is similar to the approach taken in [200], where they randomly, instead of actively, acquire the data and consider comparing online vs offline.

We re-run our IMDB experiment from section 6.5 with this online active

Prompt	SUBREDDIT: r/relationships TITLE: I(17M)am gealous because my girlfriend(16F) of over 4 months is having conversation that she starts with my classmate. POST: I am now really down because of this. We had an argument 2 months ago about this and I told her to stay away from my classmate(he is a really bad person and always speaks really bad about her when he is with others or even with me, we've been colleagues for 10 years now and he is awful, I hate him), and I'm sick of it. The problem is, today after checking her phone I realized she starts most conversation and he is mostly not answering to her immediately. I am not sure what to do and I am really confused and really really sad and disappointed. What should I do? TL;DR:
A: Fine-tuned completion	GF of over 4 months starts most conversation with my classmate, he is mostly not answering to her immediately. I am not sure what to do and am really sad and disappointed. Need advice. [endoftext]
B: Original completion	I am really sad because my girlfriend starts most conversation with my classmate and I'm sick of it. What should I do?[endoftext]
Preference	A
Rationale	Summary A is preferred as it includes more relevant details from the post, such as the duration of the relationship, the classmate's lack of immediate response, and the poster's need for advice, which were missing in Summary B.
Prompt	SUBREDDIT: r/dogs TITLE: Need a little advice regarding our elderly dog. POST: Our rescue [Lt. Ellen Ripley] has been showing her age lately. Actually for the last six months or so she has been going a bit downhill. Formerly very active and happy to play she's blind, deaf and now sleeps or spins in circles all day. She hasn't been able to hold her water or poops at all and we've been coming home to a nightly cleanup. Worse she's been biting at us a lot and seems to think our hands are actually delicious fish sticks. We've tried medicine and have seen the vet numerous times but she just seems to be getting worse and worse. So, advice time. Am I being selfish? I feel like keeping her around may not be the best thing for her anymore. I worry that she spends her time now scared, confused and in the dark. But so far as I know, she's not in any pain and she can still recognize me on occasion. I'll TL;DR:
A: Fine-tuned completion	Old rescue dog has been getting worse and worse. Tried medicine and the vet. Don't know if keeping her around is the best thing for her anymore. Need advice.[endoftext]
B: Original completion	We have a dog that is getting older and is starting to become more depressed. What should we do about it?[endoftext]
Preference	A
Rationale	Summary A is more accurate and detailed, capturing the severity of the dog's condition, the attempts at treatment, and the owner's dilemma, while Summary B oversimplifies the situation by referring to the dog as merely "more depressed."

Table D.4: Sampled completions, and GPT-4 oracle evaluation from an TLDR fine-tuned with certainty acquisition.

variation for random and preference certainty acquisitions. We do 3 random seeds and include standard error bars in figure D.2. We find that preference certainty significantly outperforms random on this problem, motivating further study.

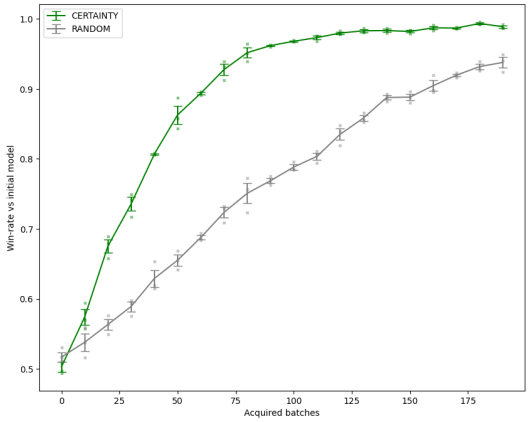


Figure D.2: Win-rate vs initial model after each acquired batch for IMDB with random and preference certainty acquisition and online fine-tuning. Only a single fine-tuning gradient step is taken on the latest batch.

Bibliography

[1] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Re. Snorkel: Rapid Training Data Creation with Weak Supervision. In *The VLDB Journal*. 2020.

[2] M. Zhang, P. Hayes, and D. Barber. Generalization Gap in Amortized Inference. In *Advances in Neural Information Processing Systems*. 2022.

[3] W. Muldrew, P. Hayes, M. Zhang, and D. Barber. Active Preference Learning for Large Language Models. In *International Conference on Machine Learning*. 2024.

[4] D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*. 2020.

[5] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. In *Nature*. 2015.

[6] J. Schmidhuber. Deep Learning In Neural Networks: An Overview. In *Neural Networks*. 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 2012.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[9] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE Conference on Acoustics, Speech and Signal Processing*. 2013.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*. 2017.

[11] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*. 2014.

[12] L. Chen, Y. Li, C. Huang, B. Li, Y. Xing, D. Tian, L. Li, Z. Hu, X. Na, Z. Li, S. Teng, C. Lv, J. Wang, D. Cao, N. Zheng, and F. Y. Wang. Milestones in Autonomous Driving and Intelligent Vehicles: Survey of Surveys. In *IEEE Transactions on Intelligent Vehicles*. 2023.

[13] T. Dohmke, M. Iansiti, and G. Richards. Sea Change in Software Development: Economic and Productivity Analysis of the AI-Powered Developer Lifecycle. In *arXiv preprint arXiv:2306.15033*. 2023.

[14] C. Biever. ChatGPT broke the Turing test-the race is on for new ways to assess AI. In *Nature*. 2023.

[15] U. K. Government. The Bletchley Declaration. In *https://www.gov.uk/government/publications/ai-safety-summit-2023-the-bletchley-declaration/the-bletchley-declaration-by-countries-attending-the-ai-safety-summit-1-2-november-2023*. 2023.

[16] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. In *Psychological review*. 1958.

[17] M. Minsky and S. Papert. Perceptrons: An Introduction to Computational Geometry. In *MIT Press*. 1969.

[18] L. Devroye, L. Györfi, and G. Lugosi. A probabilistic theory of pattern recognition. In *Springer*. 2013.

[19] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen. Classification and Regression Trees. In *Chapman and Hall/CRC*. 1984.

[20] S. Amari. A theory of adaptive pattern classifiers. In *IEEE Transactions on Electronic Computers*. 1967.

[21] S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. In *Master's Thesis (in Finnish), Univ. Helsinki*. 1970.

[22] M. Hu, Y. Peng, Z. Huang, X. Qiu, F. Wei, and M. Zhou. Reinforced mnemonic reader for machine reading comprehension. In *International Joint Conference on Artificial Intelligence*. 2018.

[23] Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu. Attention-over-Attention Neural Networks for Reading Comprehension. In *Association for Computational Linguistics*. 2017.

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. In *Nature*. 2016.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *arXiv preprint arXiv:1312.5602*. 2013.

[26] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*. 2020.

[27] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. In *arXiv preprint arXiv:2001.08361*. 2020.

[28] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP. In *arXiv preprint arXiv:1906.02243*. 2019.

[29] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. In *Science*. 2015.

[30] L. Shen, L. R. Margolies, J. H. Rothstein, E. Fluder, R. McBride, and W. Sieh. Deep learning to improve breast cancer detection on screening mammography. In *Scientific Reports*. 2019.

[31] S. Sajid, S. Hussain, and A. Sarwar. Brain tumor detection and segmentation in MR images using deep learning. In *Arabian Journal for Science and Engineering*. 2019.

[32] M. Dildar, S. Akram, M. Irfan, H. U. Khan, M. Ramzan, A. R. Mahmood, S. A. Alsaiari, A. H. M. Saeed, M. O. Alraddadi, and M. H. Mahnashi. Skin cancer detection: a review using deep learning techniques. In *International Journal of Environmental Research and Public Health*. 2021.

[33] D. C. Elton, Z. Boukouvalas, M. D. Fuge, and P. W. Chung. Deep learning for molecular design — a review of the state of the art. In *Molecular Systems Design & Engineering*. 2019.

[34] J. Bradshaw, B. Paige, M. J. Kusner, M. Segler, and J. M. Hernández-Lobato. A model to search for synthesizable molecules. In *Advances in Neural Information Processing Systems*. 2019.

[35] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. In *MIT Press*. 2018.

[36] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. A survey of deep learning techniques for autonomous driving. In *Journal of Field Robotics*. 2020.

[37] B. Singh, R. Kumar, and V. P. Singh. Reinforcement learning in robotic applications: a comprehensive survey. In *Artificial Intelligence Review*. 2022.

[38] Z.-H. Zhou. A brief introduction to weakly supervised learning. In *National Science Review*. 2018.

[39] J. Zhang, Y. Yu, Y. Li, Y. Wang, Y. Yang, M. Yang, and A. Ratner. WRENCH: A Comprehensive Benchmark for Weak Supervision. In *CoRR*. 2021.

[40] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. In *Artificial Neural Networks and Machine Learning - ICANN*. 2018.

[41] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang. A survey of deep active learning. In *ACM Computing Surveys*. 2021.

[42] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. In *Journal of Big Data*. 2019.

[43] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2021.

[44] Y. Ouali, C. Hudelot, and M. Tami. An overview of deep semi-supervised learning. In *arXiv preprint arXiv:2006.05278*. 2020.

[45] P. Hayes, M. Zhang, R. Habib, J. Burgess, E. Yilmaz, and D. Barber. Integrated Weak Learning. In *arXiv preprint arXiv:2206.09496*. 2022.

[46] P. Hayes, M. Zhang, Z. Andi, and D. Barber. Solipsistic Reinforcement Learning. In *International Conference on Learning Representations Workshop on Self-Supervision for Reinforcement Learning*. 2021.

[47] M. Morris, P. Hayes, I. J. Cox, and V. Lampos. Neural network models for influenza forecasting with associated uncertainty using Web search activity trends. In *PLoS Computational Biology*. 2023.

[48] M. Zhang, P. Hayes, T. Bird, R. Habib, and D. Barber. Spread Divergence. In *International Conference on Machine Learning*. 2020.

[49] E. Yilmaz, P. Hayes, R. Habib, J. Burgess, and D. Barber. Sample efficient model evaluation. In *arXiv preprint arXiv:2109.12043*. 2021.

[50] M. Zhang, O. Key, P. Hayes, D. Barber, B. Paige, and F.-X. Briol. Towards Healing the Blindness of Score Matching. In *Advances in Neural Information Processing Systems Workshop on Score-Based Methods*. 2022.

[51] D. J. MacKay. Information Theory, Inference and Learning Algorithms. In *Cambridge University Press*. 2003.

[52] C. M. Bishop. Pattern Recognition and Machine Learning. In *Springer*. 2006.

[53] D. Barber. Bayesian Reasoning and Machine Learning. In *Cambridge University Press*. 2012.

[54] G. A. Barnard. Statistical Inference. In *Journal of the Royal Statistical Society*. 1949.

[55] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. In *Journal of Machine Learning research*. 2003.

[56] T. K. Moon. The expectation-maximization algorithm. In *IEEE Signal processing magazine*. 1996.

[57] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt. Advances in variational inference. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2018.

[58] M. J. Wainwright, M. I. Jordan, et al. Graphical models, exponential families, and variational inference. In *Foundations and Trends in Machine Learning*. 2008.

[59] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*. 2013.

[60] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and variational inference in deep latent gaussian models. In *International Conference on Machine Learning*. 2014.

[61] K. Hornik. Approximation capabilities of multilayer feedforward networks. In *Neural Networks*. 1991.

[62] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. In *Biological Cybernetics*. 1980.

[63] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*. 2018.

[64] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. In *Neural Computation*. 1997.

[65] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In *arXiv preprint arXiv:1412.3555*. 2014.

[66] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*. 2015.

[67] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 2019.

[68] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs. In *http://github.com/google/jax*. 2018.

[69] P. Werbos. Backpropagation through time: what it does and how to do it. In *Proceedings of the IEEE*. 1990.

[70] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. In *Nature*. 1986.

[71] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. In *Journal of Marchine Learning Research*. 2018.

[72] A. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data Programming: Creating Large Training Sets, Quickly. In *arXiv:stat.ML:1605.07723*. 2017.

[73] A. Ratner, B. Hancock, J. Dunnmon, F. Sala, S. Pandey, and C. Ré. Training complex models with multi-task weak supervision. In *AAAI Conference on Artificial Intelligence*. 2019.

[74] S. H. Bach, D. Rodriguez, Y. Liu, C. Luo, H. Shao, C. Xia, S. Sen, A. Ratner, B. Hancock, H. Alborzi, R. Kuchhal, C. Ré, and R. Malkin. Snorkel DryBell: A Case Study in Deploying Weak Supervision at Industrial Scale. In *CoRR*. 2018.

[75] D. Fu, M. Chen, F. Sala, S. Hooper, K. Fatahalian, and C. Ré. Fast and three-rious: Speeding up weak supervision with triplet methods. In *International Conference on Machine Learning*. 2020.

[76] J. Dunnmon, A. Ratner, N. Khandwala, K. Saab, M. Markert, H. Sagreiya, R. E. Goldman, C. Lee-Messer, M. P. Lungren, D. L. Rubin, and C. Ré. Cross-Modal Data Programming Enables Rapid Medical Machine Learning. In *CoRR*. 2019.

[77] G. B. Goh, C. Siegel, A. Vishnu, and N. Hodas. Using rule-based labels for weak supervised learning: a ChemNet for transferable chemical property prediction. In *ACM International Conference on Knowledge Discovery Data Mining*. 2018.

[78] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *CoRR*. 2017.

[79] S. R. Cachay, B. Boecking, and A. Dubrawski. End-to-End Weak Supervision. In *CoRR*. 2021.

[80] W. Ren, Y. Li, H. Su, D. Kartchner, C. Mitchell, and C. Zhang. Denoising Multi-Source Weak Supervision for Neural Text Classification. In *EMNLP*. 2020.

[81] F. Locatello, S. Bauer, M. Lucic, G. Raetsch, S. Gelly, B. Schölkopf, and O. Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *International Conference on Machine Learning*. 2019.

[82] G. Casella and R. L. Berger. Statistical Inference. In *Duxbury Advanced Series*. 2021.

[83] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Association for Computational Linguistics*. 1995.

[84] G. Karamanolakis, S. Mukherjee, G. Zheng, and A. H. Awadallah. Self-training with weak supervision. In *arXiv preprint arXiv:2104.05514*. 2021.

[85] C. Liang, Y. Yu, H. Jiang, S. Er, R. Wang, T. Zhao, and C. Zhang. Bond: Bert-assisted open-domain named entity recognition with distant supervision. In *ACM International Conference on Knowledge Discovery Data Mining*. 2020.

[86] R. Zhang, Y. Yu, and C. Zhang. Seqmix: Augmenting active sequence labeling via sequence mixup. In *arXiv preprint arXiv:2010.02322*. 2020.

[87] Y. Yang, W. Chen, Z. Li, Z. He, and M. Zhang. Distantly supervised NER with partial annotation learning and reinforcement learning. In *International Conference on Computational Linguistics*. 2018.

[88] Z. Y. Zhang, P. Zhao, Y. Jiang, and Z. H. Zhou. Learning from incomplete and inaccurate supervision. In *IEEE Transactions on Knowledge and Data Engineering*. 2021.

[89] M. Ren, W. Zeng, B. Yang, and R. Urtasun. Learning to Reweight Examples for Robust Deep Learning. In *International Conference on Machine Learning*. 2018.

[90] C. Northcutt, L. Jiang, and I. Chuang. Confident learning: Estimating uncertainty in dataset labels. In *Journal of Artificial Intelligence Research*. 2021.

[91] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel. Using trusted data to train deep networks on labels corrupted by severe noise. In *Advances in Neural Information Processing Systems*. 2018.

[92] H. Wang, B. Liu, C. Li, Y. Yang, and T. Li. Learning with noisy labels for sentence-level sentiment classification. In *arXiv preprint arXiv:1909.00124*. 2019.

[93] A. P. Dawid and A. M. Skene. Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm. In *Journal of the Royal Statistical Society*. 1979.

[94] Y. Zhang, X. Chen, D. Zhou, and M. I. Jordan. Spectral methods meet EM: A provably optimal algorithm for crowdsourcing. In *Advances in Neural Information Processing Systems*. 2014.

[95] N. Dalvi, A. Dasgupta, R. Kumar, and V. Rastogi. Aggregating Crowdsourced Binary Ratings. In *International Conference on World Wide Web*. 2013.

[96] R. Tanno, A. Saeedi, S. Sankaranarayanan, D. C. Alexander, and N. Silberman. Learning from noisy labels by regularized estimation of annotator confusion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.

[97] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *arXiv preprint arXiv:1910.01108*. 2019.

[98] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*. 2014.

[99] S. Shalev-Shwartz and S. Ben-David. Understanding machine learning: From theory to algorithms. In *Cambridge University Press*. 2014.

[100] M. Zhang, A. Zhang, and S. McDonagh. On the Out-of-distribution Generalization of Probabilistic Image Modelling. In *Neural Information Processing Systems*. 2021.

[101] L. Theis, A. V. D. Oord, and M. Bethge. A note on the evaluation of generative models. In *arXiv preprint arXiv:1511.01844*. 2015.

[102] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*. 2014.

[103] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*. 2017.

[104] J. Townsend, T. Bird, and D. Barber. Practical lossless compression with latent variables using bits back coding. In *International Conference on Learning Representations*. 2019.

[105] J. Townsend, T. Bird, J. Kunze, and D. Barber. Hilloc: Lossless image compression with hierarchical latent variable models. In *International Conference on Learning Representations*. 2020.

[106] F. Kingma, P. Abbeel, and J. Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *International Conference on Machine Learning*. 2019.

[107] M. Zhang, J. Townsend, N. Kang, and D. Barber. Parallel Neural Local Lossless Compression. In *arXiv preprint arXiv:2201.05213*. 2022.

[108] C. E. Shannon. A Mathematical Theory of Communication. In *The Bell System Technical Journal*. 1948.

[109] Y. Wang and D. M. Blei. Frequentist consistency of variational Bayes. In *Journal of the American Statistical Association*. 2019.

[110] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. In *Journal of the American statistical Association*. 2017.

[111] F. V. Agakov and D. Barber. An auxiliary variational method. In *International Conference on Neural Information Processing*. 2004.

[112] L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther. Auxiliary deep generative models. In *International Conference on Machine Learning*. 2016.

[113] E. Challis and D. Barber. Affine independent variational inference. In *Advances in Neural Information Processing Systems*. 2012.

[114] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*. 2015.

[115] C. Cremer, X. Li, and D. Duvenaud. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning*. 2018.

[116] R. Shu, H. H. Bui, S. Zhao, M. J. Kochenderfer, and S. Ermon. Amortized inference regularization. In *Neural Information Processing Systems*. 2018.

[117] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The Helmholtz Machine. In *Neural Computation*. 1995.

[118] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The wake-sleep algorithm for unsupervised neural networks. In *Science*. 1995.

[119] Y. LeCun. The MNIST database of handwritten digits. In *http://yann.lecun.com/exdb/mnist/*. 1998.

[120] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. In *University of Toronto*. 2009.

[121] A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *arXiv preprint arXiv:1711.00937*. 2017.

[122] S. Zhao, J. Song, and S. Ermon. Infovae: Information maximizing variational autoencoders. In *arXiv preprint arXiv:1706.02262*. 2017.

[123] B. Dai and D. Wipf. Diagnosing and enhancing VAE models. In *arXiv preprint arXiv:1903.05789*. 2019.

[124] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*. 2016.

[125] M. Zhang, T. Z. Xiao, B. Paige, and D. Barber. Improving VAE-based Representation Learning. In *arXiv preprint arXiv:2205.14539*. 2022.

[126] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2013.

[127] Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. In *arXiv preprint arXiv:1509.00519*. 2015.

[128] D. Barber, A. T. Cemgil, and S. Chiappa. Bayesian time series models. In *Cambridge University Press*. 2011.

[129] S. Zhao, H. Ren, A. Yuan, J. Song, N. Goodman, and S. Ermon. Bias and generalization in deep generative models: An empirical study. In *arXiv preprint arXiv:1811.03259*. 2018.

[130] Y. Ruan, K. Ullrich, D. Severo, J. Townsend, A. Khisti, A. Doucet, A. Makhzani, and C. J. Maddison. Improving Lossless Compression Rates via Monte Carlo Bits-Back Coding. In *arXiv preprint arXiv:2102.11086*. 2021.

[131] A. T. Cemgil, S. Ghaisas, K. Dvijotham, S. Gowal, and P. Kohli. Autoencoding Variational Autoencoder. In *Neural Information Processing Systems*. 2020.

[132] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*. 2020.

[133] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. Grammar variational autoencoder. In *International Conference on Machine Learning*. 2017.

[134] A. Vahdat and J. Kautz. NVAE: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*. 2020.

[135] C. Dennis and J. Mitterer. Introduction to Psychology: Gateways to Mind and Behavior. In *Wadsworth: Thomson Learning*. 2004.

[136] S. Blackburn. The Oxford Dictionary of Philosophy. In *OUP Oxford*. 2005.

[137] M. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *International Conference on Machine Learning*. 2011.

[138] Y. Gal, R. McAllister, and C. E. Rasmussen. Improving PILCO with Bayesian Neural Network Dynamics Models. In *International Conference on Machine Learning Workshop on Data-Efficient Machine Learning*. 2016.

[139] B. Amos, L. Dinh, S. Cabi, T. Rothörl, S. G. Colmenarejo, A. Muldal, T. Erez, Y. Tassa, N. de Freitas, and M. Denil. Learning Awareness Models. In *International Conference on Learning Representations*. 2018.

[140] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep Reinforcement Learning in a Handful of Trials Using Probabilistic Dynamics Models. In *Advances in Neural Information Processing Systems*. 2018.

[141] S. Chiappa, S. Racanière, D. Wierstra, and S. Mohamed. Recurrent Environment Simulators. In *International Conference on Learning Representations*. 2017.

[142] D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*. 2018.

[143] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning Latent Dynamics for Planning from Pixels. In *International Conference on Machine Learning*. 2019.

[144] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. In *Nature*. 2015.

[145] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. In *arXiv preprint arXiv:1707.06347*. 2017.

[146] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust Region Policy Optimization. In *International Conference on Machine Learning*. 2015.

[147] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU. In *International Conference on Learning Representations*. 2017.

[148] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *arXiv preprint arXiv:1912.01603*. 2019.

[149] S. Chopra, R. Hadsell, and Y. LeCun. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2005.

[150] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2006.

[151] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In *arXiv preprint arXiv:2002.05709*. 2020.

[152] D. P. Bertsekas. Dynamic Programming and Optimal Control. In *Athena Scientific Belmont*. 1995.

[153] J. Staines and D. Barber. Variational Optimization. In *arXiv preprint arXiv:1212.4507*. 2012.

[154] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. In *arXiv preprint arXiv:1703.03864*. 2017.

[155] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. In *arXiv preprint arXiv:1606.01540*. 2016.

[156] M. Zhang, P. Hayes, T. Bird, R. Habib, and D. Barber. Spread Divergences. In *International Conference on Machine Learning*. 2020.

[157] W. B. Powell. Approximate Dynamic Programming: Solving the Curses of Dimensionality. In *Wiley Series in Probability and Statistics*. 2007.

[158] S. Mahadevan. Learning Representation and Control in Markov Decision Processes. In *Now Foundations and Trends*. 2009.

[159] D. P. Bertsekas. Feature-Based Aggregation and Deep Reinforcement Learning: A Survey and Some New Implementations. In *IEEE/CAA Journal of Automatica Sinica*. 2018.

[160] L. Kuvayev and R. S. Sutton. Model-Based Reinforcement Learning with an Approximate, Learned Model. In *Yale Workshop on Adaptive and Learning Systems*. 1996.

[161] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*. 2019.

[162] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. In *arXiv preprint arXiv:2006.10742*. 2020.

[163] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2013.

[164] A. v. d. Oord, Y. Li, and O. Vinyals. Representation Learning with Contrastive Predictive Coding. In *arXiv preprint arXiv:1807.03748*. 2018.

[165] A. Srinivas, M. Laskin, and P. Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. In *arXiv preprint arXiv:2004.04136*. 2020.

[166] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *International Conference on Learning Representations*. 2017.

[167] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. In *arXiv preprint physics/0004057*. 2000.

[168] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. In *arXiv preprint arXiv:1612.00410*. 2016.

[169] K. Hu. Reuters report. In *https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/*. Feb. 2023.

[170] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. In *arXiv preprint arXiv:2204.02311*. 2022.

[171] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*. 2022.

[172] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. In *arXiv preprint arXiv:2305.18290*. 2023.

[173] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. In *arXiv preprint arXiv:2212.08073*. 2022.

[174] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*. 2017.

[175] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. In *Oxford University Press, Biometrika Trust*. 1952.

[176] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. In *arXiv preprint arXiv:1909.08593*. 2019.

[177] N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*. 2020.

[178] D. Cacciarelli and M. Kulahci. A survey on online active learning. In *arXiv preprint arXiv:2302.08893*. 2023.

[179] S. Kadavath, T. Conerly, A. Askell, T. Henighan, D. Drain, E. Perez, N. Schiefer, Z. Hatfield-Dodds, N. DasSarma, E. Tran-Johnson, S. Johnston, S. El-Showk, A. Jones, N. Elhage, T. Hume, A. Chen, Y. Bai, S. Bowman, S. Fort, D. Ganguli, D. Hernandez, J. Jacobson, J. Kernion, S. Kravec, L. Lovitt, K. Ndousse, C. Olsson, S. Ringer, D. Amodei, T. Brown, J. Clark, N. Joseph, B. Mann, S. McCandlish, C. Olah, and J. Kaplan. Language Models (Mostly) Know What They Know. In *arXiv preprint arXiv:2207.05221*. 2022.

[180] Y. Chen, R. Wang, H. Jiang, S. Shi, and R. Xu. Exploring the use of large language models for reference-free text quality evaluation: A preliminary empirical study. In *arXiv preprint arXiv:2304.00723*. 2023.

[181] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*. 2022.

[182] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*. 2017.

[183] H. Dong, W. Xiong, D. Goyal, R. Pan, S. Diao, J. Zhang, K. Shum, and T. Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. In *arXiv preprint arXiv:2304.06767*. 2023.

[184] K. Margatina, T. Schick, N. Aletras, and J. Dwivedi-Yu. Active Learning Principles for In-Context Learning with Large Language Models. In *arXiv preprint arXiv:2305.14264*. 2023.

[185] M. Marion, A. Üstün, L. Pozzobon, A. Wang, M. Fadaee, and S. Hooker. When less is more: Investigating data pruning for pretraining llms at scale. In *arXiv preprint arXiv:2309.04564*. 2023.

[186] Z. Chen, Y. Deng, H. Yuan, K. Ji, and Q. Gu. Self-play fine-tuning converts weak language models to strong language models. In *arXiv preprint arXiv:2401.01335*. 2024.

[187] W. Yuan, R. Y. Pang, K. Cho, S. Sukhbaatar, J. Xu, and J. Weston. Self-rewarding language models. In *arXiv preprint arXiv:2401.10020*. 2024.

[188] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. In *OpenAI Blog - https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf*. 2019.

[189] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*. 2023.

[190] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. In *arXiv preprint arXiv:2106.09685*. 2021.

[191] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *arXiv preprint arXiv:2305.14314*. 2023.

[192] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 2019.

[193] G. E. Hinton and D. Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*. 1993.

[194] J. Duda. Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. In *arXiv preprint arXiv:1311.2540*. 2013.

[195] Y. Yang, R. Bamler, and S. Mandt. Improving inference for neural image compression. In *arXiv preprint arXiv:2006.04240*. 2020.

[196] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *arXiv preprint arXiv:1701.05517*. 2017.

[197] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*. 2015.

[198] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. OpenAI Baselines. In *https://github.com/openai/baselines*. 2017.

[199] A. Paszke. Reinforcement Learning (DQN) Tutorial. In *https://github.com/pytorch/tutorials*. 2020.

[200] S. Guo, B. Zhang, T. Liu, T. Liu, M. Khalman, F. Llinares, A. Rame, T. Mesnard, Y. Zhao, B. Piot, et al. Direct Language Model Alignment from Online AI Feedback. In *arXiv preprint arXiv:2402.04792*. 2024.