Applications of Time-Series Generative Models and Inference Techniques



Michael Teng

Wolfson College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2022

Acknowledgements

To my parents and siblings, thank you for your unconditional support. Without it, none of this would have been possible.

To my computer science mentors and colleagues, Luke, Kevin, Dmitriy, Nick, Dustin, and Angel, thank you for your encouragements on and off the job. To Michiel van de Panne, Pawan Kumar, and Güneş Baydin, thank you for the guidance throughout my DPhil studies. My greatest thanks to Frank Wood, for the most supportive and patient mentorship throughout.

To all my friends throughout the years, thank you for all the deep conversations, personal motivations, and formative experiences. Thank you to Eric, Dan, Yanyang, Marcelo, Justin, Sai, Jack, Amogh, Richard, Karna, and Si Si, for all the adventures and shared curiosity. To Chris, Kevin, Victor, Lawrence, Webster, Nauman, Madhav, Kuno, Emil, Jihoon, and Kyu, nothing has prepared me more for the past four years than the four years we spent in undergrad. Finally, to Shyam, Bowei, John, Nancy, Rob, Tuan Anh, Bradley, Andy, Will, Andreas, Adam, Wilder, Matej, Rango, and Nghia, thank you for all the insight and wisdom.

Lastly, this thesis is dedicated to Sabina. Thank you for being in my life.

Abstract

In this dissertation, we apply deep generative modelling, amortised inference and reinforcement learning methods to real-world, practical phenomenon, and we ask if these techniques can be used to predict complex system dynamics, model biologically plausible behaviour, and guide decision making. In the past, probabilistic modelling and Bayesian inference techniques have been successfully applied in a wide array of fields, achieving success in financial market prediction, robotics, and the natural sciences. However, the use of generative models in these contexts has usually required a rigid set of linearity constraints or assumptions about the distributions used for modelling. Furthermore, inference in non-linear models can be very difficult to scale to high-dimensional models. In recent years, deep learning has been a key innovation in enabling non-linear function approximation. When applied to probabilistic modelling, deep non-linear models have significantly improved the generative capabilities of computer vision models. While an important step towards general artificial intelligence, there remains a gap between the successes of these early single-time-step deep generative models and the temporal models that will be required to deploy machine learning in the real-world.

We posit that deep non-linear time-series models and sequential inference are useful in a number of these complex domains. In order to test this hypothesis, we made methodological developments related to model learning and approximate inference. We then present experimental results, which address several questions about the application of deep generative models.

First, can we train a deep temporal model learning complex dynamics to perform sufficiently accurate inference and predictions at run-time. Here, "sufficient accuracy" means that the predictions and inferences made using our model lead to stronger performance than that given by a heuristic approach on some downstream task performed in real-time. We specifically model large compute cluster hardware performance using a deep generative model in order to use the model to tackle the downstream task of improving the overall throughput of the cluster. Generally, this question is useful to answer for a number of wider applications similar to ours which may use such modelling techniques to intervene in real-time. For example, we may be interested in applying generative modelling and inference to come up with better trading algorithms with the goal of increasing returns. We may also

wish to use a deep generative epidemiology model to determine government policies that help prevent the spread of disease. Simply put, we want to ask the question, "are deep generative models powerful enough to be useful?"

Next, are deep state-space models important for the generative quality of animal-like behaviour? Given a perceptual dataset of animal behaviour, such as camera views of fruit-flies interactions or collections of human handwriting samples, can a deep generative model capture the latent variability underlying such behaviour. As a step towards artificial intelligence that mirrors human and other biological organisms, we must assess whether deep generative modelling is a viable approach to capture what may be one of the most stochastic and challenging phenomenon to model.

Finally, is inference a useful perspective in decision making and reinforcement learning? If so, can we improve the uncertainty estimation of different quantities used in classic reinforcement learning to further take advantage of an inference perspective? Answering these questions may help us determine if a "Reinforcement Learning as Inference" framework coupled with a distributional estimate of the sum of future rewards can lead to better decision making under the control setting.

Although our findings are positive in terms of these questions, they come with caveats for each. First, deep generative models must be accurate to be useful for downstream tasks. Second, modelling biologically plausible behaviour is difficult without additional partial supervision in the latent space. Third, while we have made orthogonal progress in using the inference perspective for policy learning and leveraging a distributional estimate in reinforcement learning, it remains unclear how to best combine these two approaches. This thesis presents the progress made in tackling these challenges.

Contents

1 Introduction **1**

1.1 Motivation 1
1.2 Contribution 5
1.3 Outline 6

2 Background **9**

2.1 Probabilistic Modelling and Inference 10
2.1.1 Probability Theory 10
2.1.2 Generative Modelling 13
2.1.3 Model Learning 16
2.1.4 Latent Variables and Expectation-Maximisation 21
2.1.5 Variational Inference and Importance Sampling 24
2.2 Sequential Modelling and Decision Making 28
2.2.1 Hidden Markov Models and Linear Dynamical Systems . . . 29
2.2.2 Sequential Monte Carlo 35
2.2.3 Markov Decision Processes and Reinforcement Learning . . . 38
2.3 Deep Generative Modelling and Amortised Inference 41
2.3.1 Deep Learning and Stochastic Gradient Descent 42
2.3.2 Variational Auto-Encoder 43
2.3.3 Implicit Deep Generative Models 50
2.3.4 Deep Sequential Modelling 51
2.4 Control as Inference 54
2.4.1 Optimal Policies via Inference 55
2.4.2 Actor-Critic Algorithms 57
2.4.3 Streaming Priors 58
2.4.4 Bayesian Statistics and Applications 60

3 Distributed Training with Deep Markov Models **61**

3.1 Introduction 61
3.2 Background and Motivation 63
3.2.1 Effect of Stragglers 65
3.3 Methodology 65

3.3.1 Modelling Compute Cluster Worker Performance 68
3.3.2 Neural Network Architectures 74
3.3.3 Parameter Server and Worker Design 76
3.4 Experiments 77
3.4.1 Small Compute Cluster 78
3.4.2 Large Scale Computing 79
3.4.3 Experimental Details 80
3.5 Discussion 81

4 Semi-supervised Sequential Generative Models 85
4.1 Introduction 85
4.2 Background 87
4.2.1 Variational and Importance-Weighted Auto-encoders 87
4.2.2 Semi-supervised Variational Auto-Encoder 88
4.2.3 Variational Recurrent Neural Network 89
4.3 Caffeinated Wake Sleep 89
4.3.1 Generative Model 90
4.3.2 Inference 91
4.3.3 Optimisation 91
4.4 Experiments 96
4.4.1 Semi-supervised MNIST 96
4.4.2 Handwriting 98
4.4.3 Fly tracking 99
4.4.4 Experimental Details 101
4.5 Discussion 102

5 KL-Regularised Reinforcement Learning as Inference 105
5.1 Introduction 105
5.2 Background 107
5.2.1 Markov Decision Processes 108
5.2.2 Soft Actor Critic 108
5.2.3 Reinforcement Learning as Inference 110
5.3 Policy Gradient Estimators Via The Reverse KL Divergence 112
5.4 Reweighted Policy Gradient Estimators Via the Forward KL . . . 115
5.5 Experiments 118
5.6 Related Work 119
5.7 Discussion 121

6 Exploration with Multi-Sample Target Values for Distributional Reinforcement Learning **125**
6.1 Introduction . . . 125
6.2 Background . . . 128
6.2.1 Soft Actor-Critic . . . 129
6.2.2 Implicit Quantile Networks . . . 130
6.2.3 Distributional SAC . . . 131
6.3 Extra Exploration with Distributional Critics . . . 132
6.3.1 Multi-sample Target Value . . . 134
6.3.2 UCB Exploration . . . 135
6.3.3 Algorithm Description . . . 135
6.4 Experiments . . . 136
6.4.1 Distribution Matching Study . . . 137
6.4.2 MuJoCo and Box2d . . . 138
6.4.3 Humanoid Ablation Study . . . 139
6.5 Discussion . . . 141

7 Conclusion **143**

Bibliography **147**

x

Spiritual unhealthiness and misfortunes can generally be traced to excessive love of something which is subject to many variations.

— Spinoza

1 Introduction

Contents

1.1 Motivation **1**
1.2 Contribution **5**
1.3 Outline **6**

1.1 Motivation

The goal of artificial intelligence and machine learning (AI/ML) is to develop computational models of cognition. As humans, we are able to learn about our environment, make decisions under uncertainty, update our understanding with new experience, and generalise our mental models to unseen situations. One possible way to formalise these tasks is via Bayesian statistics, which confers a principled way to update a probabilistic model using prior knowledge and observations. Thus, probabilistic modelling and Bayesian inference is a framework that could, in principle, mathematically describe certain cognitive processes that occur while interacting with the world [Griffiths et al., 2008]. Statistical machine learning combines these mathematical principles with the computational power of modern hardware. Since much of intelligence seems to resemble the task of manipulating probabilities under this framework, it is the our hope that by using computers to

solve these same problems we can arrive at more powerful machine intelligence [Parr et al., 2018].

The foundational models and inference algorithms in statistical ML have been successfully applied to a number of generalised tasks, such as clustering [Biernacki et al., 2000, McLachlan and Basford, 1988], dimensionality reduction [Blei et al., 2003], classification and regression [Bernardo et al., 1998, Wang et al., 2007, Kocijan et al., 2004], and time-series analysis [Varga and Moore, 1990, Beal et al., 2002, Fine et al., 1998]. However, this success has largely been predicated on various constraints being placed on the models, including linearity assumptions, distributional flexibility, and specific factorisation of the model. Without these limitations, inference, the mechanism used to update our model and reason under new observations, can become intractable.

Deep learning, in contrast, was developed to tackle discriminative tasks, and, in the language of statistics, conditional probability estimation. In discriminative learning, we require a large dataset of labeled pairs of training data, and our goal is to use the function approximating power of deep neural networks (DNN) to learn the mapping from input to output pairs. The main mechanism underlying deep learning is the back-propagation algorithm, which computes partial derivatives of the loss function with respect to neural network weights using the chain rule [LeCun et al., 2015]. Along with an i.i.d. assumption, this enables DNNs to be efficiently trained using stochastic optimisation. Due to the availability of large labeled datasets and growing computational resources, deep learning has achieved great success in a number of domains, such as computer vision [Szeliski, 2010, Forsyth and Ponce, 2011], speech recognition [Amodei et al., 2016], and other tasks where the goal is to train a discriminative model to learn arbitrary, complex non-linear relationships [Vaswani et al., 2017, Brown et al., 2020].

However, while there has been some recent research on the possibility of neuro-mechanisms that simulate back-propogation [Lillicrap et al., 2020], there is still a large part of human cognition that is more readily explainable in the Bayesian paradigm. In particular, humans are able to learn from limited data and generalise

to new scenarios, which are two known drawbacks of deep learning methods [Osawa et al., 2019]. In fact, Knill and Pouget [2004] attributes these cognitive modes to a brain-based "Bayesian coding" framework. Most notably, they hypothesise that human cognition may represent perceptual input and latent information as probability distributions, updating these internal distributions in an uncertainty-aware and robust manner, similar to the way Bayesian statistics functions. Furthermore, several works in machine learning demonstrate that applying Bayesian principles to neural networks allows trained neural networks to output well-calibrated uncertainties, subsequently leading to stronger performance on novel datasets in both robustness measures (i.e. fewer errors on out-of-distribution data; Osawa et al. [2019]) and for continual learning (non-catastrophic forgetting and learning with limited examples; Achille et al. [2018]).

But rather than be at odds with one another, neural networks and Bayesian statistics can be informative of one another [Griffiths et al., 2012, Lee et al., 2017]. Much of the recent efforts in machine learning and artificial intelligence since the success of deep learning has been to combine the two methods in order to address the shortcomings of each. One line of research does so by framing neural network training as Bayesian inference, by placing priors on the network parameters [Blundell et al., 2015] or drawing connections between deep learning methods and Bayesian inference [Gal and Ghahramani, 2016]. Another approach, referred to as deep generative modelling, uses neural networks to parameterise the probabilities used in statistical ML. Unlike the simple models we previously mentioned, this line of research enables the modelling of non-linear dependencies between random variables. Our focus is on the latter.

The framework of deep generative modelling can be viewed as casting a classic machine learning task as an inference task in a particular, probabilistic model with complex non-linear dependency structure. For example, computer vision researchers are often interested in inferring an object's true, intrinsic properties given a visual representation of it. With a conventional deep learning approach, this is usually solved by using a learning a discriminator that maps the input

representation to a corresponding high-level property. In deep generative modelling, however, we can instead posit a forward, generative model of possible images given a property class. With a prior over the classes, we can then accomplish the same vision task by obtaining the posterior distribution over properties given the visual representation. The use of deep neural networks, in contrast to simple, linear models, gives us additional modelling capacity, resulting in more accurate and realistic samples.

Of course, cognitive processes outside of perception may seem different than this, but they are actually very similar. The language of probabilities and Bayesian statistics gives us a way to reframe many problems as an inference task, and in this dissertation, we are most interested in time-series analysis. Human decision making, creativity, and the society we live in, have complex underlying structure, specifically with long-ranging, non-linear correlations and partially observable state throughout [Igl et al., 2018]. If we accept these assumptions about the world, along with the evidence for Bayesian models of cognition, then it is very likely that our actions through time is from modelling an evolving state of the world, performing inference when possible to update this state, and acting according to the predictive probabilities in our model.

This dissertation presents findings that push forward the argument for time-series models and sequential inference. We initially apply these models in a variety of sequential modelling domains, including computing system dynamics, fly behaviour, and human handwriting. Then, we focus in on the domain of reinforcement learning, which is the specific context of reward-based, sequential decision making. We first show that this control problem can be framed as inference, and then we investigate improvements to distributional reinforcement learning. In terms of the stated goal of developing computational models of cognition, the combination of probabilistic modelling and inference with reinforcement learning is a leading candidate. For one, animals are known to exhibit a learned decision making process using external feedback, which is exactly the cognitive process embodied in the reinforcement learning framework [Dabney et al., 2018b, Botvinick et al.,

2009, 2019]. Second, many recent advances in this framework have been made by bringing the tools of generative modelling and approximate inference to bear [Levine, 2018, Ha and Schmidhuber, 2018].

1.2 Contribution

This dissertation is presented as an integrated thesis, based on the following papers:

- Michael Teng & Frank Wood, Bayesian Distributed Stochastic Gradient Descent. *Advances in Neural Information Processing Systems*, 6378-6388, 2018.

- Michael Teng, Tuan Anh Le, Adam Scibior, & Frank Wood, Semi-supervised Sequential Generative Models. *Uncertainty in Artificial Intelligence*, PMLR 124:649-658, 2020.

- Jonathan Lavington Wilder, Michael Teng, Mark Schmidt, & Frank Wood, A Closer Look at Gradient Estimators with Reinforcement Learning as Inference. *NeurIPS Workshop on Deep Reinforcement Learning*, 2021.

- Michael Teng, Michiel Van de Panne, & Frank Wood, Extra Exploration with Multi-Sample Target Values for Distributional Reinforcement Learning. *Preprint*, 2021.

We first apply a deep Markov model to modelling node performance in a scientific computing cluster. Our goal is to then use this model at run-time to increase computation throughput during neural network training. We show that the amortised inference artefact we train is useful at test-time in a deployed application. A common criticism of Bayesian inference is that the posterior of interest can be computationally intractable and therefore not practically useful. Our first contribution is to show how deep amortised inference is a counter argument to this, and that it can be a particularly valuable tool for real-time, practical applications.

Second, we show that these models better approximate biologically plausible behaviour, and specifically by modelling a stochastic latent state instead of a

deterministic hidden state. We assume in this dissertation that non-linear models are more powerful than linearised counterparts. A question that remains for us is whether latent random variables are helpful. Our second contribution finds that increased stochasticity in modelling animal behaviour seems to improve uncertainty quantification and improves generative quality of samples.

Next, we transition into reinforcement learning domain. We start by presenting detailed proofs connecting the control problem to approximate inference. We derive two objectives commonly used in classic reinforcement learning, by starting with a generative model of the control task and recasting the decision making problem as amortising inference in this model. We examine the pros and cons of each objective, showing how the inference perspective can lead to more principled algorithm design for policy learning.

Finally, while the reinforcement learning as inference framework is powerful for policy learning, it still relies on bootstrapped value-function approximation to be practically useful. In our final contribution, we investigate an improvement to value-estimation in RL, in using distributional value-functions. While not directly related to Bayesian inference, we are motivated here by the idea that the intrinsic uncertainty of an environment is better modelled using a full distribution of possible values, and we hope this approach is eventually more compatible with reinforcement learning as inference.

1.3 Outline

The rest of this thesis is as follows. In Chapter 2, we review relevant background and literature. We begin with defining probabilistic modelling and inference in Section 2.1, highlighting toy models and foundational inference methods. Next, we expand these ideas into sequential modelling and decision processes in Section 2.2. Section 2.3 reviews recent literature relevant to deep learning approaches to generative modelling, approximate inference, and sequential models. Finally, Section 2.4 introduces the "reinforcement learning as inference" framework, which casts the control problem as approximate inference.

Following the background, we present our contributions as four chapters, each dedicated to a chapter in the thesis. Chapter 3 presents our compute cluster model and amortised inference for distributed computing experiments. Chapter 4 presents our semi-supervised method for learning structured, time-series generative models. Chapter 5 begins our exposition into reinforcement learning as inference, with derivations and experiments on continuous control tasks. Chapter 6 presents our modifcations to distributional value-functions, which encourages additional exploration for continuous control. Where necessary, we defer introductory background to the paper chapters, and we will note this in the general background given in Chapter 2. Finally, we conclude in Chapter 7 with an overall discussion and future directions.

2
Background

Contents

2.1 Probabilistic Modelling and Inference 10
2.1.1 Probability Theory 10
2.1.2 Generative Modelling 13
2.1.3 Model Learning 16
2.1.4 Latent Variables and Expectation-Maximisation 21
2.1.5 Variational Inference and Importance Sampling 24
2.2 Sequential Modelling and Decision Making 28
2.2.1 Hidden Markov Models and Linear Dynamical Systems 29
2.2.2 Sequential Monte Carlo 35
2.2.3 Markov Decision Processes and Reinforcement Learning 38
2.3 Deep Generative Modelling and Amortised Inference . 41
2.3.1 Deep Learning and Stochastic Gradient Descent 42
2.3.2 Variational Auto-Encoder 43
2.3.3 Implicit Deep Generative Models 50
2.3.4 Deep Sequential Modelling 51
2.4 Control as Inference 54
2.4.1 Optimal Policies via Inference 55
2.4.2 Actor-Critic Algorithms 57
2.4.3 Streaming Priors 58
2.4.4 Bayesian Statistics and Applications 60

We begin our background with a brief introduction to probabilistic modelling and inference. We then connect these ideas, along with model learning, to early techniques in the time-series domain. Of these sequential problems, we dive deeper

into reinforcement learning, and draw connections to inference in a probabilistic model. In the remaining sections, we cover the advances brought forth by deep learning in deep generative models, amortised inference, and deep reinforcement learning.

2.1 Probabilistic Modelling and Inference

In this section, we introduce the framework of Bayesian statistics, including basic probability theory, latent variable models, and Bayesian inference. Within this framework, we are interested in both model learning and posterior inference. To start, we show a toy example to demonstrate these tasks in a simple model. Then, we highlight the foundational algorithm used for model learning, called "expectation-maximisation". Finally, we introduce several computational approaches to various tasks in Bayesian statistics, namely Monte Carlo estimation, importance sampling, and variational inference. For readers familiar with these foundational topics in probabilistic modelling and inference, we refer them to Chapter 2.2.

2.1.1 Probability Theory

We start with an introduction to probability theory. A *probability space* is a three-tuple, $(\Omega, \mathcal{F}, \mathbb{P})$, referred to as the *sample space*, *event space*, and *probability function*, respectively. The sample space is the set of all possible outcomes, $\omega \in \Omega$. The event space is a is a collection of subsets of Ω, obeying the properties of σ-algebras. There can be multiple event spaces for a given sample space. For example, both $\{\varnothing, \Omega\}$ and $\text{pow}(\Omega)$ (denoting the powerset, i.e. all subsets of Ω) are valid event spaces. Finally, the probability function, $\mathbb{P} : \mathcal{F} \to [0, 1]$, assigns a probability to the events, $E \in \mathcal{F}$, and $\mathbb{P}(\Omega) = 1$. For a more detailed overview of measure theory, including literature on σ-algebras, we refer the reader to Tao [2011].

Now, given a probability space, $(\Omega, \mathcal{F}, \mathbb{P})$, let X be a measurable function, $X : \Omega \to \mathbb{R}$, such that for each Borel set $B \in \mathcal{B}$, the set, $X^{-1}(B) \triangleq \{\omega : X(\omega) \in B\}$ is in the event space, $\mathcal{F}$ (i.e. $X^{-1}(B) \in \mathcal{F}$). Then, we can say that X is a *random*

variable induced by the probability space, $(\Omega, \mathcal{F}, \mathbb{P})$. We can further define a measure, $P_X(B) = \mathbb{P}[X^{-1}(B)]$, and the space, $(\mathbb{R}, \mathcal{B}, P_X)$ is a probability space itself.

Random variables are either discrete or continuous. If P_X gives measure 1 to a countable set of real numbers, then X is a *discrete* random variable. Otherwise, if for every countable set of reals, P_X gives measure 0, then X is a *continuous* random variable.

We can now define several important objects used in Bayesian statistics. First, let there be some function, $p : \mathbb{R} \to [0, \infty)$, such that:

$$P_X(\{a \leq X \leq b\}) = \int_a^b p(x)dx \quad \text{and} \quad \int_{-\infty}^{\infty} p(x)dx = 1, \tag{2.1}$$

for continuous random variables, and:

$$P_X(\{X = x\}) = p(x) \quad \text{and} \quad \sum_x p(x) = 1, \tag{2.2}$$

for a discrete random variable. We call this function $p(x)$ the probability density of some outcome, x (some refer to $p(x)$ as a probability mass function for discrete variables, but we will use the term interchangeably). The cumulative distribution function, $F_X(x)$ is obtained by summing or integrating over the density up to some value, x:

$$\begin{aligned} F_X(x) &= P(X \leq x) \\ &= \int_{-\infty}^{x} p(t)dt \qquad \text{(continuous)} \qquad (2.3) \\ &\quad \text{or} \\ &= \sum_{a \leq x} p(a) \qquad \text{(discrete)} \qquad (2.4) \end{aligned}$$

We are usually interested in mathematically writing commonly seen distributions and their densities. For example, the most ubiquitous density of a Gaussian random variable is given as:

$$\mathcal{N}(x; \mu, \Sigma) = \frac{\exp\Big(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\Big)}{\sqrt{\det(2\pi\Sigma)}}, \tag{2.5}$$

where μ, Σ denote *parameters* specific to the Gaussian distribution. We will denote common densities by their distribution name, unless otherwise specified. Next,

we define the expected value, or *expectation/mean*, of a random variable X as an integral of the random variable over its measure, P:

$$\mathbb{E}[X] = \int X dP_X, \tag{2.6}$$

which can be converted to a Riemann integral or a summation for each type of random variable:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} xp(x)dx \qquad \text{(continuous)} \tag{2.7}$$

or

$$= \sum_x xp(x) \qquad \text{(discrete)} \tag{2.8}$$

Next, we need to extend these definitions to the case of multiple random variables. Given two random variables, X, Z, we can let the joint probability density, $p_{X,Z} : \mathbb{R} \times \mathbb{R} \to [0, \infty)$, denote the relationship between them, such that the original densities, $p(x)$ and $p(z)$, can be obtained by *marginalising* out one or the other:

$$p(z) = \int p(z, x)dx \tag{2.9}$$

The use of multiple random variables also gives us a way to express *conditioning*, where a conditional density, $p(x|z)$ gives a probability density for X, *given* some fixed value, z for Z. Conditional densities are also related to the joint density:

$$p(x, z) = p(x)p(z|x) \tag{2.10}$$

Finally, we define the *support* of P to be the set of non-zero probability events, $\{B \in \mathcal{B} : P_X(B) > 0\}$, and correspondingly, the range of values for x, for which the density function is non-zero, i.e. $p(x) > 0$.

The intuition behind these definitions serves as a foundation for statistical ML. Probability theory, as the name suggests, gives us a way to assign probabilities to different outcomes of "experiments" or processes in nature. In later sections, we refer to a function, f, as a *likelihood*, if it is a probability of such an observed outcome. Our goal, then, is to use the language of probability densities to express

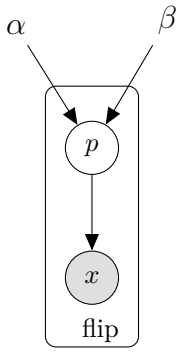


Figure 2.1: A graphical model of a coin flip.

specific cognitive processes as models relating various densities to one another. With this in mind, we will next move on to the practical definitions of the models we will be working with.

2.1.2 Generative Modelling

In this section, we define models, latent variables, model learning, and expectation maximisation.

Model Definition

How do we define a model? We start with the assumption that there is an underlying generative process for an outcome represented by a random variable, X. We are interested in finding a way to generate examples or realisations of the output, x. A *model* is our specification of this generative process, which acts to faithfully represent the true underlying mechanics. For example, when we see coins being flipped, in our heads we have an idea of what might happen despite the complex, physical interactions of the coin, the hand, air resistance, and the flip itself. This intuition is encoded in our "mental model" of the coin flip, which is the knowledge from *past experience* that half of the time it will land heads and the other half tails. If we know that the coin is biased one way or another, we can also tune our mental model of the outcome chances towards what we know. More importantly, however, this intuition can be encoded using mathematics and programming languages for any physical process. Moreover, tuning models based on experience or *data* can be

described by mathematical and computational operations as well. Doing so allows both the generative capabilities of our model, as well as other modelling tasks, to scale with computation and memory.

Model Denotation

In order to use models in practical applications, we must first be able to specify them. Drawing from both statistics and computer science, we have a variety of ways to do so. From a statistician's perspective, we can use the tools of probability distributions and sampling to denote the aforementioned coin flip model. Specifically, we first sample the "fairness" of the coin, z, from a Beta distribution and then sample the outcome of a single flip as a single Bernoulli trial:

$$z \sim \text{Beta}(\alpha, \beta)$$
$$x \sim \text{Bernoulli}(z)$$

While this notation is powerful, specifying models purely mathematically can become unwieldy when there are additional transformations, both linear and non-linear. A more concise, but still powerful way to denote a generative model is with graphical models. In Figure 2.1, we specify the same coin flip event as a graph where nodes represent random variables and arrows represent the conditional probability relationships. Most importantly, the graphical model gives rise to a flexible way to specify a generative model, as the conditional probabilities can be further specified using mathematical transformations or programs. Graphical model notation also gives us a way to write the joint probability density of our model. For the coin flip model, this becomes:

$$p(x, z; \alpha, \beta) = p(x|z)p(z; \alpha, \beta), \tag{2.11}$$

where the α, β, as given in the graphical model, are parameters of the distribution over z.

Finally, we can specify models programmatically, most popularly encapsulated in the framework of probabilistic programming languages (PPLs). In this thesis,

however, we will treat our models as more general computational models without adhering to the formalism. For a more detailed discussion of the PPL framework, we refer readers to van de Meent et al. [2018]. Additionally, many existing programatic models are referred to as simulators, where parts of the simulation are draws from a random variable. In the following program, we use a python program as the generative model for our coin flip.

```
1   import random
2
3   def flip(alpha, beta):
4       z = random.betavariate(alpha, beta)
5       return 'Heads' if random.random() < z else 'Tails'
```

In the above program, we simulate the first sampling procedure, $z \sim \text{Beta}(\alpha, \beta)$ with line 4, and we encode the second sampling procedure in line 5. Although we have been operating with the canonical example of a coin flip, we note that the important idea is that our generative model can be specified with both graphical models and code. This allows us to include a more complex transformation between sampling procedures, and we are limited only by what is possible within the programming language of choice.

Latent Variable Models

In our coin flip example so far, we have a model with two random variables, z and x, but have only referred to x as the "output". We now formalise this intuition with the idea of latent variable models. For the purpose of this thesis, all models we consider will have some notion of a set of observed variables and a set of *latent* or "hidden" variables. This is important for us because the observed variables correspond to our model of some observable data from the real world (hence our desire to model it). The set of remaining variables are those that we cannot directly observe or know.

We now give a general definition of a model. In this thesis, we will only deal with parametric models, which is collection of random variables that can be written as a joint probability density (and its optional factorisation) with parameters, θ.

We usually write this as:

$$p_\theta(x, z) \quad \text{or} \quad p(x, z; \theta) \tag{2.12}$$

It should be noted that θ is the complete set of parameters for the joint. When the model further factorises, certain marginals or conditional probabilities may or may not be parametrised. Often times, we specify a "prior", which is a user-specified marginal distribution over the latent variables. In such cases, we can write the joint distribution as:

$$p_\theta(z, x) = p(z)p_\theta(x|z) \tag{2.13}$$

2.1.3 Model Learning

In the previous section, we are given a way to specify models, but they are often imperfect, even if we take for granted the conditional dependencies chosen are correct. The question then becomes, how can we do something useful with our data, either to improve the model or to reduce the uncertainty within our model. The latter is the traditional approach taken by statisticians via Bayesian inference. We first highlight an example of this to introduce posterior inference in models and to later draw a distinction between "model learning" and Bayesian inference.

Returning to our canonical coin flip example, we can imagine that there is data, $\mathcal{D} = \{x^{(1)}, x^{(2)}, ..., x^{(N)}\}$, consisting of N true outcomes of the coin we want to model. For notational clarity, we can fix the parameters of the Beta prior, $p(z; \alpha, \beta)$, and focus on the random variables in our model and their joint density, $p(z, x) = p(x|z)p(z)$. As specified, we only have a prior on the latent variable, z, and a conditional probability density, $p(z|x)$. However, probability theory and Bayes' rule gives use a way to manipulate the joint density in order to obtain the conditional probability of the latents given the output:

$$p(z|x) = \frac{p(x, z)}{p(x)} = \frac{p(x, z)}{\int p(x|z)p(z)dz} \tag{2.14}$$

With this, we can fine-tune our model to be more accurate by conditioning on our observations. In the coin flip example, the latent variable denoting the

"fairness", z, is drawn from a Beta-distribution parametrised by α, β. If we supply our guess for the prior parametrisation, we have a fully specified model, using a Beta-prior and Bernoulli output conditioned on the latent:

$$p(z; \alpha, \beta) = \frac{z^{\alpha-1}(1-z)^{\beta-1}}{\mathrm{B}(\alpha, \beta)} \quad \text{where} \quad \mathrm{B}(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}, \quad z \in [1, 0] \tag{2.15}$$

$$p(x|z) = z^x(1-z)^{(1-x)}, \quad x \in \{1, 0\}, \tag{2.16}$$

where Γ denotes the gamma function, $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$ for $z \in \mathbb{C}$ and $\mathrm{Re}(z) > 0$.

Intuitively, α and β govern our prior belief on the fairness of the coin and they correspond to our guesses for the previous positive and negative trial results, i.e. the number of times the coin landed heads and tails. If we want to weakly suggest that it is fair, we would choose $\alpha, \beta = 1$, or if we think the coin is definitely biased, we could choose $\alpha = 10, \beta = 1$. Now, given this model, when we condition on the observations, $x^{(1:N)}$, we can obtain the posterior density for the latent variable of interest:

$$\begin{aligned}
p(z|x^{(1:N)}) &= \frac{p(x^{(1:N)}, z)}{p(x^{(1:N)})} \\
&\propto p(x^{(1:N)}|z)p(z; \alpha, \beta) \\
&\propto \prod_{i=1}^{N} p(x^{(i)}|z)p(z; \alpha, \beta) \\
&\propto \prod_{i=1}^{N} z^{x^{(i)}}(1-z)^{(1-x^{(i)})} \frac{z^{\alpha-1}(1-z)^{\beta-1}}{\mathrm{B}(\alpha, \beta)} \\
&\propto \frac{z^{\sum_i x^{(i)}+\alpha-1}(1-z)^{(\sum_i 1-x^{(i)})+\beta-1}}{\mathrm{B}(\alpha, \beta)}
\end{aligned} \tag{2.17}$$

As we can see, the posterior here is another Beta-distribution with its parameters shifted by the actual number of heads and tails seen in our observations. This is the approach taken by Bayesian inference and in this particular model, as well as a handful of other models, the *conjugacy* of the model allows us to obtain the full posterior. When this is not possible, we resort to a sampling based approximation of the true posterior, referred to as Markov chain Monte Carlo (MCMC). These methods involve iteratively constructing a Markov chain that converges to the true posterior and is the subject of years of statistical study and application. For more

details on this, we refer the reader to Brooks et al. [2011], Andrieu and Doucet [2002]. The important note for us is that given a model, $p(z, x)$, and observations for part of the model, $x^{(i)} \in \mathcal{D}$, our approach to improving the model is to characterise the posterior over unseen variables given the observed variables, $p(z|x^{(1:N)})$. In this way, we obtain a posterior distribution over the quantity we were uncertain about when we designed our model.

However, this approach has one major drawback, which is its dependency on fully specified models, which we define as models containing only random variables and their fixed parametrisation. Specifically, we would not have been able to obtain the posterior for our coin flip example if we did not have a prior placed on the latent variable of interest, i.e. the Beta-prior, $p(z; \alpha, \beta)$. Furthermore, we gave our best guesses for the parameters of the prior, but if we were uncertain about those, we would treat α, β themselves as latent variables and so on. Constructing these models with well chosen priors becomes increasingly difficult, and furthermore, the computational cost of MCMC methods does not scale with dimensionality.

Maximum Likelihood

In the absence of a fully specified model, we instead turn to maximum likelihood estimation. This approach still requires a model, but instead of placing priors over all unknown parameters, we can directly target a subset of the parameters to be learned using data. Returning to the canonical example, instead of treating the z as a random variable, we can directly model z as a learnable value that parametrizes the output distribution, $p(x; z)$. In this case, we no longer have a latent-variable model, but instead a single density over the observed outcome, whose parameter we wish to learn. This is often the case in supervised machine learning, and serves as a good motivating example for model learning in statistical models with large data. To do so, we target the likelihood function of the observations [Goodfellow et al., 2016]:

$$\theta^* = \arg\max_{\theta} \sum_i \log p(x^{(i)}; \theta), \quad \text{where} \quad x^{(i)} \in \mathcal{D} \tag{2.18}$$

where θ denotes the entire parameter set, and $x^{(1:N)} \in \mathcal{D}$ denotes the data. For our coin flip example, we again suppose that we have a number of observed flips, $x^{(1:N)} \in \{1, 0\}$. Our goal is to find the optimal parameter for z given these observations and the maximum likelihood objective function:

$$\begin{aligned} z^* &= \arg\max_z \log p(x^{(1:N)}; z) \\ &= \arg\max_z \log \prod_i z^{x^{(i)}} (1-z)^{1-x^{(i)}} \\ &= \arg\max_z \log z^{\sum_i x^{(i)}} (1-z)^{N-\sum_i x^{(i)}} \\ &= \arg\max_z \sum_i x^{(i)} \log z + (N - \sum_i x^{(i)}) \log(1-z) \end{aligned} \tag{2.19}$$

We can then set the derivative to zero and find the maximum value analytically in this example:

$$\begin{aligned} 0 &= \frac{\delta}{\delta z} \sum_i x^{(i)} \log z + (N - \sum_i x^{(i)}) \log(1-z) \\ &= \frac{\sum_i x^{(i)}}{z^*} + \frac{N - \sum_i x^{(i)}}{(1-z^*)} \\ &= \frac{(\sum_i x^{(i)})(1-z^*) + (N - \sum_i x^{(i)}) z^*}{(1-z^*)z^*} \\ &= \frac{\sum_i x^{(i)} - (\sum_i x^{(i)}) z^* + z^* N - z^* \sum_i x^{(i)}}{(1-z^*)z^*} \\ &= \sum_i x^{(i)} + z^* N \\ &\implies z^*_{\text{MLE}} = \frac{\sum_i x^{(i)}}{N} \end{aligned} \tag{2.20}$$

Notice that in the Bayesian approach, we obtained the full posterior over the latent variable, while in this new maximum likelihood approach, we will be converging towards a point estimate from the start. Actually, in many Bayesian analyses, we can still provide a point estimate for the unknown variable, called the Maximum a Posteriori (MAP). This is done simply taking the value which maximizes the posterior density. Recall for our earlier beta posterior, we determined the posterior is a beta-distribution, $p(z; \alpha, \beta)$, with parameters, $\alpha = \sum_i x^{(i)} + \alpha_0$ and $\beta = N - \sum_i x^{(i)} + \beta_0$. For a beta-distribution, the MAP estimate is given by the mean of the distribution, $z^*_{\text{MAP}} = \frac{\alpha}{\alpha+\beta} = \frac{\sum_i x^{(i)} + \alpha_0}{N + \alpha_0 + \beta_0}$. Notice that the difference between

the optimal values for the parameter is simply the original parameters for the prior. However, in the Bayesian inference case, we had uncertainty estimates for this parameter along with its mean value. Furthermore, we can see that for this example, the MAP estimate converges to the MLE in the limit of infinite data. This is encapsulated in the Bernstein von Mises theorem, which states using MLE for parameter estimate in statistical models is good when we have large data.

Finally, we motivate the objective used in maximum likelihood estimation, i.e. to maximise the log-evidence, or $\log p(x^{(1:N)}; \theta)$. One way to view this objective is that it minimises the Kullback-Leibler (KL) divergence between the empirical data distribution, $\hat{p}(x_{\text{data}})$ and the marginal model distribution, $p(x; \theta)$. To see this, we start with the definition of the KL-divergence:

$$\begin{aligned}
\theta^* &= \arg\min_\theta \mathrm{KL}(\hat{p}(x) \parallel p(x;\theta)) \\
&= \arg\min_\theta \mathbb{E}_{\hat{p}(x)}[\log \hat{p}(x) - \log p(x;\theta)] \\
&= \arg\min_\theta \mathbb{E}_{\hat{p}(x)}[-\log p(x;\theta)] \\
&= \arg\max_\theta \mathbb{E}_{\hat{p}(x)}[\log p(x;\theta)] \\
&= \arg\max_\theta \sum_i \log p(x^{(i)};\theta) \quad \text{where} \quad x_i \in \mathcal{D}
\end{aligned} \tag{2.21}$$

For the rest of this thesis, we will approach model learning from this perspective. Rather than compute the posterior over the model parameters, we instead directly target them using maximum likelihood estimation. This is a valid approach in large data regimes, and is possible computationally due to stochastic optimisation in order to find θ^*.

Finally, we note that in this thesis, the model is always treated as a fixed, user-defined object up to its parametrisation. This is in contrast to "program induction", which means that parts of the program, graph connectivity, or high-level dependencies are not well specified. In this thesis, we will treat all programs as fixed in structure, either by the programmer or the scientist.

2.1.4 Latent Variables and Expectation-Maximisation

In the previous section, we covered model learning from a maximum likelihood perspective for a simple toy example. However, what can we do when there are additional latent variables. We now introduce the canonical Gaussian Mixture Model (GMM) and the connection between Bayesian inference and model learning. We start with a GMM model definition with K mixture components:

$$\begin{aligned}
\pi_{1:K} \in [0,1] \;\; \text{s.t.} \;\; \sum_i \pi_i = 1 \\
z \sim \text{Categorical}(\{\pi_1, \pi_2, ..., \pi_K\}) \\
\mu_{1:K},\, \sigma_{1:K} | z \;\; \text{corresponding to a mixture component } k \\
x \sim \text{Normal}(\mu_k, \sigma_k^2 | z = k) \;\; \text{given } z = k \text{ mixture component.}
\end{aligned} \tag{2.22}$$

In this model, we have observations, $x^{(1:N)}$, model parameters, $\theta = \{\pi_{1:K}, \mu_{1:K}, \sigma_{1:K}\}$, and unobserved latents, $z^{(1:N)}$. That is, for each of the observed values, we do not know which mixture component it was drawn from. Our goals is to find the optimal values for the parameters, as well as be able to infer the values for the latent variables, $z^{(1:N)}$. This has many practical applications in machine learning for clustering.

First, we reiterate that we can characterise the posterior fully in this model over all the unknown values, both for the parameter set and the latents. This requires that we target the posterior:

$$\begin{aligned}
p(z^{(1:N)}, \pi_{1:K}, \mu_{1:K}, \sigma_{1:K} | x^{(1:N)}) \\
\propto p(z^{(1:N)}|\pi) p(\pi) p(\mu_{1:K}, \sigma_{1:K}) \prod_i^N p(x^{(i)} | z^{(i)} = k, \mu_k, \sigma_k)
\end{aligned} \tag{2.23}$$

As previously mentioned, treating the parameters as a random variable requires additional model specification, and requires us to place a prior over the parameters we want to infer, i.e. defining $p(\pi)$ and $p(\mu_{1:K}, \sigma_{1:K})$. However, unlike before, this cannot be done in closed form and instead requires computational methods, such as Gibbs sampling or MCMC sampling in order to obtain the posterior.

Expectation-Maximisation

Instead of characterising the full posterior, however, recall that we have another tool which casts parameter estimation as a optimisation problem targeting a well motivated objective from Equation 2.18:

$$(\pi^*_{1:K}, \mu^*_{1:K}, \sigma^*_{1:K}) = \underset{\pi_{1:K}, \mu_{1:K}, \sigma_{1:K}}{\arg\max}\ p(x^{(1:N)}; \pi_{1:K}, \mu_{1:K}, \sigma_{1:K}) \tag{2.24}$$

Unlike before, we do not have access to the likelihood term when there are additional latent variables, $z^{(1:N)}$. However, in some given density, $p(x, z; \theta)$, the marginal likelihood, $p(x; \theta)$ can always be obtained by marginalizing over the latent variables.

$$p(x; \theta) = \int p(x, z; \theta) dz = \mathbb{E}_{p(z)}[p(x|z; \theta)] \tag{2.25}$$

In our GMM model, the likelihood term can then be written as:

$$\begin{aligned} p(x^{(1:N)}; \pi_{1:K}, \mu_{1:K}, \sigma_{1:K}) &= \sum_z p(x^{(1:N)}, z^{(1:N)}; \pi_{1:K}, \mu_{1:K}, \sigma_{1:K}) \\ &= \sum_z p(z^{(1:N)}; \theta) p(x^{(1:N)}|z^{(1:N)}; \theta) \\ &= \prod_i \sum_z p(z_k; \theta_k) \mathcal{N}(x^{(i)}; \theta_k) \\ &= \prod_i \sum_{j=1}^{K} \pi_j \text{Normal}(x^{(i)}; \mu_j, \sigma_j) \end{aligned} \tag{2.26}$$

where we replace the integral with a summation since our z's are discrete variables. However, the optimisation problem over the log-evidence is not analytically tractable and difficult for numerical methods due to its non-convexity.

We now introduce the Expectation-Maximisation (EM) algorithm for parameter estimation in latent variable models. Then, we will motivate its objective from a maximum likelihood perspective. EM is an iterative algorithm that optimises the log-joint.

When the parameters are fixed, we can compute the posterior in closed form as well by summing over the latents. For a given set of fixed parameters, $\hat{\theta} =$

$\{\hat{\pi}_{1:K}, \hat{\mu}_{1:K}, \hat{\sigma}_{1:K}\}$, we have:

$$p(z^{(1:N)}|x^{(1:N)};\hat{\theta}) = \frac{p(z^{(1:N)}, x^{(1:N)};\hat{\theta})}{p(x^{(1:N)};\hat{\theta})} = \prod_i \frac{\text{Categorical}(z^{(i)};\hat{\pi})\text{Normal}(x^{(i)};\hat{\mu}_{z^{(i)}}, \hat{\sigma}_{z^{(i)}})}{\sum_{j=1}^K \hat{\pi}_j \text{Normal}(x^{(i)};\hat{\mu}_j, \hat{\sigma}_j)} \tag{2.27}$$

Using this posterior, we iteratively refine our parameter estimate using coordinate ascent on the log-joint objective. The E-step refers to obtaining the posterior required for computing the expectation to be maximised in the M-step. We alternating do the following until convergences.

First, we compute the posteriors for a fixed parameter set:

$$q(z^{(i)}) \triangleq p(z^{(i)}|x^{(i)};\hat{\theta}) : x^{(i)} \in \mathcal{D} \tag{2.28}$$

Once we have the posteriors, we maximise the following objective:

$$\hat{\theta} = \arg\max_\theta \mathbb{E}_{q(z^{(1:N)})}[\log p(z^{(1:N)}, x^{(1:N)})] \tag{2.29}$$

The above maximisation problem can be solved by gradient optimisation. notice it is more tractable than the previous one due to its form:

$$\arg\max_\theta \mathbb{E}_{q(z^{(1:N)})}[\log p(z^{(1:N)}, x^{(1:N)})] = \sum_i^N \sum_j^K q(z^{(i)})[\log \pi_j + \log \text{Normal}(x^{(i)}|\mu_j, \sigma_j)] \tag{2.30}$$

We repeat this until convergence by initialising random parameters, θ, and setting the new parameter set to those found during the M-step to be used for the fixed E-step. Finally, we note that this requires the posterior to be exactly computable. Finally, we include a short motivation and connection back to our original maximum likelihood objective. This will set up what we subsequently refer to as model learning in latent variable models. It turns out that our original MLE objective is equivalent to the following:

$$\arg\max_\theta \log p(x;\theta) = \arg\max_\theta \mathbb{E}_{p(z|x;\theta)}[\log p(x, z;\theta)] \tag{2.31}$$

$$= \arg\max_\theta \mathbb{E}_{p(z|x;\theta)}[\log p(x;\theta) + \cancelto{c}{\log p(z|x;\theta)}] \tag{2.32}$$

As we can see, when we can evaluate this new objective by characterising the posterior over z, we obtain a tractable way to update our parameters towards its MLE.

Although we do not use the exact posterior, but one with fixed parameters, we can show that the optimisation problem is strictly improving in parameter space with respect to the objective function. Similarly, we will later show how this new objective holds for a general *variational* posterior, $q(z)$, i.e. that maximising the log-joint is equivalent to maximising the log-evidence.

2.1.5 Variational Inference and Importance Sampling

In the GMM model, the posterior was exactly computable in closed form given the model setup. However, this is almost never the case in other higher dimensional models. The question then becomes what do we do. In these cases, we can again rely on some MCMC method to characterise the posterior, or we can use a more computationally efficient way.

Now, we introduce variational inference and importance sampling.

Monte Carlo Estimation

In the previous sections, we have assumed that the expectations or integrals we are after can be computed exactly, either analytically or using a summation. When this fails, we have generally referred to characterising the posterior using samples from MCMC instead, but what does this actually mean? It turns out that we can use a sample based estimation of an expectation. Suppose we are interested in a quantity for a real-valued random variable, Z:

$$I(f(z)) = \mathbb{E}_{p(z)}[f(z)] = \int p(z)f(z)dz \tag{2.33}$$

If we cannot compute this exactly, we can still produce an estimate of this quantity using K IID samples from $p(z)$:

$$\mathbb{E}_{p(z)}[f(z)] \approx I_K(f(z)) = \frac{1}{K}\sum_{i=1}^{K} f(z^i), \quad \text{with } z^{1:K} \sim p(\cdot) \tag{2.34}$$

Monte Carlo estimation has several nice properties. First, I_K is an unbiased estimator of I:

$$\begin{aligned}\mathbb{E}_{p(z)}[I_K] &= \mathbb{E}_{p(z)}[\frac{1}{K}\sum_{i=1}^{K} f(z^i)] \\ &= \frac{1}{K}\mathbb{E}_{p(z)}[f(z^1) + f(z^2) + ... + f(z^K)] \\ &= \frac{1}{K}(IK) = I \end{aligned} \tag{2.35}$$

Second, its variance is inversely proportional the the variance of $f(z)$ and decreases as the number of samples, K, increases. Given $\text{Var}[f(z)] = \sigma_f^2$:

$$\begin{aligned}\text{Var}[I_K] &= \text{Var}[\frac{1}{K}\sum_{i=1}^{K} f(z^i)] \\ &= \text{Var}[\frac{1}{K}(f(z^1) + f(z^2) + ... + f(z^K))] \\ &= \frac{1}{K^2}\text{Var}[(f(z^1) + f(z^2) + ... + f(z^K))] \\ &= \frac{1}{K^2}(K\sigma_f^2) = \frac{\sigma_f^2}{K} \end{aligned} \tag{2.36}$$

We can also compute a MC estimate of the variance using our samples, using the sample mean we obtained:

$$\hat{\sigma}_{I_K}^2 = \frac{1}{K}\sum_{i=1}^{K}[f(z^i) - I_K]^2, \quad \text{with } z^{1:K} \sim p(\cdot) \tag{2.37}$$

Therefore, once we have samples from the target distribution, we can compute any expectations over the distribution as a MC estimate. As we previously mentioned briefly, such a sampling process can be achieved using MCMC (and hence the term 'Monte Carlo' with MCMC). However, this has a number of drawbacks including its expensive nature, inability to sample in parallel, and under-exploration of important areas of the space.

Importance Sampling

When sampling is too difficult, expensive, or impossible, we instead turn to importance sampling (IS). Again, starting with the expectation we wish to compute:

$$I(f(z)) = \mathbb{E}_{p(z)}[f(z)] = \int p(z)f(z)dz \tag{2.38}$$

The importance sampled estimator, $I^{\text{IS}}(f(z))$, is computed by sampling from a proposal distribution, $q(z)$, such that the support of $q(z)$ is at least the support of $p(x)$. Then, we can derive an equivalent expression to I using an expectation under $q(z)$ instead:

$$\mathbb{E}_{p(z)}[f(z)] = \int p(z)f(z)dz = \int \frac{q(z)}{q(z)}p(z)f(z)dz = \mathbb{E}_{q(z)}[\frac{p(z)}{q(z)}f(z)] \tag{2.39}$$

Using Monte Carlo estimation to sample from $q(z)$ instead, we can derive the MC based estimate of $I^{\text{IS}}(f(z))$ as:

$$I_K^{\text{IS}}(f(z)) = \frac{1}{K}\sum_{i=1}^{K}\frac{p(z^i)f(z^i)}{q(z^i)}, \quad \text{with } z^{1:K} \sim q(\cdot) \tag{2.40}$$

This estimator, I_K^{IS}, is an unbiased estimate of I, i.e. $\mathbb{E}_{q(z)}[I_K^{\text{IS}}] = I$. However, its variance does not behave exactly as the original MC estimator, I_K. The variance of this estimator is straightforward and given as:

$$\text{Var}[I_K^{\text{IS}}] = \frac{1}{K}\sum_{i=1}^{K}[\frac{p(z^i)f(z^i)}{q(z^i)} - I_K^{\text{IS}}, \quad \text{with } z^{1:K} \sim q(\cdot) \tag{2.41}$$

Rearranging terms from the above, we can see that the variance of the estimator depends on our choice of q given fixed p, f. What we could like is to choose a $q(z) \propto p(z)|f(z)|$, that is to ideally pick densities where q is large wherever the integrand or likelihood is large, and we use the absolute value, $|f(z)|$, as f can be negative. This gives us an intuition for how to choose our proposals q to provide lower variance estimates as well as intuition for one final reason to use IS even when we can sample inexpensively from p. In many cases, even when we can sample from p to compute the normal MC estimate, $I_K = E_{z_{q:K}}[f(z_i)]$, the "important" regions of $f(z)$ may have low probability under the original density $p(z)$. In these cases, our estimate, I_K, although unbiased, will rarely see these outcomes, unless we can specifically target them using a well designed $q(z)$.

Self-Normalised Importance Sampling

Finally, we have been working under the assumption that the original density, $p(\cdot)$, and the proposal, $q(\cdot)$ are computable point-wise. However, often times, they are

only computable up to a normalising constant, Z. Consider the specific case of latent-variable models that we have discussed, $p(z, x)$. We are often interested in evaluating some function under the posterior $\mathbb{E}_{p(z|x)}[f(z)]$. However, even with a proposal to sample $z \sim q(\cdot)$, we often cannot evaluate the density, $p(z|x)$ for samples of z. In these cases, we resort to self-normalised importance sampling (SNIS):

$$\begin{aligned}
\mathbb{E}_{q(z)}[\frac{p(z)}{q(z)}f(z)] &= \frac{1}{Z}\mathbb{E}_{q(z)}[\frac{p(z)Z}{q(z)}f(z)] \\
&= \frac{1}{Z}\mathbb{E}_{q(z)}[\frac{\tilde{p}(z)}{q(z)}f(z)] \\
&= \frac{1}{Z}\mathbb{E}_{q(z)}[\tilde{w}(z)f(z)] \\
&= \frac{\mathbb{E}_{q(z)}[\tilde{w}(z)f(z)]}{\mathbb{E}_{q(z)}[\tilde{w}(z)]} \\
&= \frac{\frac{1}{K}\sum_{i=1}^{K}\tilde{w}(z^i)f(z^i)}{\frac{1}{K}\sum_{j=1}^{K}\tilde{w}(z^j)} \\
&= \sum_{i=1}^{K}\frac{\tilde{w}(z^i)}{\sum_{j=1}^{K}\tilde{w}(z^j)}f(z^i) \quad \text{with } z^{1:K} \sim q(\cdot),
\end{aligned} \tag{2.42}$$

where $p(z) = \frac{\tilde{p}(z)}{Z}$, $\tilde{w}(z^k) = \frac{\tilde{p}(z^k)}{q(z^k)}$. From the above derivations, we can also see the normalisation constant is estimated as:

$$\hat{Z} = \mathbb{E}_{q(z)}[\tilde{w}(z)] \approx \frac{1}{K}\sum_{j=1}^{K}\tilde{w}(z^j), \tag{2.43}$$

and the target distribution, $p(z)$ is estimated as the sum of re-weighted Dirac masses:

$$p(z) \approx \sum_{i=1}^{K}\frac{\tilde{w}(z^i)}{\sum_{j=1}^{K}\tilde{w}(z^j)}\delta_{z^j} \tag{2.44}$$

Variational Inference

Instead of performing exact inference, we can also approach posterior inference using an approximate inference technique. One of the leading approaches to approximate inference is called variational inference (VI). This class of methods casts posterior inference as an optimisation problem. Given our target density, $p(z|x)$, and a family of distributions, $\mathcal{Q}$, we wish to find some density, $q(z) \in \mathcal{Q}$,

that minimises the KL-divergence to the true posterior, $p(z|x)$:

$$q^*(z) = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}(q(z) \parallel p(z|x)) \tag{2.45}$$

$$= \arg\min_{q \in \mathcal{Q}} \mathbb{E}_{q(z)}[\log(\frac{q(z)}{p(z|x)})] \tag{2.46}$$

In this way, the approximating posterior density is obtained by solving this optimisation problem. However, this objective, as written, presents a problem for us, since the quantity we want to approximate needs to be computed. Instead, our requirement to use variational inference is that we can compute the joint density, $p(z, x)$. If this is possible, optimising the variational objective above is equivalent to the following:

$$\begin{aligned} q^*(z) &= \arg\min_{q \in \mathcal{Q}} \mathbb{E}_{q(z)}[\log(\frac{q(z)}{p(z|x)})] \\ &= \arg\min_{q \in \mathcal{Q}} \mathbb{E}_{q(z)}[\log(\frac{q(z)}{p(z|x)})] \\ &= \arg\min_{q \in \mathcal{Q}} \mathbb{E}_{q(z)}[\log(\frac{q(z)p(x)}{p(z|x)p(x)})] \\ &= \arg\min_{q \in \mathcal{Q}} \mathbb{E}_{q(z)}[\log(\frac{q(z)}{p(z,x)})] + \cancelto{c}{\log(p(x))} \end{aligned} \tag{2.47}$$

We will later connect this to an objective used to train VAEs, but for now, it can be considered a tool to do approximate inference. We also note, if we believe our proposal is good, we can further use it for importance sampling to evaluate an integral [Golinski et al., 2019]. So far, we have introduced the preliminaries for posterior estimation and model learning. In the following sections, we will see how all of these techniques apply to sequential models.

2.2 Sequential Modelling and Decision Making

Suppose instead of a coin flip or a single static event, we have a sequence of observations correlated in time. This is what time-series models aim to capture, and especially latent variable sequential models. In this section, we will introduce sequential models, starting with linear Markov models and then the more general state space model. Finally, we introduce reinforcement learning, which is the

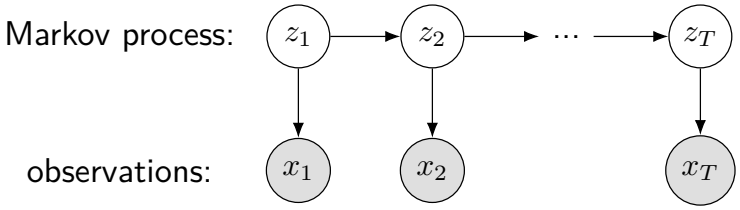


Figure 2.2: Hidden Markov model

framework for sequential decision making. We are interested in these models, as they are the bridge towards intelligent agents that can act in the world in a temporally consistent way.

2.2.1 Hidden Markov Models and Linear Dynamical Systems

We have so far introduced simple models of systems in a single time-step that allow us to reason about an unchanging representation of the world. In sequential models, we instead attempt to reason about a sequence of these static models at a given point in time. However, for these to be temporally consistent, the model at a given time-step must depend on the previous time-steps, i.e., for each time-step, t, we are interested in joint density:

$$p(x_{1:T}) = p(x_1) \prod_{t=2}^{T} p(x_t|x_{1:t-1}) \tag{2.48}$$

However, this becomes unbounded as t grows, so often, we assume a Markovian dependence:

$$p(x_t|x_{1:t-1}) = p(x_t|x_{t-n:t-1}) \tag{2.49}$$

where x_t is independent of the initial sequence, $x_{1:t-n-1}$, given the previous fixed window, $x_{t-n:t-1}$. n denotes a n^{th} order Markov chain. In practice, we often consider the first order Markov assumption, $p(x_t|x_{1:t-1}) = p(x_t|x_{t-1})$. Using these models, we can fit them to data as before via maximum likelihood estimation.

However, since we are operating in the sequential domain, in addition to model learning, we are now able to perform a variety of other interesting tasks once we have learned a model. We can for example, do forecasting, i.e. to compute the

probability of some future state given the observed states, $p(x_{t+1}|x_{1:t})$. We can also score the observed sequence itself, by computing its joint likelihood. Furthermore, with the model fixed in a discrete observation space, we can also compute the stationary distribution, $p(x_t) = p(x_{t+1}) = \sum_x p(x_{t+1}|x_t)p(x_t)$. This stationary distribution describes the marginal distribution over states at any given time-step. In the later sections, we will describe other interesting tasks that can be performed, such as filtering and smoothing, which require a latent variable model.

Hidden Markov Models

As before, we use latent variables to obtain a more accurate model of the world. In the sequential domain, we can model the sequence of observations and latent variables as:

$$p(x_{1:T}, z_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t|z_{1:t-1}, x_{1:t-1}) \prod_{t=1}^{T} p(x_t|z_{1:t}, x_{1:t-1}) \tag{2.50}$$

Relaxing the factorisation to depend less on previous time-steps allows for more tractable inference and model learning. When the current model slice only depends on the previous time-step, we can again take advantage of the Markov assumption. A Markovian state space model is given by the following:

$$p(x_{1:T}, z_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t|z_{t-1}) \prod_{t=1}^{T} p(x_t|z_t) \tag{2.51}$$

Because these models are factorised in time, we can perform many different kinds of state estimation operations. In state space models, we are generally interested in the following quantities:

- Smoothing: $p(z_t|x_{1:T})$ for $t < T$
- Filtering: $p(z_t|x_{1:t})$
- Prediction: $p(x_t|x_{1:t-1})$
- Likelihood: $p(x_{1:T})$

We start with a discrete hidden Markov model (HMM) to illustrate how these tasks can be computed in a simple state-space model with rigid assumptions. Our HMM is defined using M discrete states, $z_t \in 1, 2, ..., M$. All transition dynamics, $p(z_t|z_{t-1})$, are given by a $M \times M$ transition matrix, P, where the rows index the previous state value, z_{t-1}, and the column indexes the next state value, z_t. Each entry in the matrix then denotes the probability of transition from the row-value to the column-value, and rows in this matrix sum to 1. Finally, the emission densities, $p(x_t|z_t)$ are left as a design choice, depending on the data. The discrete HMM gives us a way to do tractable inference in a time-series model with an evolving discrete latent state.

We will first look at the smoothing problem. Given a sequence of observations, we would like to compute the posteriors given the full sequence at every time-step:

$$\begin{aligned} p(z_t|x_{1:T}) &= \frac{p(z_t, x_{1:T})}{p(x_{1:T})} \\ &= \frac{p(x_{1:T}|z_t)p(z_t)}{p(x_{1:T})} \\ &= \frac{p(x_{1:t}|z_t)p(z_t)p(x_{t+1:T}|z_t)}{p(x_{1:T})} \end{aligned} \tag{2.52}$$

If we look at the numerator of this posterior, we can break it up into two constituents, $\alpha(z_t) = p(x_{1:t}, z_t)$ and $\beta(z_t) = p(x_{t+1:T}|z_t)$, referred to as the *forward* and *backward* messages, respectively. These messages can both be computed recursively:

$$\begin{aligned} \alpha(z_t) &= p(x_{1:t}, z_t) \\ &= \sum_{z_{t-1}} p(x_{1:t}, z_t, z_{t-1}) \\ &= \sum_{z_{t-1}} p(x_{1:t-1}, z_{t-1})p(x_t, z_t|x_{1:t-1}, z_{t-1}) \\ &= \sum_{z_{t-1}} \alpha(z_{t-1})p(x_t, z_t|x_{1:t-1}, z_{t-1}) \\ &= \sum_{z_{t-1}} \alpha(z_{t-1})p(x_t, z_t|z_{t-1}), \end{aligned} \tag{2.53}$$

where the last line follows from the Markov property of HMMs. As shown computing $\alpha(\cdot)$ can be done recursively starting with $\alpha(z_1) = p(x_1|z_1)p(z_1)$:

$$\begin{aligned}
\alpha(z_2) &= \sum_{z_1} p(x_1|z_1)p(z_1)p(x_2, z_2|z_1) \\
\alpha(z_3) &= \sum_{z_2}\sum_{z_1} p(x_1|z_1)p(z_1)p(x_2, z_2|z_1)p(x_3, z_3|z_2) \\
\alpha(z_t) &= \sum_{z_{t-1}} ... \sum_{z_2}\sum_{z_1} p(x_1|z_1)p(z_1)p(x_2, z_2|z_1)p(x_3, z_3|z_2)...p(x_t, z_t|z_{t-1})
\end{aligned} \tag{2.54}$$

Similarly, the backwards message, $\beta(z_t)$, can also be computed recursively in the other direction:

$$\begin{aligned}
\beta(z_t) &= p(x_{t+1:T}|z_t) \\
&= \sum_{z_{t+1}} p(x_{t+1:T}, z_{t+1}|z_t) \\
&= \sum_{z_{t+1}} p(x_{t+1:T}|z_{t+1}, z_t)p(z_{t+1}|z_t) \\
&= \sum_{z_{t+1}} p(x_{t+2:T}|z_{t+1}, z_t)p(z_{t+1}|z_t)p(x_{t+1}|z_{t+1}, z_t) \\
&= \sum_{z_{t+1}} p(x_{t+2:T}|z_{t+1})p(z_{t+1}|z_t)p(x_{t+1}|z_{t+1}) \\
&= \sum_{z_{t+1}} \beta(z_{t+1})p(z_{t+1}, x_{t+1}|z_t)
\end{aligned} \tag{2.55}$$

where the second to last line follows from the Markov property again. Similarly as before, we can start with the final state and compute $\beta(z_{T-1})$ and the rest recursively:

$$\begin{aligned}
\beta(z_{T-1}) &= \sum_{z_T} \beta(z_T)p(x_T, z_T|z_{T-1}) \\
\beta(z_{T-2}) &= \sum_{z_{T-1}}\sum_{z_T} p(x_T, z_T|z_{T-1})p(z_{T-1}, x_{T-1}|z_{T-2}) \\
\beta(z_1) &= \sum_{z_2} ... \sum_{z_{T-1}}\sum_{z_T} p(x_T, z_T|z_{T-1})p(z_{T-1}, x_{T-1}|z_{T-2})...p(z_2, x_2|z_1)
\end{aligned} \tag{2.56}$$

where $\beta(z_T)$ is initialised to be uniform for the final time-step. Each smoothing posterior can then be trivially normalised. This normalisation normally requires the evidence term, but in the case of discrete posteriors, we can do so trivially, knowing the terms probabilities for the discrete state sum to 1. This is referred to as the forward-backward algorithm and is a foundation for each of the tasks in time-series analysis.

We are able to perform the other aforementioned sequence related tasks as well, based on forward-backward message passing. For filtering, the posterior, $p(z_t|x_{1:t})$, can be computed by normalising the forward message for each hidden state, in the same way as the smoothing posterior described in the previous paragraph:

$$p(z_t|x_{1:t}) \propto p(x_{1:t}, z_t) \propto \alpha(z_t) \tag{2.57}$$

In fact, the normalising constant is exactly the likelihood of the sequence, which can can be computed by marginalising the forward-message over the hidden state:

$$p(x_{1:t}) = \sum_{z_t} p(x_{1:t}, z_t) = \sum_{z_t} \alpha(z_t) \tag{2.58}$$

Finally, the predictive distribution can be calculated using the filtering distribution as well. It simply runs the forward model after computing the final time-step posterior:

$$p(x_{t+1}|x_{1:t}) = \sum_{z_{t+1}} \sum_{z_t} p(x_{t+1}|z_{t+1}) p(z_{t+1}|z_t) p(z_t|x_{1:t}) \tag{2.59}$$

Aside from these tasks, we can also combine the smoothing posteriors, $p(z_{1:T}|x_{1:T})$, for model learning using EM. Finally, we note that in the discrete HMM, the emission and transition functions can be generalised to be time dependent, i.e. we do not need to use the same function each time.

Linear Dynamical Systems

In the previous section, we introduced the HMM, which specified arbitrary emission probabilities conditioned on an evolving discrete state space. We used the forward-backward algorithm to obtain posteriors, where it was possible to perform exact inference due to the summation across discrete states and trivial normalisation of the posterior. However, we are often interested in modelling a continuous-valued state as well. In these cases, we will start by considering a case with rigid assumptions, referred to as a linear dynamical system (LDS).

A LDS has the same graphical structure as a HMM, but the discrete latent space is replaced with continuous valued spaces. Recall that we wanted to perform inference using the $\alpha(z_t)\beta(z_t)$. This was possible when, after doing the forward

backward passes, the messages could be normalised. In the case of continuous distributions, performing the normalisation is intractable due to the integral needed to compute the evidence. Therefore, we will place additional assumptions on the graphical model in order to analytically compute the integrals. When we assume linear transition and emission dynamics are all Gaussian densities, we are able to once again compute all the aforementioned quantities of interest.

A LDS has the same factorisation as the discrete HMM given in Equation 2.51. The transition and emission densities can be written as linear transformations on the conditioned state variables:

$$\begin{aligned} p(z_1) &= \mathcal{N}(x_1; \mu_1, \Sigma_1) \\ p(z_t|z_{t-1}) &= \mathcal{N}(z_t; Az_{t-1}, B) \\ p(x_t|z_t) &= \mathcal{N}(x_t; Cz_t, D) \end{aligned} \tag{2.60}$$

This is typically written as the system of equations:

$$\begin{aligned} \epsilon_t, \eta_t &\sim \mathcal{N}(\,\cdot\,; 0, I) \\ z_t &= Az_{t-1} + B\epsilon_t \\ x_t &= Cz_t + D\eta_t \end{aligned} \tag{2.61}$$

where ϵ_t, η_t are noise sample from the standard Gaussian. We can use a recursive Bayesian updating identity to estimate the posteriors at each state time-step:

$$\begin{aligned} p(z_t|x_{1:t}) &= \frac{p(x_t|z_t, x_{1:t-1})p(z_t|x_{1:t-1})}{p(x_t|x_{1:t-1})} \\ &\propto p(x_t|z_t)p(z_t|x_{1:t-1}) \end{aligned} \tag{2.62}$$

$$p(z_t|x_{1:t-1}) = \int p(z_t|z_{t-1})p(z_{t-1}|x_{1:t-1})dz_{t-1} \tag{2.63}$$

Since all the densities involved are Gaussian, we can compute all the necessary quantities of interest. Starting with the first time-step:

$$p(z_1|x_1) = \frac{\mathcal{N}(x_1; \mu_1, \Sigma_1)\mathcal{N}(x_1; Cz_1, D)}{\int \mathcal{N}(x_1; \mu_1, \Sigma_1)\mathcal{N}(x_1; Cz_1, D)dz_1}. \tag{2.64}$$

Then, if we define the posterior at $t-1$ to be $p(z_{t-1}|x_{1:t-1}) = \mathcal{N}(z_{t-1}; \hat{\mu}_{t-1}, \hat{\Sigma}_{t-1})$, we can compute the recursion by applying Equation 2.63 to Equation 2.62 for Gaussian densities:

$$p(z_t|x_{1:t-1}) = \mathcal{N}(z_t; A\hat{\mu}_{t-1}, A\hat{\Sigma}_{t-1}A^\top + B) \qquad \text{(prior)} \tag{2.65}$$

$$H_t = A\hat{\Sigma}_{t-1}A^\top + B$$

$$p(x_t|x_{1:t-1}) = \mathcal{N}(x_t; CA\hat{\mu}_{t-1}, CH_tC^\top + D) \qquad \text{(evidence)} \tag{2.66}$$

$$G_t = CH_tC^\top + D$$

$$p(z_t|x_{1:t}) = \mathcal{N}(z_t; A\hat{\mu}_{t-1} + H_tC^\top G_t^{-1}(x_t - CA\hat{\mu}_{t-1}), H_t - H_tC^\top G_t^{-1}CH_t) \tag{2.67}$$

All the densities of interest in this model can be computed analytically, since Gaussians are closed under multiplication. The above quantities follow from the conditioning rules of multivariate Gaussians. For more details on the theoretical derivations, we refer the reader to Meinhold and Singpurwalla [1983]. Parameter estimation in LDS can again be done using the EM algorithm. When we compute the objective using the expectation with fixed parameters, we can take the partial derivative of the objective function with respect to A, B, C, D in order to optimise. This gives us an update rule for improving our model, which is analytically tractable at each step.

2.2.2 Sequential Monte Carlo

So far, we have covered two sequential models with rigid assumptions. These are necessary for performing inference in the model, which is subsequently necessary for model learning. In this section, we cover how inference can be done in models with arbitrary densities and non-linear transition and emission transformations. In these models, we resort to sequential importance sampling. Given our usual SSM:

$$p(x_{1:T}, z_{1:T}) = p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1})\prod_{t=1}^{T} p(x_t|z_t), \tag{2.68}$$

we will now define a proposal distribution as follows:

$$q(z_{1:T}) = q(z_1|x_1)\prod_{i=2}^{T} q(z_i|z_{i-1}, x_i) \tag{2.69}$$

Algorithm 1 Sequential Monte Carlo

Initialise model parameters, θ, proposal parameters, ϕ
Initialise particle set, $z_1^{1:K} \sim q_\phi(\cdot|x_1)$
Initialise weights, $w_{\theta,\phi}^k(x_1, z_1^k) = \frac{p_\theta(z_1^k, x_1)}{q_\phi(z_1^k|x_1)}$
Compute likelihood, $\log \hat{p}(x_1) = \log \frac{1}{K} \sum_{i=1}^K w_{\theta,\phi}^i(x_1, z_1^i)$
for $2 \leq t \leq T$ **do**
  Normalise weights, $\tilde{w}_{\theta,\phi}^k(x_{t-1}, z_{t-1}^k) = \frac{w_{\theta,\phi}^k(x_{t-1}, z_{t-1}^k)}{\sum_{l=1}^K w_{\theta,\phi}^l(x_{t-1}, z_{t-1}^l)}$
  Resample posterior, $\tilde{z}_{t-1}^{1:K} \sim \hat{p}(z_{t-1}|x_{1:t-1}) \triangleq \sum_{k=1}^K \tilde{w}_{\theta,\phi}^k(x_{t-1}, z_{t-1}^k)\delta_{z_{t-1}^k}$
  Sample particle set, $\tilde{z}_t^{1:K} \sim q_\phi(\cdot|\tilde{z}_{t-1}^{1:K}, x_t)$
  Compute weights, $w_{\theta,\phi}^k(x_t, z_t^k) = \frac{p_\theta(z_t^k, x_t)}{q_\phi(z_t^k|\tilde{z}_{t-1}^{1:K}, x_t)}$
  Accumulate likelihood, $\log \hat{p}(x_{1:t}) = \log \hat{p}(x_{1:t-1}) + \log \frac{1}{K} \sum_{i=1}^K w_{\theta,\phi}^i(x_t, z_t^i)$
end for
Return log-evidence approximation, $\log \hat{p}(x_{1:T})$
  filtering posteriors, $\{\sum_{k=1}^K \tilde{w}_{\theta,\phi}^k(x_t, z_t^k)\delta_{z_t^k}\}_{t=1}^T$

A particle filter is a way to estimate the filtering posteriors over the current latent state given the current and past observations as well as a way to estimate the marginals over the observations by using a particle set that approximates our current empirical estimate of the posterior at each time-step. Here, we show how we can use *sequential* importance sampling (SIS) to estimate the filtering posteriors. As before, given our factorisation of the proposal distribution, we obtain a way to recursively estimate the state distribution and marginal likelihoods at each time-step using samples. We want to approximate the posteriors for each time-step, t, as in the updates given in Equations 2.62,2.63:

$$p(z_t|x_{1:t}) \propto p(x_t|z_t)p(z_t|x_{1:t-1}), \tag{2.70}$$

where the recursion is given by the posterior at the first time-step, followed by the integral given as:

$$\begin{aligned} p(z_1|x_1) &= \frac{p(z_1, x_1)}{p(x_1)} \\ p(z_t|x_{1:t-1}) &= \int p(z_t|z_{t-1})p(z_{t-1}|x_{1:t-1})dz_{t-1} \end{aligned} \tag{2.71}$$

We are additionally trying to estimate the marginal densities:

$$p(x_{1:T}) = \prod_{t=1}^T p(x_t|x_{1:t-1}), \tag{2.72}$$

where the same recursion from above is used to compute the intermediate quantities:

$$p(x_t|x_{1:t-1}) = \int p(x_t|z_t)p(z_t|x_{1:t-1})dz_t \tag{2.73}$$

Now, considering the factorisation of our proposal distribution, we can actually compute these quantities of interest by importance sampling. Using Equation 2.44, we can obtain the weighted particle set that corresponds to the filtering posterior at each time-step. For example, at the first time-step, we sample particles from $q(z_1|x_1)$, which is the proposal distribution for $p(z_1|x_1)$. However, since we can only compute $p(z_1, x_1)$, we rely on SNIS to obtain the particle set, $\hat{p}(z_1|x_1) \approx \sum_{k=1}^{K} \tilde{w}(x_1, z_1^k)\delta_{z_1^k}$, where δ_z denotes a Dirac-delta mass on z, and the weights, $\tilde{w}$, are computed as:

$$\begin{aligned} z_1^{1:K} &\sim q(\cdot|x_1) \\ w(x_1, z_1^k) &:= \frac{p(z_1^k, x_1)}{q(z_1^k|x_1)} \quad \text{(unnormalised weights)} \\ \tilde{w}(x_1, z_1^k) &= \frac{w(x_1, z_1^k)}{\sum_i w(x_1, z_1^i)} \qquad (2.74) \\ \hat{p}(z_1|x_1) &\approx \sum_{k=1}^{K} \tilde{w}(x_1, z_1^k)\delta_{z_1^k} \qquad (2.75) \end{aligned}$$

Next, we sample the subsequent proposal using the particles we obtain form the posterior estimate that we have, $z_t^k \sim q(\cdot|z_{t-1}^i, x_t)$. This quantity, i.e. the newly sampled particles, z_t^k correspond to:

$$\int q(z_t|z_{t-1}, x_t)p(z_{t-1}|x_{1:t-1})dz_{t-1} \approx \int q(z_t|z_{t-1}^k, x_t)\tilde{w}(x_{t-1}, z_{t-1}^k)dz_{t-1}^{1:K} \tag{2.76}$$

Finally, as we are using SNIS throughout, we can also obtain the log-evidence terms using the normalisation computed from our weights and given by Equation 2.43:

$$\hat{p}(x_t) \approx \frac{1}{K}\sum_{i=1}^{K} \tilde{w}(x_t, z_t^i) \tag{2.77}$$

This is the core approach to SIS, and can be used to obtain the full trajectory of likelihoods and posteriors. However, naively doing this eventually degrades the particle set, and the state estimation deteriorates exponentially. For this reason,

we instead resample the particles before continuing, and reweigh the empirical posterior according to the re-sampling method. The simplest illustrative method is multinomial resampling, which works by sampling K times from a categorical distribution parametrised by the original self-normalised weights. The new weights would be recomputed to be uniform, and often results in the smallest weighted particles being thrown away.

When SIS is used with re-sampling, we obtain the class of methods known as sequential Monte Carlo (SMC). The full algorithm for this is given in Algorithm 1, where we have written the model and proposal densities to be parametrised by θ, ϕ, respectively. We will later show how this can be used for model-learning in time-series generative models. For now, we have introduced the SMC method, which can be a general approach to performing all of the time-series tasks we are interested in. Following the development of these methods, improvements have been made to the re-sampling step as detailed in the algorithm description above. For a more detailed reading on SMC methods, we refer the readers to Doucet et al. [2001].

2.2.3 Markov Decision Processes and Reinforcement Learning

In this thesis, we consider one of the most important sequential modelling tasks, which is decision making under optimality. We first introduce the basic ideas behind reinforcement learning, and in later sections, describe how it ties back into inference and the probabilistic modelling ideas from earlier.

Markov Decision Process

Formally, the RL problem considers a task modelled as a finite-time Markov decision process (MDP) comprised of states $s \in S$, actions $a \in A$, initial state distribution $p(s_1)$, transition function $p(s_{t+1}|s_t, a_t)$, and the random reward $R : S \times A \to \mathbb{R}$, distributed as $R(s, a) \sim r(\cdot|s, a)$. We wish to find the conditional distribution $\pi(a_t|s_t)$, referred to as the policy, that maximises the expectation

over the cumulative reward under the system dynamics $p(s_{t+1}|s_t, a_t)$ and initial conditions $p(s_0)$ under some discount factor, γ:

$$J(\pi) = \mathop{\mathbb{E}}_{\pi,p,r}\left[\sum_t \gamma^t R(a_t, s_t)\right] \tag{2.78}$$

We define the value-function, $V(s)$, and the action value-function, $Q^\pi(s,a)$ (also referred to as the Q-function or Q-value). $Q^\pi(s,a)$ is defined as the discounted expectation of the sum of rewards ahead following some policy after taking some action, a, in some state, s (this action is not necessarily following the policy). The value-function is then the Q-function marginalised over all actions for a given state.

$$Q^\pi(s,a) := \mathop{\mathbb{E}}_{\pi,p,r}\left[\sum_t \gamma^t R(a_t, s_t) \quad | \quad a_0 = a,\ s_0 = s\right] \tag{2.79}$$

$$V^\pi(s) := \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a)\right] \tag{2.80}$$

Next, we define the random variable, $Z^\pi(s,a)$, whose possible values are the full support of the distribution over discounted sum of rewards ahead starting at some state-action pair, (s,a), and following the policy, transition function, and random environment interactions:

$$Z^\pi(s,a) \overset{D}{:=} \sum_t \gamma^t R(a_t, s_t) \quad | \quad a_0 = a,\ s_0 = s, s_{t+1} \sim p(\cdot|s_t, a_t), a_t \sim \pi(\cdot|s_t), \tag{2.81}$$

$$Z^\pi(s) \overset{D}{:=} Z^\pi(s,a) \quad | \quad a \sim \pi(\cdot|s) \tag{2.82}$$

where for two random variables, U, V, $U \overset{D}{:=} V$ denotes that U and V follow the same distribution. We write $Z(s,a)$ as distributed according to $P_Z(\cdot|s,a)$ for clarity. The goal of distributional RL is to somehow capture or estimate P_Z^π in contrast to standard expected RL, which aims to estimate Q^π. Finally, it should be clear that the relationship between Q and Z is that the Q-value is the mean of Z:

$$Q^\pi(s,a) := \mathbb{E}_{P_Z}[Z^\pi(s,a)] \tag{2.83}$$

Temporal Difference

Reinforcement learning presents a uniquely challenging task where decision making takes into account future rewards. In order to make learning more feasible, temporal difference (TD) errors are used for value-function optimisation. Consider some guess, $\hat{Q}$, for the true value-function, $Q^\pi(s,a)$. The TD loss moves this guess closer to the real function via a Bellman operator with guaranteed convergence properties.

TD-errors are derived as following. We use $F \in \mathcal{F}$ to be general, which is either some $Q \in \mathcal{Q}$, the space of all Q-functions, or some $Z \in \mathcal{Z}$, the space of all Z-distributions. Given a *symmetric* distance metric $d_\mathcal{F}$ between elements of $\mathcal{F}$ and an operator $\mathcal{T}^\pi F$ which maps some initial $F_0 : S \times A \to \mathcal{F}$ to another F' in the same space, i.e. $F_{k+1} = \mathcal{T}^\pi F_k$ then there exists a fixed point, which is F^π in the limit as $k \to \infty$, if and only if $\mathcal{T}^\pi F$ is a contraction using $d_\mathcal{F}$ in $\mathcal{F}$:

$$d_\mathcal{F}(\mathcal{T}^\pi F_1, \mathcal{T}^\pi F_2) \leq \gamma d_\mathcal{F}(F_1, F_2). \tag{2.84}$$

If this is the case, i.e. the operator is contractive, then any current guess, $\hat{F}$, can be moved towards the fixed point by repeatedly minimising the distance, $d_\mathcal{F}(\hat{F}, \mathcal{T}^\pi \hat{F})$.

Q-learning

In the case of Q-functions, we define the following Bellman operator:

$$(\mathcal{T}^\pi Q)(s,a) = R(s,a) + \mathop{\mathbb{E}}_{s' \sim p,\ a' \sim \pi} \Big[\gamma Q(s', a') \Big]. \tag{2.85}$$

It can be shown that this operator is contractive under the supremum norm, L_∞:

$$||\mathcal{T}^\pi Q_1 - \mathcal{T}^\pi Q_2||_\infty \leq \gamma ||Q_1 - Q_2||_\infty \tag{2.86}$$

As a result, the TD-error in Q-learning is defined as the following:

$$\mathcal{L}^Q(s,a) = \mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a),\ a' \sim \pi} \left[\left(R(s,a) + \gamma Q(s', a') - Q(s,a) \right)^2 \right] \tag{2.87}$$

Policy Improvement

With a method to improve the value-function, we can turn to policy learning. Given that the RL task is to maximise the sum of rewards, we can simply target the Q-function as the policy objective:

$$\pi'(a|s) = \arg\max_a Q^\pi(s, a) \tag{2.88}$$

In the deterministic setting, this is the policy itself, and the *policy improvement theorem* gives us $V^{\pi'}(s) \geq V^\pi(s)$. We will later see how this can be generalised to the stochastic policy setting.

We defer additional derivations of classic results used in reinforcement learning to Section 2.4, but from the perspective of recasting the decision problem as an inference task. We refer the reader to Sutton and Barto [2018] for a more conventional approach to reinforcement learning and control theory.

Reinforcement learning algorithms are most often benchmarked via two metrics, total return and *sample efficiency*. Total return measures the goodness of a policy on the actual control task at test time, while sample efficiency measures how many time-steps under some some other policy was required in order to train our agent. The latter is equally important to reinforcement learning as total return. This is because we would like to minimise our interactions with the environment before being able to learn. We next cover current state-of-the-art approaches to the topics covered so far, and we return to the control problem using an inference perspective in Section 2.4.

2.3 Deep Generative Modelling and Amortised Inference

So far, we have introduced the early background in probabilistic modelling, inference, Monte Carlo estimation, and sequential modelling. But, we have not yet dealt with how these approaches are combined with deep learning for non-linear probabilistic modelling. In this section, we introduce a framework central

for the remainder of our thesis, the *variational auto-encoder* (VAE). The VAE encompasses both model learning in deep generative models (DGM) as well as amortised inference learning. We thus tie its ideas back to VI and deep non-linear time-series modelling, detailing its connection to HMM and SMC methods. Finally, we introduce background on a variety of modern techniques used to improve gradient estimation and learning in DGMs.

2.3.1 Deep Learning and Stochastic Gradient Descent

We first provide a brief background on deep learning. Generally, deep learning uses DNNs to approximate a mapping from inputs to outputs. Given a neural network, $y = f_\theta(x)$, and a dataset, $\mathcal{D} = \{x^{(n)}, y^{(n)}\}_{n=1}^N$, the goal of *supervised learning* is to minimise a loss function, $\mathcal{L}(x, y, \theta)$, relating the NN output to the true output:

$$\theta^* = \arg\min_\theta \mathcal{L}(y, f_\theta(x)) \tag{2.89}$$

This optimisation is typically done using stochastic gradient descent (SGD), where for a mini-batch update, a "stochastic" gradient is computed by sampling a subset, $M << N$, of $\mathcal{D}$, computing the gradient of the loss function with respect to parameters, θ, averaged over the subset, and applying this gradient to the parameter set in the direction of the gradient with some step-size, α:

$$\theta' = \theta - \alpha \nabla_\theta \mathbb{E}_{(x,y)^{(1:M)} \sim \mathcal{D}}[\mathcal{L}(y, f_\theta(x))] \tag{2.90}$$

Iteratively applying this update above with sampled mini-batches eventually converges to a local minima, and allows deep learning methods to approximate arbitrarily complex functions. This is because the function, f, is typically a series of linear combinations of network weights and non-linear "activations" (such as the sigmoid function), in an arbitrarily deep stack.

Under certain loss functions, this can also be viewed as maximising the conditional likelihood, $p_\theta(y|x)$:

$$\theta^* = \arg\max_\theta \mathbb{E}_\mathcal{D}[p_\theta(y|x)] \tag{2.91}$$

2.3.2 Variational Auto-Encoder

Recall from Sec 2.1.5 that we can cast inference as an optimisation problem by minimising the distance between a variational distribution, $q(\cdot)$, approximating the posterior and the true intractable posterior, $p(z|x)$. However, this has two major drawbacks: 1). the limited expressivity of the variational family, and 2). having to resolve the amortisation problem for each new observation, x. These issues are both tackled by the variational auto-encoder (VAE) framework, which couples a generative model with an *amortised inference* artefact.

In this section, we introduce this important framework, which helped popularise deep latent variables models while also introducing a training objective that can be used to both learn model parameters as well as a parametrised amortised inference network [Kingma and Welling, 2013a]. We can first define a deep latent variable model, $p_\theta(z, x)$, using our usual convention of a generative model factorised as $p(z)p_\theta(x|z)$, with latent variable, z, and observation, x. The prior, $p(z)$, is usually fixed for simple DGMs and the conditional *decoder*, $p_\theta(x|z)$, is a density which uses deep neural networks transforming the latent variable to an output. We can say that the observation density is parametrised by model parameters, θ.

As usual, we are interested in model learning, and in this case, maximising the log-joint density with respect to θ and under our dataset, $x^{(n)} \sim \mathcal{D}$:

$$\theta^* = \arg\max_\theta \mathbb{E}_{x\sim\mathcal{D}}[\mathbb{E}_{p(z|x)}[p_\theta(x, z)]] \tag{2.92}$$

One way to characterise the posterior here is to use VI and gradient optimisation to obtain an optimal $q(\cdot)$ for each observation x. However, this is very expensive and time-consuming per example. Instead, we now introduce the inference object or *encoder*, $q_\phi(z|x)$, which allows a single conditional object to be learned that *amortises* inference for any new datapoint, x. From Eq 2.47, we know that finding the optimal inference network is:

$$\phi^* = \arg\min_\phi \mathbb{E}_{q_\phi(z|x)}[\log(\frac{q_\phi(z|x)}{p_\theta(z, x)})] \tag{2.93}$$

As we mentioned, the VAE framework gives us an objective for both model learning and amortised inference. We so far have detailed the objective with respect to inference network parameters, ϕ, but it turns out this same objective can be used for learning θ as well. We can see this by starting with the log-evidence term we wish to maximise:

$$\begin{aligned}
\log[p(x)] &= \log[\int p_\theta(x,z)dz] \\
&= \log[\int q_\phi(z|x)\frac{p_\theta(x,z)}{q_\phi(z|x)}dz] \\
&= \log(\mathbb{E}_{q_\phi(z|x)}[\frac{p_\theta(x,z)}{q_\phi(z|x)}]) \\
&\geq \mathbb{E}_{q_\phi(z|x)}[\log(\frac{p_\theta(x,z)}{q_\phi(z|x)})] \quad \text{(by Jensen's)} \\
&= \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(x,z)
\end{aligned} \tag{2.94}$$

Eq 2.94 is referred to as the evidence lower bound (ELBO), and if we look back at the optimal inference network parameters, we will see that it is trying to minimise the negative ELBO, i.e. $\phi^* = \arg\min_\phi -\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(x,z)$. Furthermore, we can exactly specify the gap between the ELBO and the log-evidence, which is obtained by rearranging terms:

$$\begin{aligned}
\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(x,z) &= \mathbb{E}_{q_\phi(z|x)}[\log(\frac{p_\theta(x,z)}{q_\phi(z|x)})] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x,z)) - \log(q_\phi(z|x))] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(p(z|x)p(x)) - \log(q_\phi(z|x))] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(p(z|x)) + \log(p(x)) - \log(q_\phi(z|x))] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(p(z|x)) - \log(q_\phi(z|x))] + \log(p(x)) \\
&= -\text{KL}(q_\phi(z|x) \parallel p(z|x)) + \log(p(x))
\end{aligned} \tag{2.95}$$

As seen, the ELBO and the log-evidence differ by exactly the KL-divergence gap between the variational and true posterior. This means that, as the inference network improves, so too will the estimate of the ELBO for model learning.

Finally, we make a quick note of the auto-encoding interpretation of this loss, which can be seen as a reconstruction loss term and a regularisation term by

extracting $p_\theta(x|z)$ from the joint-density:

$$\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(x,z) = \mathbb{E}_{q_\phi(z|x)}[\log(p(x|z))] - \text{KL}(q_\phi(z|x) \parallel p(z)) \tag{2.96}$$

This can intuitively be thought of as learning a neural network to encode observations, x, to a low-dimensional embedding, z, such that the original observation can be *reconstructed* or decoded using, $p_\theta(x|z)$. In this interpretation, the second term of Eq 2.96 acts to regularise the encoder towards the prior, $p(z)$. This will be useful to us when we wish to over-emphasise this regularisation, for example, in order to encourage disentanglement of the latent space in representation learning [Higgins et al., 2016].

Reparametrisation and REINFORCE

So far, we have just derived the ELBO objective, and detailed its connections to the statistical learning concepts, but we have not yet specified how the optimisation problem is solved. It turns out that one important contribution the VAE framework provided was a way reparametrise the variational posterior distribution, in order to make the objective learnable via SGD methods. In order to learn $\{\phi, \theta\}$, we must take a gradient of the ELBO with respect to the parameters:

$$\nabla_{\phi,\theta}\mathbb{E}_{q_\phi(z|x)}[\frac{p_\theta(z,x)}{q_\phi(z|x)}] \tag{2.97}$$

However, we cannot simply push the gradient with respect to ϕ into the expectation, since the object under which we take the expectation, q, is dependent on ϕ. Kingma and Welling [2013b] provide a way around this, which is to specify a model with a reparametrisable density for the latent z. Then, we can estimate the gradient by sampling from an auxiliary distribution, and compute the gradient via:

$$\nabla_\phi \mathbb{E}_{q_\phi(z|x)}[w(x,z)] \triangleq \nabla_\phi \mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}(0,1)}[w(x, \mu_\phi + \epsilon\sigma_\phi)] \tag{2.98}$$

$$= \mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}(0,1)}[\nabla_\phi w(x, \mu_\phi + \epsilon\sigma_\phi)] \tag{2.99}$$

$$\text{where} \quad z \triangleq \mu_\phi + \epsilon\sigma_\phi \tag{2.100}$$

This allows us to take an equivalent expectation under a different sampling distribution, and thereby compute gradient the loss with respect to parameters, ϕ.

However, not all distributions are reparametrisable, most notably discrete distributions. In these cases, we resort to the REINFORCE trick, which is higher variance when used for computing the ELBO, but necessary when non-reparametrisable distributions are used:

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi(z|x)}[w(x,z)] &= \int \nabla_\phi [q_\phi(z|x) w(x,z)] dz \\ &= \int w(x,z) \nabla_\phi q_\phi(z|x) + q_\phi(z|x) \nabla_\phi w(z,x) dz \\ &= \int q_\phi(z|x) \nabla_\phi \log q_\phi(z|x) w(x,z) + q_\phi(z|x) \nabla_\phi w(z,x) dz \\ &= \mathbb{E}_{q_\phi(z|x)}[w(x,z) \nabla_\phi \log q_\phi(z|x) + \nabla_\phi w(z,x)] \end{aligned} \tag{2.101}$$

In subsequent sections, we will see the development of various techniques to combat this high-variance gradient estimate when non-reparametrisable distributions must be used. However, for now, these two gradient estimators provide a stable way to learn model parameters and inference parameters in VAE models.

Importance Weighted Auto-Encoder

After the VAE objective was developed, Burda et al. [2015] introduced an importance sampling modification to it. The importance weighted auto-encoder (IWAE) gives a tighter bound on the evidence log-likelihood by using multiple samples from the learned variational distribution:

$$\mathcal{L}^{\text{IWAE}}_{\theta,\phi,K}(x,z) = \mathbb{E}_{q(z^{1:K}|x)}\Big[\log\left(\frac{1}{k}\sum_{k=1}^{K}\frac{p_\theta(x,z^k)}{q_\phi(z^k|x)}\right)\Big] \tag{2.102}$$

Setting $K = 1$ recovers the original VAE objective in Eq 2.94. It can also be shown that as $K \to \infty$, the bound gets tighter and approaches $\log p(x)$:

$$\log p(x) \geq \mathcal{L}^{\text{IWAE}}_{\theta,\phi,K+1} \geq \mathcal{L}^{\text{IWAE}}_{\theta,\phi,K} \tag{2.103}$$

Finally, we can show that the gradient of the IWAE objective with respect to θ corresponds to the original model learning objective under self-normalised im-

portance sampling:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{p(z|x)}[\log p_\theta(x,z)] &= \mathbb{E}_{q(z|x)}\Big[\frac{p(z|x)}{q(z|x)}\nabla_\theta \log p_\theta(x,z)\Big] \\
&= \frac{1}{\hat{Z}}\mathbb{E}_{q(z|x)}\Big[\frac{p(z,x)}{q(z|x)}\nabla_\theta \log p_\theta(x,z)\Big] \\
&\approx \frac{\frac{1}{K}\sum_{k=1}^K w(z^k,x)\nabla_\theta \log p_\theta(x,z^k)}{\frac{1}{K}\sum_{l=1}^K w(z^l,x)} \\
&= \sum_{k=1}^K \frac{w(z^k,x)}{\sum_l w(z^l,x)}\nabla_\theta \log p_\theta(x,z^k), \qquad (2.104)
\end{aligned}$$

where $w(z,x) = \frac{p(z,x)}{q(z|x)}$ and $z^{1:K} \sim q(\cdot|x)$, and the expectation is approximated with Eq 2.42. Using this notation, we can also rearrange terms in the IWAE estimator to obtain the same gradient estimator:

$$\begin{aligned}
\nabla_\theta \mathcal{L}^{\text{IWAE}}_{\theta,\phi,K}(x,z) &= \mathbb{E}_{q(z^{1:K}|x)}\Big[\nabla_\theta \log\Big(\frac{1}{k}\sum_{k=1}^K \frac{p_\theta(x,z^k)}{q_\phi(z^k|x)}\Big)\Big] \\
&= \mathbb{E}_{q(z^{1:K}|x)}\Big[\frac{\nabla_\theta \frac{1}{k}\sum_{k=1}^K w(z^k,x)}{\frac{1}{k}\sum_{l=1}^K w(z^l,x)}\Big] \\
&= \mathbb{E}_{q(z^{1:K}|x)}\Big[\sum_{k=1}^K \frac{\nabla_\theta w(z^k,x)}{\sum_{l=1}^K w(z^l,x)}\Big] \\
&= \mathbb{E}_{q(z^{1:K}|x)}\Big[\sum_{k=1}^K \frac{w(z^k,x)}{w(z^k,x)}\frac{\nabla_\theta w(z^k,x)}{\sum_{l=1}^K w(z^l,x)}\Big] \\
&= \mathbb{E}_{q(z^{1:K}|x)}\Big[\sum_{k=1}^K \frac{w(z^k,x)}{\sum_{l=1}^K w(z^l,x)}\nabla_\theta \log w(z^k,x)\Big] \\
&\approx \sum_{k=1}^K \frac{w(z^k,x)}{\sum_l w(z^l,x)}\nabla_\theta \log p_\theta(x,z^k) \qquad (2.105)
\end{aligned}$$

We thereby connect model learning under VAE models to the original maximum likelihood objective. We can also see connections to the EM algorithm. Learning the amortised inference object is akin to characterising the posterior and using it for learning θ is akin to stepping model parameters in the M-step.

Limitations of Auto-Encoding Losses

Much recent work has also gone into examining the limitations of the ELBO and IWAE, and here, we briefly highlight some of these works and specifically how it relates to the modelling capacity of latent variable models. First, Alemi

et al. [2018] and Hoffman and Johnson [2016] suggest that the ELBO does not, in its basic form, support representation learning. They suggest that many VAE models only learn how to reconstruct the observations, agnostic to what the actual latent representation encodes. This is either due to unused latent variables in the decoder, or otherwise designing one that has capacity to map arbitrary input to the observation space.

The information-theoretic perspective is often used to combat these problems by reframing the objective in a way to better support representation learning. For example, Higgins et al. [2016] introduce the β-VAE, a modification to the ELBO that encourages disentanglement in the latent space:

$$\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(x,z) = \mathbb{E}_{q_\phi(z|x)}[\log(p(x|z))] - \beta \text{KL}(q_\phi(z|x) \parallel p(z)) \tag{2.106}$$

In the information theoretic perspective, the first term on the RHS is referred to the distortion and the second is the rate. By placing a multiplier on the rate term, Higgins et al. [2016] show that the latent space learns a factorised, interpretable representation, while incurring a tradeoff against the reconstruction loss. Burgess et al. [2018] further explore the β-VAE and improve the tradeoff.

We are also interested in partial observations, which takes additional modifications to the ELBO objectives. Kingma et al. [2014] extend the ELBO with an additional *semi-supervised* term using partially observed latent variables and Siddharth et al. [2016] decouple partially observed latent variables from the latent space and provide a semi-supervised training objective for improved representation learning. Approaching structured representation learning is also much easier when we have some training signal for the latent space. We will explore this idea in Chapter 4 for challenging sequential tasks.

Another known limitation of VAE objectives is the tradeoff between better inference networks and model learning. Le et al. [2018] and Rainforth et al. [2018] show that inference networks learn sub-optimally when using tighter bounds for model learning. The intuition here is that larger sampling bounds reduce the variance of the importance weighted estimate enough such that no learning signal is provided

to $q_\phi(z|x)$ when performing gradient optimisation. These papers modify the IWAE bound and decouple the objectives used for inference network learning and model learning. They are able to maintain the lower variance gradient estimates for the decoder while introducing alternative objectives for encoder training.

Discrete Latent Variables

In order to optimise ELBO-like objectives, continuous, reparameterisable densities are generally used to model latent variables so that we are able to compute unbiased gradient estimates. However, this limits the modelling capabilities of our generative models, since there are only a limited number of such densities. Most notably, discrete distributions are not reparametrisable. A main approach to overcome this is the use of *continuous relaxations* such as the Gumbel-Softmax [Jang et al., 2016] and the Concrete distributions [Maddison et al., 2016]. These densities approximate "discrete" outcomes, but does so by relaxing the hard discontinuities between discrete parameters. While this allows gradients to flow through discrete random variables, it also results in biased gradient estimators.

As previously mentioned, the main approach used when we have discrete latent variables is REINFORCE gradient estimation. However, we can see that this would produce a high-variance gradient estimator, for example, when we apply Equation 2.101 to the IWAE objective, $\mathcal{L}^{\text{IWAE}}_{\theta,\phi,K}(x,z)$:

$$\begin{aligned}\nabla_\phi \mathcal{L}^{\text{IWAE}}_{\theta,\phi,K}(x,z) &= \nabla_\phi \mathbb{E}_{q_\phi(z^{1:K}|x)}\left[\log\left(\frac{1}{k}\sum_{k=1}^{K}\frac{p_\theta(x,z^k)}{q_\phi(z^k|x)}\right)\right] \\ &= \mathbb{E}_{q_\phi(z^{1:K}|x)}\left[\nabla_\phi \log\left(\frac{1}{k}\sum_{k=1}^{K}\frac{p_\theta(x,z^k)}{q_\phi(z^k|x)}\right)\right. \\ &\quad \left. + \log\left(\frac{1}{k}\sum_{k=1}^{K}\frac{p_\theta(x,z^k)}{q_\phi(z^k|x)}\right)\nabla_\phi \log\left(q_\phi(z^{1:K}|x)\right)\right] \end{aligned} \tag{2.107}$$

The above estimator contains a high-variance second term in the expectation, which causes training to be unstable. When it is necessary to use this gradient estimator, one approach to combat the high-variance is to add a zero-expectation baseline, known as a *control variate*. Since the added term is zero-expectation, the bias of

the estimate will not change, but the variance will be reduced. Mnih and Rezende [2016] introduce the VIMCO control variate, given by:

$$
\begin{aligned}
f(z^{-l}, x) &= \exp\Big(\frac{1}{K-1}\sum_{k\neq l}\frac{p_\theta(x, z^k)}{q_\phi(z^k|x)}\Big) \\
g(z^{-l}, x) &= \log\Big(\frac{1}{K}\Big(f(z^{-l}, x) + \sum_{k\neq l}\frac{p_\theta(x, z^k)}{q_\phi(z^k|x)}\Big)\Big) \\
h(z^{1:K}, x) &= \sum_{l=1}^{K} g(z^{-l}, x)\nabla_\phi \log(q(z^l|x))
\end{aligned} \tag{2.108}
$$

When we add the term $h(z^{1:K}, x)$ to the expectation in Equation 2.107, we will reduce the variance of the REINFORCE estimator. Also, as the samples are drawn from the inference network, $z^{1:K} \sim q_\phi(\cdot|x)$, and $g(z^{-l}, x)$ is independent of l, $h(z^{1:K}, x)$ will have zero-expectation under $q_\phi(z^{1:K}|x)$. Similar approaches have been introduced in past years which build on the control variate framework. Tucker et al. [2017] improve this with a control variate known as REBAR, and Grathwohl et al. [2017] introduce a further extension called RELAX that uses a neural network to parametrise the control variate.

2.3.3 Implicit Deep Generative Models

In this section, we briefly review other classes of DGMs. We refer to those outside of the VAE framework as *implicit* generative models, as they are constructed to represent an underlying distribution, but do not explicitly model the joint density. For example, we can define a generative model simply as sampling from some initial density, $p_0(\cdot)$, and then transforming those using a deterministic neural network.

These implicit models do not use a lower-bound on the log-evidence in order to train, as they are not able to evaluate the likelihood. Of the successful implicit DGMs, the most popular include generative adversarial networks (GAN; Goodfellow et al. [2014]) and score-based models [Hyvärinen and Dayan, 2005, Song and Ermon, 2019].

GANs solve a minimax game which corresponds to minimising the Jensen-Shannon divergence between the data-generating distribution and the empirical data distribution. This is accomplished in practice with a discriminator jointly

trained with the generative model, which encourages the generative model to move closer to the true data. One advantage of this approach is the quality of generative samples and a discriminator network that is robust to adversarial examples. However, they are notoriously difficult to train due to the jointly networks [Salimans et al., 2016, Metz et al., 2016].

Score-based models, on the other hand, are trained by matching an output of a neural network to the *gradient* or score-function of the log-likelihood. This technique is called *score matching* and allows the Fisher divergence to be used as a loss for the generative model learning. The advantage of this is that the Fisher divergence is independent of the normalising constant, and thus expands the family of models that can be defined. However, they also face a drawback in their training difficulty and scalability [Song and Ermon, 2020].

We are generally interested in implicit models when we want a more flexible model, since they are not constrained by maximum likelihood training. The additional flexibility has often led to improved sample generation, but they lack the formalism of Bayesian statistics.

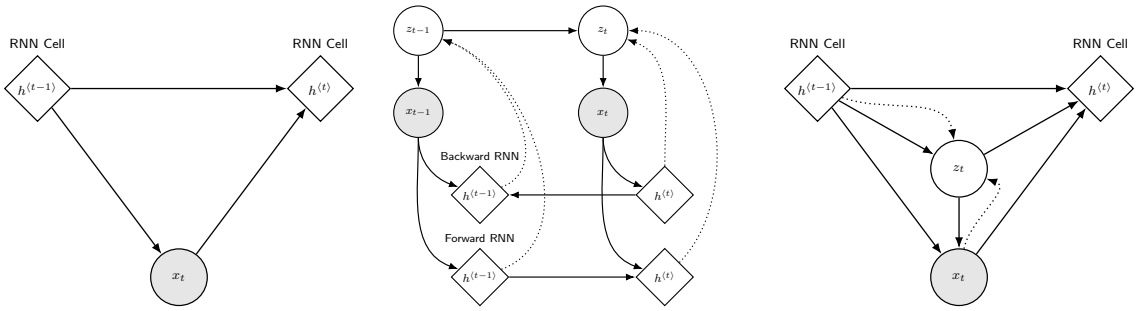


Figure 2.3: Graphical model representations of an RNN (left), deep Markov Model with BiRNN based inference network (middle), and variational RNN with shared RNN for model and inference networks (right).

2.3.4 Deep Sequential Modelling

In this section, we extend the ideas of deep generative modelling for deep sequential models. As before, the theme here is to take a gradient of an ELBO-like objective within a time-series model. We highlight both challenges and advances in this area.

Sequential Monte Carlo Gradient Estimator

Recall the log-evidence approximation given by the SMC estimator in Algorithm 1. It is easy to see that this is an IWAE-like bound using SIS throughout the roll-out. Thus, we can use SMC in deep neural network based sequential models and inference networks as in the VAE framework [Le et al., 2017b, Gu et al., 2015]. However, there is now an additional complication in that the "proposal" network used here contains a discrete re-sampling step. Recall the log-evidence given by an SMC method:

$$\mathcal{L}^{\text{SMC}}_{\phi,\theta}(x_{1:T}) = \mathbb{E}_{\bar{q}_{\phi,\theta}}\Big[\log \hat{p}(x_{1:T})\Big] \tag{2.109}$$

$$= \mathbb{E}_{\bar{q}_{\phi,\theta}(z^{1:K}_{1:T}, a^{1:K}_{1:T-1}|x_{1:T})}\Big[\sum_{t=1}^{T}\log\left(\frac{1}{K}\sum_{i=1}^{K}\frac{p_\theta(x_t, z^i_t|z^i_{t-1})}{q_\phi(z^i_t|x_t, z^i_{t-1})}\right)\Big], \tag{2.110}$$

where $a^{1:K}_{1:T-1}$ denotes the series of ancestral samples of z^k_t, i.e. before resampling. When written out with a resampling step, it is clear to see the true proposal distribution factorises as:

$$\bar{q}_\phi(z^k_t, a^k_{t-1}|x_t, z^k_{t-1}) \triangleq q_\phi(z^k_t|x_t, z^{a^k_{t-1}}_{t-1})\text{Discrete}(a^k_{t-1}|z^k_{t-1}; \tilde{w}^k_{t-1}), \tag{2.111}$$

for time-steps, $t > 1$, and $q_\phi(z^k_1|x^k_1)$ for $t = 1$. We also use the normalised weights, $\tilde{w}^k_t$, to parametrise the discrete resampling distribution, and they are given as:

$$w^k_t = \frac{p_\theta(x_t, z^k_t|z^k_{t-1})}{q_\phi(z^k_t|x_t, z^k_{t-1})} \quad \text{and} \quad \tilde{w}^k_t = \frac{w^k_t}{\sum_{l=1}^{K} w^l_t} \tag{2.112}$$

Now, we finally can write the gradient estimator of the SMC log-evidence with respect to model parameters, θ:

$$\nabla_\theta \mathcal{L}^{\text{SMC}}_{\phi,\theta}(x_{1:T}) = \nabla_\theta \mathbb{E}_{\bar{q}_{\phi,\theta}(z^{1:K}_{1:T}, a^{1:K}_{1:T-1}|x_{1:T})}\Big[\sum_{t=1}^{T}\log\left(\frac{1}{K}\sum_{i=1}^{K}\frac{p_\theta(x_t, z^i_t|z^i_{t-1})}{q_\phi(z^i_t|x_t, z^i_{t-1})}\right)\Big] \tag{2.113}$$

$$= \mathbb{E}_{\bar{q}_{\phi,\theta}}\Big[\nabla_\theta \sum_{t=1}^{T}\log\Big(\frac{1}{K}\sum_{i=1}^{K} w^k_t\Big)\Big] + \mathbb{E}_{\bar{q}_{\phi,\theta}}\Big[\log \hat{p}(x_{1:T})\nabla_\theta \log \bar{q}_{\phi,\theta}(\cdot)\Big] \tag{2.114}$$

Here, we note the difficulty with stochastic optimisation approaches to model-learning using SMC estimators. Since the second term in the above equation depends on model parameters, the gradient must be back-propagated through the

term. However, the model is used in Equation 2.111 specifically for the discrete resampling step, through the normalised weights in the ancestral sampling scheme.

Many works have presented ways to overcome this problem in SMC gradient estimators. Most notably, Le et al. [2017b], Maddison et al. [2017], and Naesseth et al. [2017] propose to ignore the problematic gradient, and show when conditions are met such that the biased gradient of only using the first term is not detrimental to learning. Another line of recent work, such as Corenflos et al. [2021] and Ścibior et al. [2021], attempts to make $\bar{q}_{\phi,\theta}$ differentiable with respect to θ.

Deep Markov Model

The deep Markov model (DMM) was introduced by Krishnan et al. [2017] which used recurrent neural networks (RNN) to parametrise the inference artefact for variational learning. RNNs are a class of neural networks that are used to model sequences using an evolving *deterministic* hidden state. At each time-step, the RNN keeps a "memory" of previous time-steps in its hidden state update. he DMM factorises the same way as Equation 2.51, but with neural network parametrised link functions:

$$p_\theta(x_{1:T}, z_{1:T}) = p(z_1) \prod_{t=2}^{T} p_\theta(z_t|z_{t-1}) \prod_{t=1}^{T} p_\theta(x_t|z_t) \tag{2.115}$$

By using an RNN for inference, we can immediately factorise the posterior in a convenient way:

$$q_\phi(z_{1:T}|x_{1:T}) = q_\phi(z_1|x_{1:T}) \prod_{t=2}^{T} q_\phi(z_t|z_{t-1}, x_{1:T}) \tag{2.116}$$

This is accomplished by using a bi-directional RNN to first embed the full observation sequence. Then, the smoothing posterior is obtained simply from the hidden state of the RNN at a given time-step. A depiction of this model, along with an RNN, is given in Figure 2.3. We also leave further exposition on this model for its use in Chapter 3.

Variational Recurrent Neural Network

Finally, we introduce the variational RNN (VRNN) [Chung et al., 2015]. Similar to the DMM, it uses an RNN to embed the observations and latent variables at each time-step. However, it does this for both the model and inference network, respectively:

$$p_\theta(x_{1:T}, z_{1:T}) = p(z_1)p(x_1|z_1)\prod_{t=2}^{T} p_\theta(x_t|z_{1:t}, x_{1:t-1})p_\theta(z_t|z_{1:t-1}, x_{1:t-1}) \tag{2.117}$$

$$q_\phi(z_{1:T}|x_{1:T}) = q_\phi(z_1|x_1)\prod_{t=2}^{T} q_\phi(z_t|x_{1:t}, z_{1:t-1}) \tag{2.118}$$

By using an RNN in the forward model, this deep latent variable model is no longer a Markov model, which increases inference difficulty. However, this issue is mitigated by sharing RNN hidden state for both model and inference network. We will use this model for the tasks given in Chapter 4, and its graphical model is given in Figure 2.3.

2.4 Control as Inference

In this section, we return to the RL problem and introduce the RL as inference (RLAI) framework. Recall that the RL or control problem is about find an optimal policy that maximises an expected sum of rewards over a time-horizon. In this dissertation, we are only concerned with stochastic policies, for which policy iteration methods can also be used to derive a way of improving the policy at each iteration, similar to Eq 2.88. Given a parametrised policy, $\pi_\phi(a|s)$, the goal is find policy parameters to maximise the Q-function, as before.

$$\phi^* = \arg\max_\phi \mathbb{E}_{s\sim d^{\pi_\phi}, a\sim\pi_\phi}[Q^{\pi_\phi}(a, s)], \tag{2.119}$$

where the stationary distribution of the Markov chain, $d^{\pi_\phi}(s)$, is referred to as the *state occupancy*. Then, it can be shown via the *policy gradient method* [Williams, 1992], that the gradient estimator of the policy parameters, ϕ, can be obtained as:

$$\nabla_\phi \mathcal{L}_\phi = \mathbb{E}_{s\sim d^{\pi_\phi}, a\sim\pi_\phi}[Q^{\pi_\phi}(a_t, s_t)\nabla_\phi \pi_\phi(a_t|s_t)] \tag{2.120}$$

For a more complete treatment of policy gradients, we refer the reader to Schulman et al. [2015b], Sutton et al. [2000]. Here, we will instead approach the RL problem from an inference perspective here, as in Levine [2018]. The goal of RLAI, at a high-level, is to embed the control problem into a probabilistic model, in which solving an inference task corresponds to learning the optimal policy in the model. By reframing the problem this way, we can design better RL algorithms that are tied to advances in approximate inference and probabilistic modelling.

2.4.1 Optimal Policies via Inference

In this section, we formalise the RLAI framework and derive the optimal policy as an posterior distribution in the specified model. Consider the graphical model given by Fig 5.1. We can write a corresponding joint density as:

$$p(\tau, o_{1:T}) = p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) p(o_t|s_t, a_t) p(a_t|s_t), \tag{2.121}$$

where $\tau = \{s_1, a_1, s_2, a_2, ..., s_T, a_T\}$ and the binary random variable, O_t, denotes denote whether an action, a_t, is optimal at some state, s_t:

$$p(o_t|s_t, a_t) = \left(1 - \frac{1}{Z} \exp r_t(a_t, s_t)\right)^{1-o_t} \left(\frac{1}{Z} \exp r_t(a_t, s_t)\right)^{o_t}. \tag{2.122}$$

When we condition on optimality, $O_{1:T} = 1$, we will use the unnormalised density, $\tilde{p}(O_t = 1|s_t, a_t) = \exp\left(r_t(a_t, s_t)\right)$. Finally, the density, $p(a_t|s_t)$ denotes the action prior given the state. In practice, we often will use a uniform prior, i.e. $p(a_t|s_t) = \frac{1}{|\mathcal{A}|}$ to obtain some classic RL algorithms. However, one advantage of RLAI is being able to incorporate prior information [Galashov et al., 2019, Warrington et al., 2021]. Thus, to avoid loss of generality, we continue to specify it in our model.

We next use variational inference to obtain the variational posterior corresponding to the optimal policy. Recall from the Section 2.1.5, that variational inference recasts posterior inference as an optimisation problem which maximises a lower bound on the evidence. In RLAI, the evidence is $O_t = 1$ for all time-steps, t. We define a variational posterior, over the trajectory rollout, τ:

$$q_\phi(\tau) = q(s_1) \prod_{t=1}^{T} q(s_{t+1}|s_t, a_t) q_\phi(a_t|s_t), \tag{2.123}$$

where $q(s_1) = p(s_1)$ and $q(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t, a_t)$, in order to simplify the optimisation objective. Finally, we can now write the evidence lower bound as:

$$\begin{aligned}
\log p(o_{1:T}) &= \log \int p(\tau, o_{1:T}) d\tau \\
&= \log \int \int p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) p(o_t|s_t, a_t) p(a_t|s_t) ds_{1:T} da_{1:T} \\
&= \log \int \int \frac{q_\phi(\tau)}{q_\phi(\tau)} p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) p(o_t|s_t, a_t) p(a_t|s_t) ds_{1:T} da_{1:T} \\
&= \log \mathbb{E}_{q_\phi(\tau)} \Big[\frac{p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) p(o_t|s_t, a_t) p(a_t|s_t)}{q_\phi(\tau)} \Big] \\
&\geq \mathbb{E}_{q_\phi(\tau)} \Big[\log \Big(\frac{p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) p(o_t|s_t, a_t) p(a_t|s_t)}{q(s_1) \prod_{t=1}^{T} q(s_{t+1}|s_t, a_t) q_\phi(a_t|s_t)} \Big) \Big] \\
&= \mathbb{E}_{q_\phi(\tau)} \Big[\log \Big(\frac{\prod_{t=1}^{T} p(o_t|s_t, a_t) p(a_t|s_t)}{\prod_{t=1}^{T} q_\phi(a_t|s_t)} \Big) \Big] \\
&= \mathbb{E}_{q_\phi(\tau)} \Big[\sum_{t=1}^{T} r_t(a_t, s_t) + \log \Big(\frac{p(a_t|s_t)}{q_\phi(a_t|s_t)} \Big) \Big], \qquad (2.124) \\
&= \mathcal{L}_\phi(\tau)
\end{aligned}$$

where in the last line, we condition on $O_{1:T} = 1$ and we omit the normalising constant for clarity.

Now, in order to reduce the variance of the above, we can further marginalise the gradient estimator, $\nabla_\phi \mathcal{L}_\phi(\tau)$. In RLAI, we define $Q(a, s)$ as,

$$Q(s_t, a_t) = \mathop{\mathbb{E}}_{q_\phi(\tau_{t+1:T})|\tau_t = \{a_t, s_t\}} \left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) + \log \frac{p(a_{t'}|s_{t'})}{q_\phi(a_{t'}|s_{t'})} \right],$$

it can be shown that the gradient estimator we want is:

$$\nabla_\phi \mathcal{L}_\phi(\tau) = \sum_{t=1}^{T} \mathbb{E}_{q_\phi(a_t, s_t)} \left[\nabla_\phi \log q_\phi(a_t|s_t) Q(s_t, a_t) \right] \qquad (2.125)$$

We will defer the proof for these objectives and their marginalisation to Chapter 5. Additionally, we subsequently write the action prior, $p(a_t|s_t)$ as π_0 and the parametrised policy, q_ϕ as π_ϕ. As seen, the objective given by maximising the usual variational lower bound on optimality closely follows the objective given by the policy gradient theorem. However, we now have a free choice of both an action prior and the inclusion of an entropy term in the loss.

2.4.2 Actor-Critic Algorithms

In order to improve sample efficiency, many modern policy gradient algorithms rely on function approximation to estimate the expected reward ahead given in Equation 2.4.1 by using a parametrised deep neural network, i.e. $V_\theta(s_t)$ or $Q_\theta(a_t, s_t)$. Additionally, these function approximators can be trained using a gradient of the TD-loss in Equation 2.87. Combining function approximation with the marginalised policy gradient objective given in Equation 2.125 yields the class of algorithms known as actor-critic methods. An actor-critic approach learns both a policy (actor), and a function approximator (critic).

Advantage Actor-Critic

In this section, we cover a powerful *on-policy* actor-critic algorithm known as advantage actor-critic (A2C). On-policy refers to training without memory of environment interactions gathered under a previous policy. When rollouts are collected, they are used immediately in the MC estimate of the TD-loss and policy loss and then discarded. A2C is tackles long time-horizons and large state-spaces by performing multiple policy updates over the course of a single roll-out, thereby reducing the required batch size per update and improving sample efficiency. We use a trained value-function, V_θ, to estimate the expected reward ahead. Then we can use the approximator to partially evaluate the Q-value:

$$\hat{Q}^{\pi_\phi}(a_t, s_t) = \mathop{\mathbb{E}}_{q_\phi^{t+1}}\left[\left(\sum_{t'=t}^{t+k-1} r(s_{t'}, a_{t'}) + \log\frac{\pi_0(a_{t'}|s_{t'})}{\pi_\phi(a_{t'}|s_{t'})}\right) + V_\theta^{\pi_\phi}(s_{t+k})\right], \tag{2.126}$$

where q_ϕ^{t+1} is defined as above and the superscript on V indicates the policy which is used used within q.

A2C makes an additional modifcation to the classic policy gradient update by using an advantage *baseline*. The advantage, $A(a_t, s_t)$, is defined as:

$$\begin{aligned} A(a_t, s_t) &= Q^{\pi_\phi}(a_t, s_t) - \mathbb{E}_{a_t\sim\pi_\phi}[Q^{\pi_\phi}(a_t, s_t)] \\ &\approx \hat{Q}^{\pi_\phi}(a_t, s_t) - V_\theta^{\pi_\phi}(s_t), \end{aligned} \tag{2.127}$$

which reduces the the variance of the estimator and recovers the estimator as in Schulman et al. [2015b]:

$$\mathcal{L}_{\phi}^{\text{A2C}} = \mathbb{E}_{s\sim d^{\pi_\phi}}[\mathbb{E}_{a\sim\pi_\phi}[A(a_t, s_t)]]. \tag{2.128}$$

The policy is optimised using a "rollout" strategy [Bertsekas, 2019], where it seeks to improve performance over the previous policy before taking the next step [Levine and Koltun, 2013, Deisenroth et al., 2013, Bertsekas, 2019]. The final gradient estimators for A2C for both policy and value-function training is:

$$\nabla_\theta \mathcal{L}_\theta^{\text{A2C}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta (\hat{Q}(a_t^i, s_t^i) - V_\theta^\pi(s_t^i))^2 \tag{2.129}$$

$$\nabla_\phi \mathcal{L}_\phi^{\text{A2C}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \Big[\nabla_\phi \log \pi_\phi(a_t^i|s_t^i) A(a_t^i, s_t^i)\Big], \tag{2.130}$$

where $a_t^i \sim \pi_\phi$ and $s_t^i \sim d^{\pi_\phi}$. Finally, we note that classical A2C does not use a default policy, i.e. $\pi_0 = \pi_\phi$ in Equation 2.126.

2.4.3 Streaming Priors

In this section, we show how the inference perspective can inform simple modifications to the policy gradient objective that help improve learning. Instead of ignoring the prior policy, π_0, we explore A2C optimisation cast as an approximate streaming Bayesian update scheme. We present this small experiment as a case-study for how the ideas of this dissertation tie together, i.e. how inference in sequential models can be used to improve reinforcement learning.

Recall the RLAI set of optimality variables, $O^{(1)}, O^{(2)}, ..., O^{(n)}$. Treating these as a stream of incoming observations, we can utilise the recursive Bayesian updating identity as in Equation 2.62:

$$p(\tau|O^{(1)}, O^{(2)}, ...O^{(k)}) \propto p(O^{(k)}|\tau)p(\tau|O^{(1)}, ..., O^{(k-1)}). \tag{2.131}$$

Whenever we obtain a new observation, $O^{(k+1)}$, we write the new posterior as:

$$p(\tau|O^{(1:k+1)}) \propto p(O^{(k+1)}|\tau)p(\tau|O^{(1:k)}), \tag{2.132}$$

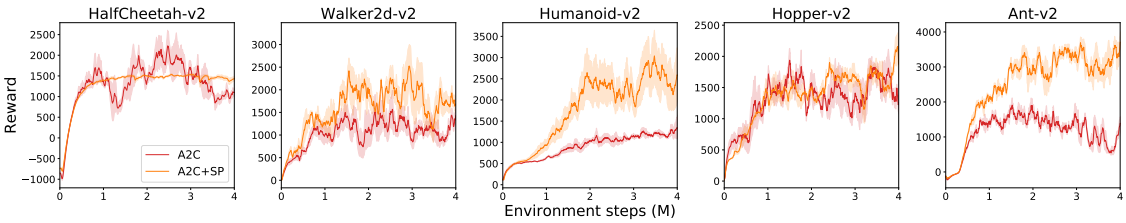


Figure 2.4: Comparison of A2C performance with and without SP regularisation on MuJoCo benchmark tasks. Each training curve is averaged over three replicates and we apply EMA smoothing at 0.99. With 'Streaming Priors', on-policy updates are able to learn with much high sample efficiency in Walker2d, Ant, and Humanoid benchmarks.

where our "prior" knowledge of τ is now represented in the previous posterior, $p(\tau|O^{(1:k)})$.

This inspires a simple approximate algorithm for incorporating prior information. We initialise π_0 to an uninformative prior on the actions. Then, after a certain number of environment interactions, we swap the *variational posterior*, π_ϕ, for π_0 and continue the policy optimisation using Equation 2.126. We refer to this as approximate since it uses variational inference to obtain the posterior that will be used as the prior for the next Bayesian update. Galashov et al. [2019] use this update to compare their custom priors methods against. Here we show how their incorporation already stabilises A2C over its classical non-RLAI interpretation.

Finally, the objective we are optimising before swapping the learned policy for the fixed action prior can be summarised as:

$$\pi_{0(k+1)} = \arg\min_{\phi} - \mathop{\mathbb{E}}_{\tau\sim q_\phi}\left[\sum_{t=1}^{T} r(s_t, a_t) - \log \frac{\pi_\phi(a_t|s_t)}{\pi_{0(k)}(a_t|s_t)}\right]. \tag{2.133}$$

Streaming Prior Experiment

We implement streaming priors (SP) on top of A2C using modified code from Kostrikov [2018]. In Figure 2.4, we find that A2C+SP improves learning substantially, achieving higher final convergence in three out of five tasks from the MuJoCo control suite [Todorov et al., 2012] while using an identical hyper-parameter set as standard A2C.

2.4.4 Bayesian Statistics and Applications

The above use case is a canonical example of the following application papers presented in this integrated thesis. We apply the ideas and methods of the topics covered in this background to argue for the use of sequential inference in a wide array of contexts. We start with a generative modelling and practical case for amortised inference in the next chapter.

3

Distributed Training with Deep Markov Models

Contents

3.1 Introduction 61
3.2 Background and Motivation 63
3.2.1 Effect of Stragglers 65
3.3 Methodology 65
3.3.1 Modelling Compute Cluster Worker Performance 68
3.3.2 Neural Network Architectures 74
3.3.3 Parameter Server and Worker Design 76
3.4 Experiments 77
3.4.1 Small Compute Cluster 78
3.4.2 Large Scale Computing 79
3.4.3 Experimental Details 80
3.5 Discussion 81

3.1 Introduction

Deep learning success stories are predicated on large neural network models being trained using ever larger amounts of data. While the computational speed and memory available on individual computers and GPUs continually grows, there always will remain some problems and settings in which the amount of training data available will not fit entirely into the memory of one computer. What is more,

and even for a fixed amount of data, as the number of parameters in a neural network or the complexity of the computation it performs increases, so too do the incurred economic and time costs to train. Both large training datasets and complex networks inspire parallel training algorithms.

In this work we focus on parallel stochastic gradient descent (SGD). Like the substantial and growing body of work on this topic [Recht et al., 2011, Dean et al., 2012, McMahan and Streeter, 2014, Zhang et al., 2015] we too focus on gradient computations computed in parallel on "mini-batches" drawn from the training data. However, unlike most of these methods which are *asynchronous* in nature, we focus instead on improving the performance of *synchronous* distributed SGD, very much like Chen et al. [2016], upon whose work we directly build.

A problem in fully synchronous distributed SGD is the *straggler effect*. This real-world effect is caused by the small and constantly varying subset of worker nodes that, for factors outside our control, perform their mini-batch gradient computation slower than the rest of the concurrent workers, causing long idle times in workers which already have finished. Chen et al. [2016] introduce a method of mitigating the straggler effect on wall-clock convergence rate by picking a fixed cut-off for the number of workers on which to wait before synchronously updating the parameter on a centralized parameter server. They found, as we demonstrate in this work as well, that the increased gradient computation throughput that comes from reducing idle time more than offsets the loss of a small fraction of mini-batch gradient contributions per gradient descent step.

Our work exploits this same key idea but improves the way the likely number of stragglers is identified. In particular we instrument and generate training data once for a particular compute cluster and neural network architecture, and then use this data to train a lagged generative latent-variable time-series model that is used to predict the joint run-time behaviour of all the workers in the cluster. For highly contentious clusters with poor job schedulers, such a model might reasonably be expected to learn to model latent states that produce correlated, grouped increases in observed run-times due to resource contention. For well-engineered clusters,

such a model might learn that worker run-times are nearly perfectly independently and identically distributed.

Specifying such a flexible model by hand would be difficult. Also, we will need to perform real-time posterior predictive inference in said model at runtime to dynamically predict straggler cut-off. For both these reasons we use the VAE loss given in Equation2.94 to simultaneously learn not only the model parameters but also the parameters of an amortised inference neural network [Ritchie et al., 2016, Le et al., 2017a] that allows for real-time approximate predictive inference of worker run-times.

The main contributions of this paper are:

- The idea of using amortised inference in a deep state space model to predict compute cluster worker run-times, in particular for use in a distributed synchronous gradient descent algorithm.
- The BDSGD algorithm itself, including the approximations made to enable real-time posterior predictive inference.
- The empirical verification at scale of the increased gradient computation throughput that our algorithm yields when training deep neural networks in parallel on large clusters.

The rest of the paper is organised as follows. In Section 3.2, we give necessary background on why and how synchronous distributed SGD can be improved. In section Section 3.3, we explain our choice of generative model for cutoff determination. In section Section 3.4, we presents our experimental results.

3.2 Background and Motivation

In stochastic gradient descent, we use unbiased estimates of gradients to update parameter settings. Synchronous distributed SGD differs from single-threaded mini-batch SGD in that the mini-batch of size m is distributed to N total workers that locally compute sub-mini-batch gradients before communicating the result

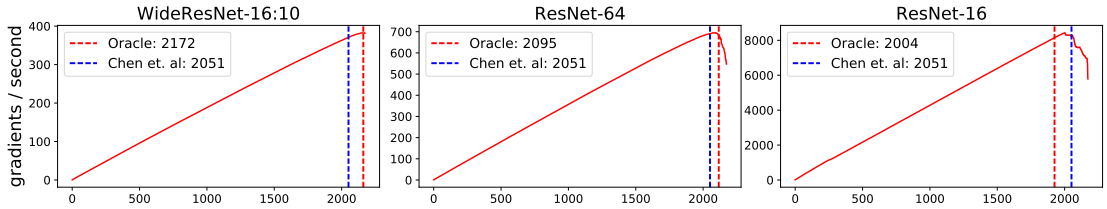


Figure 3.1: Oracle throughput curves (best achievable in hindsight) for synchronous SGD runs for three different neural networks on the same 2175-worker cluster. From left: low variance (1-2% stragglers), medium variance (2-4% stragglers), and high variance (8-12% stragglers) throughput curves with mean±std worker gradient computation times being 5.34±0.13 seconds, 2.83±0.077 seconds, and 0.24±0.018 seconds, respectively. The x-axes are the number of workers. The y-axes show throughput achieved if all workers beyond the x-axis value are ignored. When runtimes for gradient computations have low variance relative to total runtime, Chen et. al underestimates the optimal cutoff point whereas when runtimes have proportionally higher variance, Chen et al. overestimates the optimal cutoff point. Here, optimality is defined as maximising the total throughput of the cluster during neural network training, i.e. "gradients per second" (see Equation 3.2). Our approach achieves more accurate estimates of the optimal cutoff in both scenarios.

back to a centralized parameter server that updates the parameter vector using an update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{1}{N} \sum_{i=1}^{N} f\left(\theta^{(t)}, (i-1)\frac{m}{N}, i\frac{m}{N}\right) \tag{3.1}$$

with

$$f(\theta, a, b) = \frac{1}{b-a} \sum_{k=0}^{b-a} \nabla_{\theta^{(t)}} \mathcal{L}(\theta, z^{(k)}, y^{(k)})$$

where θ are the network parameters, $\mathcal{L}$ is the loss function, and α is the learning rate. Although not shown, asynchronous SGD is lock-free, and parameter updates are made whenever any worker reports a sub-mini-batch gradient to the parameter server which results in out-of-date or stale gradient information [Recht et al., 2011]. Unlike asynchronous distributed SGD, synchronous distributed SGD is equivalent to single-threaded SGD with batchsize, m. This allows hyper-parameters, α and m, to be tuned in the distributed setting without having to consider the possibility of stale gradients [Hoffer et al., 2017].

3.2.1 Effect of Stragglers

In synchronous SGD, we can attribute low throughput, in the sense of central parameter updates per unit time, to the straggler effect that arises in real-world cluster computing scenarios with multiple workers computing in parallel. Consider Equation 3.1, in which f is computed independently on a memory-isolated logical processor. Let x_j be the time it takes for f to be computed on the worker indexed by j for $j \in 1...N$. Distributed compute clusters are not perfect, otherwise x_j would be a constant, independent of j, and all workers would finish at the same time and only be idle while the parameter server aggregates the gradients and sends back the new parameters. Instead x_j is actually random. Moreover, the joint distribution of all the x_j's is likely, again in real-world settings, to be non-trivially correlated owing to cluster architecture, etc. For instance most modern clusters consist of computers or graphics processing units each in turn having a small number of independent processors, so slow-downs in one logical processing unit are likely to be exhibited by others sharing the same, for instance, bus or network address. What is more, in modern operating systems, time-correlated contention is quite common, particularly in clusters under queue management systems, when, for instance, other processes, operating system or user, are concurrently executed. All this yields worker compute times that may be non-trivially correlated in both time and in "space".

Our aim is to significantly reduce the effect of stragglers on throughput and to do so by modelling cluster worker compute times in a way that intelligently and adaptively responds to the kinds of correlated run-time variations actually observed in the real world. What we find is that doing so improves overall performance of distributed mini-batch SGD.

3.3 Methodology

Our approach works by maximising the total throughput of parameter updates during a distributed mini-batch SGD run. The basic idea, shared with Chen et al.

[2016], is to predict a cutoff, $c_t < N$, for each iteration of SGD which dictates the total number of workers on which to wait before taking a gradient step in the parameter space. While Chen et al. [2016] use a fixed cutoff, $c_t = 0.943 \cdot N$; $\forall t$, we would like for c_t to be evolving dynamically with each iteration and in a manner specific to each compute cluster, neural network architecture pair. We note that for overall rate of convergence, throughput is not the exact quantity we wish to maximise; that being some quantity related to the rate of expected gain in objective function value instead, but it is the proxy we use in this work. Also, paradoxically, lower throughput, by virtue of smaller mini-batch sizes, may in some instances increase the rate of convergence, an effect previously documented in the literature [Masters and Luschi, 2018].

The central considerations are: what is the notion of throughput we should optimise? And how do we predict the cutoff that achieves it? Simply optimising overall run-time admits a trivial and unhelpful solution of setting $c_t = 0$. Each iteration and the overall algorithm would then take no time but achieve nothing. Instead we seek to maximise the number of workers to finish in a given amount of time, i.e. throughput $\Omega(c)$, which we define to be:

$$\Omega(c) = \frac{c}{\tilde{x}_{(c)}}$$

where c indexes the *ordered* worker run-times $\tilde{x}_{(c)}$. Note that, for now, we avoid indexing run-times by SGD loop iteration. Soon, we will address temporal correlation between worker runtimes.

With this definition, we can plot throughput curves that show how throughput drops off as the straggler effect increases (Figure 3.1). On the well-configured cluster used to produce Figure 3.1, a high percentage of workers (between 80-95%) finish at roughly the same time, so in this regime, throughput of the system increases linearly for each additional worker. However, continuing to wait for more workers includes some stragglers which eventually decreases the overall throughput.

We define our objective to be to maximise the throughput of the system as defined above, i.e. $\arg\max_c \Omega(c)$ at all times. This also implicitly handles the

tradeoff between iteration speedup and the learning signal reduction that comes from using a higher variance gradient estimate given by discarding gradient information. We note that our choice of optimality assumes that the more gradients, and therefore dataset examples, seen, the better the learning signal is during each mini-batch update. But when we consider the optimisation landscape of SGD, our objective is only a proxy for the true target, which is optimising the rate of reducing the validation loss given subsets of training examples seen. As such, a more idealised notion of optimality may be to directly model the relationship between gradients accumulated and learning signal updates, in a similar way as that used in RL to prioritise replay buffer examples [Schaul et al., 2015]. However, it is not clear how to do this in the distributed cutoff setting as many gradients may be permanently lost and the learning signal is far more difficult to model and changes per update to network parameters. Schaul et al. [2015] also find that fully greedy prioritisation does not work for the same reason in their setup, as some examples are never revisited in that case.

With our goal of optimising throughput in order to maximise gradients seen during training, we also note that using as many gradients as possible may lead to poor convergence results in the overall neural network training task on a distributed cluster. While large scale training is clearly preferred in order to take advantage of modern parallel computing architectures, it may lead to a drawback in generalisation to test-set accuracy, as pointed out in Keskar et al. [2016]. Subsequent work by Amir et al. [2021] derive generalisation bounds for full-batch gradient descent (GD) vs SGD, finding that SGD generalises better in most cases. Smith et al. [2020] show that the test accuracy decreases even with careful hyperparmeter tuning to ensure optimal generalisation to test sets. However, we note that the goal of non-stochastic training of neural networks is still a promising and ongoing research problem, as it would further the reproducibility results if achieved. Recent work from Geiping et al. [2021] closes the gap between full-batch GD and SGD generalisation, using extensive hyper-parameter search and regularisation methods. When using randomised data-augmentation with hyperparameter search, they find

that GD can match SGD on the test set performance of the same dataset and neural network pairs. However, their results show that as the total training set increases, the gap begins to widen again, i.e. with SGD outperforming GD, despite careful tuning of both.

In practice, we will show that our notion of optimality does improve neural network training speeds given a model which can accurately predict throughput. Setting the cutoff optimally and dynamically requires a model which is able to learn and predict the joint ordered run-times of all cluster workers. With such a model, we can make informed and accurate predictions about the next set of run-times per worker and consequently make a real-time near-optimal choice of c for the subsequent loop of mini-batch gradient calculations. Furthermore, we note that failing to optimise throughput here is less likely to occur than failing to optimise change in validation loss, the more idealised optimality discussed above. The former is dependent on fully observable cluster run-times and the latter is dependent on a complex relationship between training examples and gain in validation accuracy, as well as a further dependence on the current parameter set of the neural network. Given this, an under-optimised throughput would be less catastrophic than an under-optimised validation loss model, as well as being more generalisable to *any* new neural networks trained on the same hardware. How we model compute cluster worker performance follows.

3.3.1 Modelling Compute Cluster Worker Performance

As before, let $x_j \in \mathbb{R}^+$ be the time it takes for f to be computed on the worker indexed by j. Assume that these are distributed according to some distribution p. Given a set of n, $p(\cdot)$-distributed random variables $x_1, x_2, \ldots, x_n$ we wish to know the joint distribution of the n *sorted* random variables $\tilde{x}_{(1)}, \tilde{x}_{(2)}, \ldots, \tilde{x}_{(n)}$. Such quantities are known as "order statistics." Each $p(\tilde{x}_{(j)})$ describes the distribution of the j^{th} largest sorted run-time under independent draws from this underlying distribution. Taking the mean of each order statistic allows us to derive a cutoff

using our notion of throughput, given as:

$$\arg\max_{c} \Omega(c) = \arg\max_{c} \mathbb{E}[\frac{c}{\tilde{x}_{(c)}}] \tag{3.2}$$

Elfving Cutoff

The first model of runtimes we consider assumes that they are are independent and identically distributed (iid) Gaussian. Under the assumption that $x \sim \mathcal{N}(\mu_x, \sigma_x^2)$ the distribution of each order statistic $p(\tilde{x}_{(1)}), p(\tilde{x}_{(2)}), ..., p(\tilde{x}_{(n)})$ is independent and $\mathbb{E}[\tilde{x}_{(1)}] \leq \mathbb{E}[\tilde{x}_{(2)}], ..., \leq \mathbb{E}[\tilde{x}_{(n)}]$.

Under the given iid normality assumption the distribution of the each order statistic has closed form:

$$p(\tilde{x}_{(j)}) := p_{\tilde{X}_{(j)}}(x) = \frac{n!}{(j-1)!(n-j)!}[F_X(x)]^{j-1}[1-F_X(x)]^{n-j}p(x)$$

where $F_X(x)$ denotes the CDF of the iid Gaussian and $p(x) = \mathcal{N}(\mu_x, \sigma_x^2)$. Then $p_{\tilde{X}_{(j)}}(x)$ gives us the density of x under the j^{th} order-statistic random variable, $\tilde{X}_{(j)}$, and we adopt the shorthand, $p(\tilde{x}_{(j)})$ for convenience. Note that each order statistic's distribution, including the maximum, increases as the variance of the run-time distribution increases, while the average run-time does not.

As a baseline in subsequent sections we will use an approximation of the expected order statistics under this iid normality assumption. This is known as the Elfving (1947) formula [Royston, 1982]:

$$\mathbb{E}[\tilde{x}_{(j)}] \approx \mu_t + \Phi^{-1}\left(\frac{n-\frac{\pi}{8}}{j-\frac{\pi}{4}+1}; 0, 1\right)\sigma_t \tag{3.3}$$

Here, we note that the Elfving model requires full observability of runtimes to predict subsequent runtimes in a production setting. In practice, the parameters μ_t, σ_t in Equation 3.3 are fit using maximum likelihood on the first fixed lagged window and remain static during the remainder of the run.

While some clusters may approximate the strong assumptions required to use the Elfving formula for cutoff prediction, most compute clusters will emit joint order statistics of non-Gaussian distributed correlated random variables, for which no analytic expression exists. However, if we have a predictive model of the joint

distribution of the x_j's (or $\tilde{x}_{(j)}$'s), we can use sorted samples from such a joint distribution to obtain a Monte Carlo approximation of the order-statistics. In the next section, we will detail how to construct the predictive model in order to learn correlations of worker runtimes.

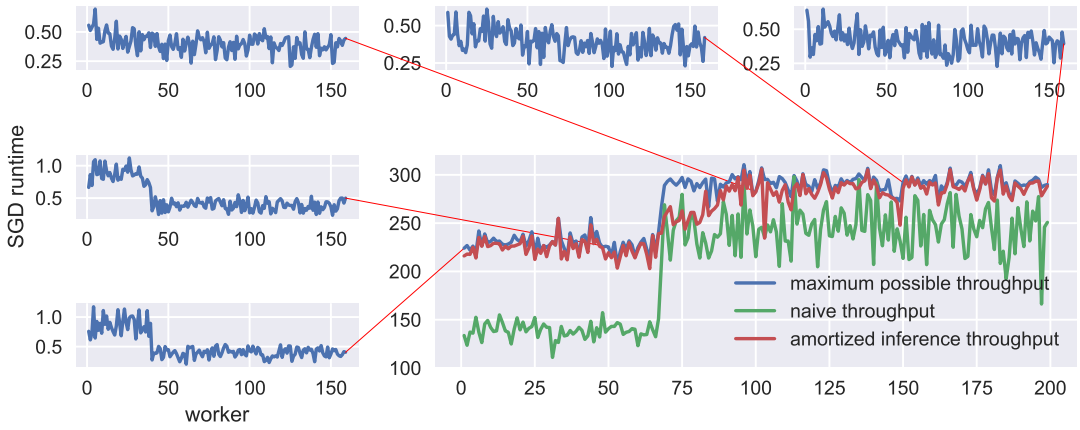


Figure 3.2: Predicted throughputs. Each runtime plot (5 surrounding the top figure) shows the individual runtimes of all workers (x-axis index) during an iteration of SGD on a 158 node cluster. We highlight SGD iterations 1, 50, 100, 150, and 200 which highlight two significantly different regimes of persistent time-and-machine-identity correlated worker runtimes. The bottom-right figure displays a comparison of throughputs achieved at each of the 200 SGD iterations by waiting for all workers to finish (green) and using our approach, BDSGD, (red) relative to the ground truth maximum achievable (blue). BDSGD predicts cutoffs that achieve near optimal throughput, in a setting where fixed-cutoffs are insufficient and Elfving assumptions do not hold.

Bayesian Cutoff

In this section, we formally introduce our proposed training method, which we call Bayesian distributed SGD (BDSGD). Before introducing the design of the generative model we use to predict worker run-times, first consider the practical implications of using a generative model instead of a purely autoregressive model. In short we can only consider worker run-time prediction models that are extremely sample efficient to train. We also can only consider a kind of model that allows real-time prediction because it will be in the inner loop of the parameter server and used to predict at run-time how many straggling workers to ignore. Deep neural net

auto-regressors satisfy the latter but not the former. Generative models satisfy the former but historically not the latter; except now deep neural net guided amortised inference in generative models does. This forms the core of our technical approach.

We will model the time sequence of observed joint worker run-times $x_{T-\ell}, \ldots, x_T$ using a deep state space model where $z_{T-\ell}, \ldots, z_T$ is the time evolving unobserved latent state of the cluster. In this framework, $x_{T-\ell:T}$ may be replaced with directly modelling $\tilde{x}_{T-\ell:T}$, and we continue with $x_{T-\ell:T}$ for clarity. The dependency structure of our model factorizes as

$$p_\theta(x_{T-\ell:T}, z_{T-\ell:T}) = \prod_{i=T-\ell}^{T} p_\theta(z_i|z_{i-1}) \prod_{i=T-\ell}^{T} p_\theta(x_i|z_i)$$

where, for reasons specific to amortising inference, we will restrict our model to a fixed-lag ℓ window. The principle model use is the accurate prediction of the next set of worker run-times from those that have come before:

$$p(x_{T+1}|x_{T-\ell:T}) = \int p_\theta(x_{T+1}|z_{T+1}) p_\theta(z_{T+1}|z_T) p(z_{T-\ell:T}|x_{T-\ell:T}) dz_{T-\ell:T+1} \tag{3.4}$$

For our generative model, we emit Gaussian distributed outputs, despite modelling "run-time", a strictly positive value. We note that a more suitable choice would be a Gamma distributed output, which would also not increase the learning difficulty of our VAE model if used instead. However, Gaussian distributions work suitably well for many strictly positive datasets (e.g. heights, weights, populations, etc.), when the negative values are several standard deviations left of the mean values in the learned model. Furthermore, we choose the Gaussian distributed for a more direct comparison to our baseline Elfving model, which requires a Gaussian iid assumption to use.

Model Learning and Amortised Inference

With the course-grained model dependency defined, it remains to specify the fine-grained parameterisation of the generative model, to explain how to train the model, and to show how to perform real-time approximate inference in the model.

We use the deep linear dynamical model introduced by Krishnan et al. [2017], that constructs the LDS with MLP link functions between Gaussian distributed

latent state and observation vectors. Inference in the model is done with a non-Markovian proposal network. Namely, the transition and emission functions in our model are parameterised by neural networks, the inference over which is guided by an RNN. For a detailed exposition, refer back to the DMM model given in Chapter 2−Section 2.3.4.

The flexibility of such a model allows us to avoid making restrictive or inappropriate assumptions that might be quite far from the true generative model while imposing rough structural assumptions that seem appropriate like correlation over time and correlation between workers at a given time.

The remaining tasks are to, given a set of training data, i.e. fully observed SGD runtimes specific to a cluster, learn θ *and* train an amortised inference network to perform realtime inference in said model. For this we utilise the VAE loss used for amortised inference in deep probabilistic programming with guide programs [Ritchie et al., 2016].

We use stochastic gradient descent to simultaneously optimise the variational evidence lower bound (ELBO) with respect to both ϕ and θ:

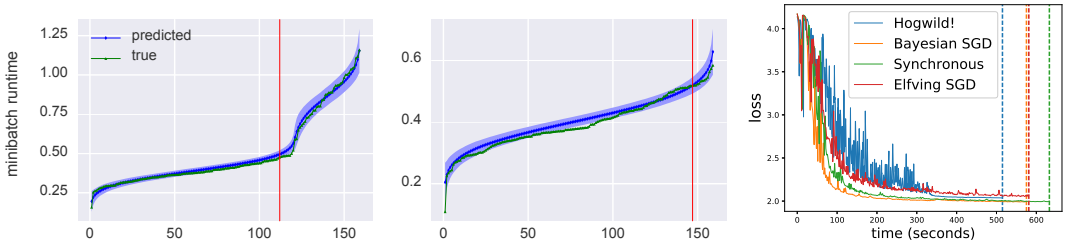
$$\text{ELBO} = \mathbb{E}_{q_\phi(z_{T-\ell:t}|x_{T-\ell:T})}\left[\log\left(\frac{p_\theta(x_{T-\ell:t}, z_{T-\ell:t})}{q_\phi(z_{T-\ell:t}|x_{T-\ell:T})}\right)\right]$$

where

$$q_\phi(z_{T-\ell:t}|x_{T-\ell:T}) = \prod_{t=T-\ell}^{T} q_\phi(z_t|z_{T-\ell:t-1}, x_{T-\ell:T}).$$

Doing this yields a useful by-product. Maximizing the ELBO also drives the KL divergence between $q_\phi(z_{T-\ell:t}|x_{T-\ell:T})$ and $p_\theta(z_{T-\ell:t}|x_{T-\ell:t})$ to be small. We will exploit this fact in our experiments to enable cutoff prediction. In particular we will directly approximate Equation 3.4 by:

$$\begin{aligned} p(x_{T+1}|x_{T-\ell:T}) &\approx \int p_\theta(x_{T+1}|z_{T+1})p_\theta(z_{T+1}|z_T)q_\phi(z_{T-\ell:T}|x_{T-\ell:T})dz_{T-\ell:T+1} \\ &\approx \frac{1}{K}\sum_{k=1}^{K} p_\theta(x_{T+1}|z_{T+1})p_\theta(z_{T+1}|z_T^{(k)}) \end{aligned} \tag{3.5}$$



(a) SGD runtime profiles for two iterations **(b)** MNIST full training run

Figure 3.3: Cumulative results of 158-worker cluster on 3-layer MLP training. The left two figures are plots of observed runtimes vs predicted runtime order statistics of two iterations of SGD of the validation set in the BDSGD training step. The maximum throughput cutoff under the model predictions is shown in red, indicating a large chunk of idle time is reduced as a result of stopping early. Notably, when there are exceptionally slow workers present, the cutoff is set to proceed without any of them as seen in left figure of subplot (a). Subplot (b) shows MNIST validation loss for model-based methods, Elfving and BDSGD, compared to naive synchronous (waiting for all workers) and asynchronous (Hogwild) approaches, where dashed vertical lines indicate the time at which the final iteration completed (all training methods perform the same number of mini-batch gradient updates). In the lower right corner of subplot (b), we observe that BDSGD achieves the fastest time to complete the fixed number of gradient updates of the synchronous methods, while also achieving the lowest validation loss.

with $z_T^{(k)}$ being the last-time-step marginal of the kth of K samples from $q_\phi(z_{T-\ell:T}|x_{T-\ell:T})$. The predictive runtimes given by this technique can now be used to determine the throughput-maximising cutoff in the objective given by Equation 3.2.

Handling Censored Run-times

As described, we use the learned inference network to predict future cutoffs rather than the generative model. Because variational inference jointly learns the model parameters along with the inference network, we could theoretically use an inference algorithm such as SMC [Doucet et al., 2001] for more accurate estimates of the true posterior predictive distribution. However, our cutoff prediction must be done in an amortised setting because we rely on it to be set for a gradient run prior to the updates returning from the workers. In a setting which requiring fast, repeated inference, using an amortised method is often the only feasible approach, especially in large complex models.

However, when using amortised inference, there is a practical issue of dealing with partially observed, censored data. Since at run-time we are only waiting for c gradients up to the cutoff, and are in fact actually killing the straggling workers, we do not have the run-time information from the straggling workers that would have finished past the cutoff. Inference in the generative model could directly be made able to deal with censored data, however our inference network runs an RNN which was trained on fully observed run-time vectors and therefore requires fully observed input to function correctly. Because of this, we deploy an effective approximate technique for imputing the missing worker runtime values, which samples a new uncensored data point for every worker whose gradients are dropped. Because we push estimates of the approximate posterior through the generative model, we have a predictive run-time distribution for the *current* iteration of SGD before receiving actual updates from any worker. When eventually the cutoff is reached, and the corresponding rate censor is observed, we are left with run-time distributions truncated at $\tilde{x}_{(c)}$:

$$p(\tilde{x}; \tilde{x} > \tilde{x}_c) = \frac{p(\tilde{x})}{\int_{\tilde{x}_{(c)}}^{\infty} p(\tilde{x}) d\tilde{x}} \tag{3.6}$$

where we have left off the time index for clarity and $\tilde{x}$ is any one of the censored worker runtime observations. When a censored value is required, we take its corresponding predicted run-time distribution and sample from its right tailed truncation to get an approximate value for that missing run-time. We find that this method works well to propagate the model forward, leading to still accurate predictions.

3.3.2 Neural Network Architectures

The generative model factorises as a latent variable state space model:

$$p_\theta(x_{T-\ell:T}, z_{T-\ell:T}) = \prod_{i=T-\ell}^{T} p_\theta(z_i|z_{i-1}) \prod_{i=T-\ell}^{T} p_\theta(x_i|z_i)$$

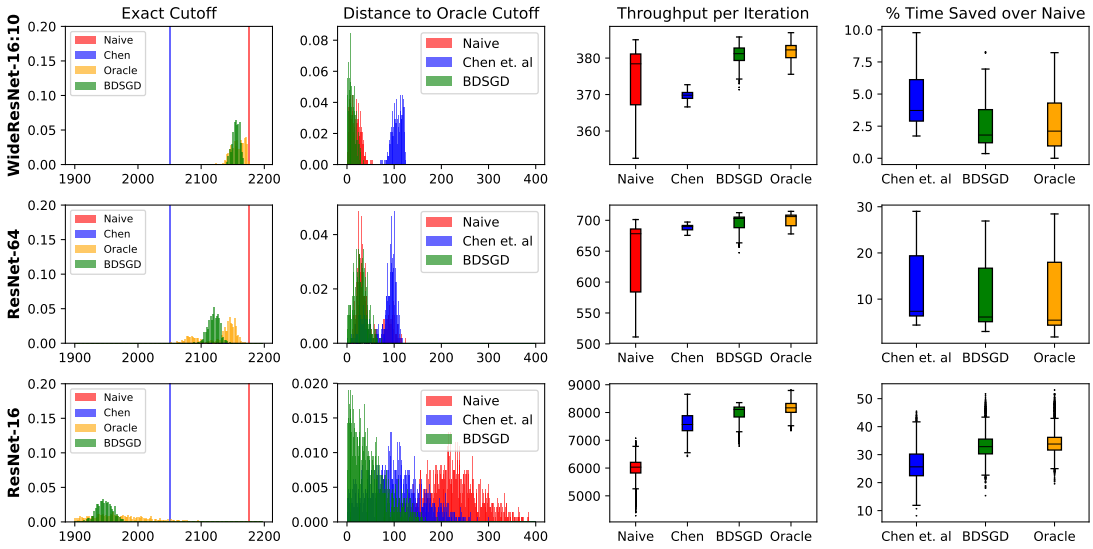


Figure 3.4: Comparison of BDSGD against Chen et. al on synchronous test data for three neural network training runs with no early worker termination. From left: histogram of cutoff set by each method (Chen et. al always uses 2051 workers and naive always uses all 2175 workers), histogram of the absolute difference between chosen cutoff and the oracle cutoff which would achieve highest throughput, quartile-boxplot of gradient computation throughput per iteration, and percent wall clock time saved using each method over fully synchronous (naive) method. We observe that BDSGD best predicts the oracle cutoff, which leads to highest throughput on all three cases, and highest expected wall-clock savings when BDSGD's average cutoff is lower than that fixed by Chen et. al (ResNet-16 model).

where $p_\theta(z_t|z_{t-1})$ is parametrised as:

$$z_t \sim \mathcal{N}(G_\theta(z_{t-1}), H_\theta(z_{t-1})))$$
$$G_\theta(z_{t-1}) = (1 - g_t) \cdot \text{1-MLP}(z_{t-1}, \text{Identity}) + g_t \cdot \text{2-MLP}(z_{t-1}, \text{ReLU}, \text{Identity})$$
$$g_t = \text{2-MLP}(z_{t-1}, \text{ReLU}, \text{Sigmoid})$$
$$H_\theta(z_{t-1}) = \text{1-MLP}(\text{ReLU}(G_\theta(z_{t-1})), \text{Softplus})$$

and $p_\theta(x_t|z_t)$ is parametrised as:

$$x_t \sim \mathcal{N}(I_\theta(z_t), J_\theta(z_t)))$$
$$I_\theta(z_t) = \text{2-MLP}(z_{t-1}, \text{Identity}, \text{Identity})$$
$$J_\theta(z_t) = \text{2-MLP}(I_\theta(z_t), \text{ReLU}, \text{Softplus})$$

where n-MLP($\cdot$, $A, B, ...$) denotes an n-layer multilayer perceptron with nonlinearities, A, B, etc.

The inference network is a structured left-right model, conditioning on observations from the future and past:

$$q_\phi(z_t|z_{T-\ell:t}, x_{T-\ell:T}) = \prod_{j=1}^{N} \mathcal{N}(\mu_{q_\phi}(j), \sigma_{q_\phi}(j))$$

where $*$ denotes scalar multiplication below:

$$\begin{aligned}
z_t &\sim \mathcal{N}(K_\phi(z_{t-1}, x_{T-\ell:T}), L_\phi(z_{t-1}, x_{T-\ell:T})) \\
K_\phi(z_{t-1}, x_{T-\ell:T}) &= \text{1-MLP}(h_{\text{out}}, \text{Identity}) \\
L_\phi(z_{t-1}, x_{T-\ell:T}) &= \text{1-MLP}(K_\phi(z_{t-1}, x_{T-\ell:T}), \text{Softplus}) \\
h_{\text{out}} &= \frac{1}{3} * (\text{1-MLP}(z_{t-1}, \text{Tanh}) + h_{\text{left}} + h_{\text{right}}) \\
h_{\text{left}} &= \text{RNN}(x_{T-\ell:t}, \text{ReLU}) \\
h_{\text{right}} &= \text{RNN}(x_{t+1:T}, \text{ReLU})
\end{aligned}$$

3.3.3 Parameter Server and Worker Design

Here, we provide the parameter server and worker client pseudo-code. Parameter servers use amortised inference to predict cutoff in line 9 of Algorithm 3. For static cutoff methods, the predict cutoff function is no longer a function of q_ϕ and returns a fixed number.

The await parameter server udpate and thread termination mechanism in lines 17-19 of Algorithm 2 can be implemented using threads, asynchronous promises, or continuous polling of parameter server socket.

Algorithm 2 Parameter Server

Input: model p_θ, inference net q_ϕ,
learning rate α, workers N

r $\leftarrow \{\}$
$t \leftarrow 0$
while not converged **do**
  $G \leftarrow \langle\rangle$
  $x_t \leftarrow \langle\rangle$
  $c, \tilde{x}_{(1):(N)} \leftarrow$ predict(p_θ, q_ϕ, r)
  for $0 < j \leq c$ **do**
    receive $\langle \nabla_\theta F \rangle^n$, $x^n \leftarrow$ any
    $x_t \leftarrow x_t \cup x^n$
    $G \leftarrow G + \langle \nabla_\theta F \rangle^n$
  end for
  $\theta' \leftarrow \theta - \alpha \frac{1}{c} G$
  send $\theta' \rightarrow$ all workers
  $x_c \leftarrow \max(x_t)$
  for $c < k \leq N$ **do**
    $x^k \leftarrow$ sample $p(\tilde{x}_{(k)}; \tilde{x}_{(k)} > x_c)$
    $x_t \leftarrow x_t \cup x^k$
  end for
  runtimes{key : t} $\leftarrow x_t$
  increment t
end while

Algorithm 3 Worker

Input: data split X,
local batch size m,
worker index n

procedure compute_gradient(θ):
  $x \leftarrow$ timestamp()
  for $i = 1$ **to** b
    $x^{(z)}, y^{(z)} \leftarrow$ sample(X)
    $\langle \nabla_\theta F \rangle^n \leftarrow \frac{1}{m} \nabla F(\theta, x^{(z)}, y^{(z)})$
  end for
  $x^n \leftarrow$ timestamp() $- x$
  send $\langle \nabla_\theta F \rangle^n$, $x^n \rightarrow$ server
end procedure

$\theta \leftarrow$ initialize(θ)
while not converged **do**
  spawn thread **run**
    compute_gradient(θ)
  await receive $\theta' \leftarrow$ server
  $\theta \leftarrow \theta'$
  terminate thread
end while

3.4 Experiments

To test our model's ability to accurately predict joint worker runtimes, we perform experiments by training 4 different neural network architectures on one of two clusters of different architectures and sizes. To train the BDSGD generative models used, we first train the neural network architecture of interest using fully synchronous SGD and use the recorded worker runtimes during each SGD iteration to learn a corresponding generative model of that particular neural network's gradient computation times on the cluster. As we will highlight, BDSGD model-based estimates of expected runtimes are sufficient to derive a straggler cutoff from their order statistics that leads to increased throughput and/or faster training times in real world situations.

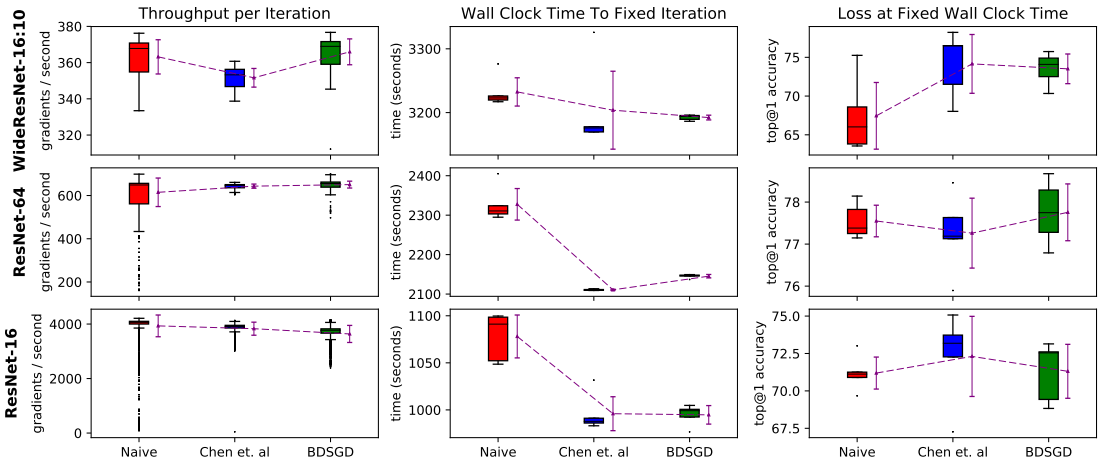


Figure 3.5: Production training runs comparing BDSGD, Chen et. al, and fully synchronous (naive) methods. The neural networks on each row are trained to convergence using the three training methods, five times each. Each row in this figure corresponds to the same row in Figure 3.4. The columns show quartile-boxplots and mean±std (purple error bar). From left to right: iteration throughput achieved during training, wall-clock time to fixed iteration (400, 600, and 1500 for WideResNet, ResNet-64, and ResNet-16, respectively), and validation loss at a fixed wall-clock time (set to the wall-clock time at 50% of the total training time taken by Chen et. al's method). The two big neural network models, ResNet-64 and WideResNet-16:10, achieve the highest throughput when using BDSGD, training to roughly equal or better validation loss at a fixed wall-clock time, while improving total training time by roughly the same as Chen et. al despite setting much higher cutoffs. The ResNet-16 model demonstrates the ability to run a modified BDSGD on a small network, where the amortised inference exceeds gradient computation time on average.

3.4.1 Small Compute Cluster

On one cluster comprised of four nodes of 40 logical Intel Xeon processors, we benchmark Elfving and BDSGD cutoff predictors against the fully synchronous and fully asynchronous SGD with a 158-worker model by using each method to train a 2-layer CNN on MNIST classification. At this scale, and on a small neural network model, we are still able to deploy a Hogwild training scheme that does not diverge.

This cluster uses a job scheduler that allows jobs to be run concurrently on each node. From one of the fully synchronous SGD runs used to gather runtime data for BDSGD model learning, we find that 40 of the workers localised to one machine node experience a temporary 2× slowdown in gradient computation times, which we believe to have been caused by another job batched onto the node.

Figure 3.2 shows the transitional window of SGD worker runtimes, where the first 75 SGD iterations experience this slowdown before returning to normal for the remaining 125 iterations. In the case of 25% slow workers running 2× slower, naive synchronous SGD decreases throughput by 50%. BDSGD, however, is able to correctly ignore all 40 slow workers (Fig. 3.3a), leading to near-optimal throughput despite the 2× slowdown in SGD iteration time.

Figure 3.3b shows that our method achieves the fastest convergence to the lowest loss among comparison methods performing synchronous SGD. Hogwild outperforms our approach in wall-clock time, but its convergence is to a higher validation loss, as seen in the tails of the loss curves. Although not shown in subsequent experiments, Hogwild training diverges on larger clusters.

3.4.2 Large Scale Computing

On a large compute cluster, we use 32 68-core CPU nodes of a Cray XC40 supercomputer to compare 2175-worker BDSGD runs against the Chen et. al cutoff and naive methods on training three neural network architectures for CIFAR10 classification: a WideResNet model Zagoruyko and Komodakis [2016] and 16 and 64 layer ResNets. Using increasingly larger networks and batch-sizes allows us to benchmark our speedup in situations called for by the large amount of recent work on training with 10K+ mini-batch sizes and high learning rates, (e.g. : Codreanu et al. [2017], You et al. [2017a,b], Smith et al. [2017]). For generative models of these neural network models on this cluster, we empirically find that training the latent variable model to directly emit sorted runtime order statistics is both faster to train and more accurate. Sampled draws from these distributions are reordered as before to calculate the predicted maximum throughput.

Unlike the 158-worker cluster, jobs on the Cray XC40 are sequestered to dedicated nodes by the scheduler. In Figure 3.4, we compare BDSGD and Chen et. al's fixed cutoff method on validation sets in order to isolate the effect of accurate cutoff prediction on expected iteration throughput and speedup. BDSGD provides the best model of these runtimes, subsequently leading to near optimal throughput.

Figure 3.5 shows the wall-clock training times and throughputs achieved under real workloads for each training method on the same three neural networks. Comparing the production throughput in rows 1 and 2 of Figure 3.5 to the expected throughput in rows 1 and 2 of Figure 3.4, all training methods experience a small drop-off in throughput when run in production due to communication costs and other additional overhead. For these two neural networks, BDSGD is still shown to produce the highest throughput when used during training. For training the WideResNet, Chen et. al's method achieves a 1.2% speedup in wall-clock training time whereas BDSGD is able to calculate 5% more gradients to achieve a 1% speedup in wall-clock training time. Similarly, BDSGD achieves a 7.8% speedup in wall-clock training time while using 3.2% more gradients than Chen et. al's method, which achieves a 9.3% speedup.

In the final row of Figure 3.5, we demonstrate the ability for BDSGD prediction to be robust to the scenario in which performing amortised inference in the generative model exceeds the time it takes for workers to finish their gradient computations. Here, we use a modified variant of DBSGD that fixes the predicted cutoff for ten iterations at a time in order to avoid being bottlenecked at every iteration by the parameter server cutoff prediction. In doing so, we show in the final row of Figure 3.5 that one may still achieve a 7.6% speedup with sparse predictions from the generative model.

All training methods for the three neural network models in Figure 3.5 train to a similar final held-out validation accuracy.

3.4.3 Experimental Details

We train 4 distinct neural network models in our experiments: 2-layer CNN on MNIST classification and ResNet-16, ResNet-64, and WideResNet-16x10 on CIFAR10 classification.

The CNN is trained with a batch size of 10112 and fixed learning rates of 0.64 for BDSGD, fully synchronous SGD, and Elfving SGD runs on a 158-worker cluster.

The CNN is also trained with Hogwild and a learning rate of 0.004 (scaled down from 0.64 by the number of workers).

The remaining WideResNet and ResNets are trained on a 2175-worker cluster uses SGD with 0.9 momentum and a linearly scaled learning rate warmup for 50 iterations from an initial learning rate of 0.001. After the warmup phase, the WideResNet model uses a learning rate of 0.12 decayed by 80% at 200 iterations and 50% more at 350 iterations, ResNet-64 uses a 0.18 learning rate decayed by 80% at 250 iterations and 80% more at 450 iterations, and ResNEt-16 uses a 0.06 learning rate with no decay. The models are trained to 400 iterations, 600 iterations, and 1500 iterations, respectively.

Each of these 4 models is trained with fully observed runtime information from naive synchronous training runs of the neural network on the corresponding cluster. All generative models use a 50-dimensional latent variable with 30-dimensional and 50-dimensional MLP link functions for the transfer and emission functions, respectively. The RNN in the proposal network uses 200 hidden units.

For BDSGD on CNN training, we train the generative model with Adam, using a 0.00004 learning rate, 0.96:0.999 beta1 and beta2 values, gradient clipping at 20.0, weight decay of 0.6, and a learning rate decay of 0.99996. and a mini-batch size of 8. For the 3 BDSGD models on the 2175-cluster, we use Adam with a mini-batch size of 20 and a 0.00002 learning rate. All models are trained with a twenty-timestep lag, i.e. $\ell = 20$.

3.5 Discussion

We have presented a principled approach to achieving higher throughput in synchronous distributed gradient descent. Our primary contributions include describing how a model of worker runtimes can be used to predict order statistics that allow for a near optimal choice of straggler cutoff that maximises gradient computation throughput.

While the focus throughout has been on on vanilla SGD, it should be clear that our method and algorithm can be nearly trivially extended to most optimisers

of choice so long as they are stochastic in their operation on the training set. Most methods for learning deep neural network models today fit this description, including for instance the Adam optimiser [Kingma and Ba, 2014].

We note that our method implicitly assumes that every mini-batch is of the same computational cost in expectation, which may not always be the case. Future work could be to extend the inference network further [Rezende and Mohamed, 2015] or to investigate variable length input in distributed training as in Ergen and Kozat [2017].

We conclude with a remark that this work demonstrates the advantages of amortising inference using the VAE framework in a sequential model. Although there is a large up-front cost in training, we are able to utilise the object for the same compute cluster in a real-time application. The only requirement is that the forward pass of the distributed training architecture is longer than the inference network, which is almost always the case, as only large architectures require distributed training. We next investigate two models which show the advantages of having a more structured model and inference network.

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

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

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

Title of Paper	Bayesian Distributed Stochastic Gradient Descent
Publication Status	□Published ▣ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Teng, M., & Wood, F. (2018). Bayesian distributed stochastic gradient descent. *Advances in Neural Information Processing Systems*, *31*, 6378-6388.

Student Confirmation

Student Name:	Michael Teng		
Contribution to the Paper	Frank Wood came up with the idea to use a model of the cluster in order to improve upon the static-cutoff prior art. I implemented and gathered all the data for training. I came up with the idea to use throughput as a target for our method. I also designed and ran the experiments on the clusters. Frank and I co-wrote about half of the paper each.		
Signature		Date	Jan 11, 2022

Supervisor Confirmation

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

Supervisor name and title:	Dr. Frank Wood, Associate Professor		
Supervisor comments:	I concur with Mr. Teng's assessment of his contributions to this work.		
Signature		Date	11 / Jan / 2022

4

Semi-supervised Sequential Generative Models

Contents

4.1 Introduction 85
4.2 Background 87
4.2.1 Variational and Importance-Weighted Auto-encoders . . 87
4.2.2 Semi-supervised Variational Auto-Encoder 88
4.2.3 Variational Recurrent Neural Network 89
4.3 Caffeinated Wake Sleep 89
4.3.1 Generative Model 90
4.3.2 Inference 91
4.3.3 Optimisation 91
4.4 Experiments 96
4.4.1 Semi-supervised MNIST 96
4.4.2 Handwriting 98
4.4.3 Fly tracking 99
4.4.4 Experimental Details 101
4.5 Discussion 102

4.1 Introduction

In recent years there has been an explosion of interest in deep generative models (DGM), which use neural networks transforming random inputs to learn complex probability distributions. Again, we focus on the variational auto-encoder (VAE)

[Kingma and Welling, 2013a] family of models, where the generative model is learned simultaneously with an associated inference network. This approach has been extended to settings with partially observed discrete variables [Kingma et al., 2014] and sequential data [Chung et al., 2015] but so far little work has been done on combining the two. In this work we address this gap.

In many scenarios it is natural to assign discrete labels to specific intervals of time-series data and try to infer these labels from observation. For example, sequences of stock prices can be identified as periods of bear or bull markets, fragments of home CCTV footage can be classified as burglaries or other events, and heart rate signals can be used to classify cardiac health. In such applications it is of interest to classify the sequences of data, generate it conditionally on a sequence of labels, or both. Existing approaches either require full supervision, and therefore are not able to leverage large amounts of unlabelled data, or are restricted to classification and cannot generate new data [Chen et al., 2013, Wei and Keogh, 2006].

Additionally, the standard approach to semi-supervised learning for non-temporal tasks is to have separate unsupervised and supervised loss terms, which are weighted to reflect the overall supervision rate in the dataset. Though this class of approaches is extendable to the time-series setting, it is unclear how to do so when there is an uneven contribution of partial labels to the supervised term, as a result of varying supervision rates in each training example. Because of this, the state-of-the-art approaches to semi-supervised learning, including Virtual Adversarial Training [Miyato et al., 2018], Mean Teacher [Tarvainen and Valpola, 2017], and entropy minimization [Grandvalet and Bengio, 2005], cannot be trivially applied to time-series models.

In this work we derive two novel objectives for training DGMs with discrete latent variables in a semi-supervised fashion. They are both based on the reweighted wake-sleep (RWS) algorithm [Bornschein and Bengio, 2014], which avoids using high variance score function estimators for expectations over the discrete latents. The first one, which we call semi-supervised wake-sleep (SSWS), is obtained by

extending RWS with a supervised classification term. While it performs well when labels available in the training set are regularly distributed, it suffers from optimisation problems when they are not, whsich can be unavoidable in real world datasets. To overcome this problem, we introduce a new approach which performs wake-sleep style optimisation using a single objective that incorporates both supervised and unsupervised terms for every gradient update. We call this approach caffeinated wake-sleep (CWS).

Although we are explicitly targeting the sequential setting, our objectives can also be useful in non-sequential settings, especially when the discrete latent space is too large to be fully enumerated. We evaluate our objectives on MNIST, a handwriting dataset, and a dataset of fruit fly trajectories labeled with behaviour classes. In the rest of the chapter, Section 4.2 reviews the relevant background information, Section 4.3 describes our model and training objective, and Section 4.4 presents the experimental results.

4.2 Background

4.2.1 Variational and Importance-Weighted Auto-encoders

Recall the VAE framework for deep generative modelling where we simultaneously learn a generative model p_θ and an amortised inference network q_ϕ, both parameterised by neural networks. For any given observation x, the learned inference network approximates the intractable posterior distribution $p_\theta(z|x)$ over a latent variable z as $q_\phi(z|x)$. Both networks are simultaneously trained using the ELBO given in Eq 2.94. Again, Burda et al. [2015] propose a tighter bound to the ELBO where instead, we use the IWAE estimator given in Eq 2.102.

In either case, the estimator of the gradient with respect to ϕ is typically high variance if z includes any discrete variables and therefore not suitable for gradient-based optimization. Various authors have proposed to reduce this variance using continuous relaxation [Maddison et al., 2016, Jang et al., 2016] or control-variate

methods [Mnih and Rezende, 2016, Mnih and Gregor, 2014, Tucker et al., 2017, Grathwohl et al., 2017] with varying degrees of success.

An alternative approach is to use the reweighted wake-sleep (RWS) algorithm [Bornschein and Bengio, 2014] which usses separate objectives for θ and ϕ, interleaving the corresponding gradient steps. The target for θ is $\mathcal{L}^K_{\text{IWAE}}(\theta, \phi, x)$, while the target for ϕ is $-\text{KL}(p_\theta||q_\phi)$. In the latter the expectation with respect to p_θ is approximated by importance sampling from q_ϕ. Le et al. [2018] show that using two separate objectives avoids the problems with high variance gradient estimates in the presence of discrete latent variables and avoids learning problems discussed by Rainforth et al. [2018]. Ba et al. [2015] demonstrate RWS as a viable method for time-series modelling by using it to train recurrent attention models.

4.2.2 Semi-supervised Variational Auto-Encoder

In certain situations it is desirable to extend the VAE with an interpretable latent variable y, the canonical example being learning on the MNIST dataset where y is the digit label, z is "style", and x is the image. Kingma et al. [2014] consider the setting where the label y is only scarcely available and the dataset consists of many instances of x and relatively few instances of (x, y), which is a typical semi-supervised learning setting. They choose a factorization $p_\theta(x, y, z) = p_\theta(x|y, z)p_\theta(y)p_\theta(z)$ and $q_\phi(y, z|x) = q_\phi(z|x, y)q_\phi(y|x)$. A naive optimization objective can constructed by summing standard ELBOs for x and (x, y). However, this objective is unsatisfactory, since it does not contain any supervised learning signal for $q_\phi(y|x)$. Letting x_u denote unsupervised input variables and (x_s, y_s) denote supervised observation-latent pairs, Kingma et al. [2014] propose to optimise the corresponding supervised and unsupervised objectives:

$$\mathcal{L}^{\text{s}}(x_s, y_s) = \mathbb{E}_{q_\phi(z|y_s,x_s)}\left[\log\left(\frac{p(x_s, y_s, z)}{q_\phi(z|y_s, x_s)}\right)\right] + \alpha q_\phi(y_s|x_s) \tag{4.1}$$

$$\mathcal{L}^{\text{u}}(x_u) = \mathbb{E}_{q_\phi(z,y|x_u)}\left[\log\left(\frac{p(x_u, y, z)}{q_\phi(z, y|x_u)}\right)\right] \tag{4.2}$$

The additional second term in Equation 4.1 can be optionally scaled by a hyperparameter controlling the relative strength of supervised and unsupervised learning

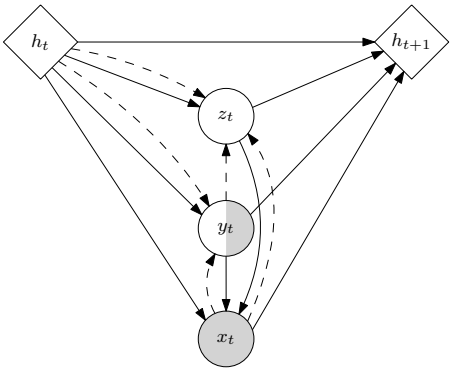


Figure 4.1: Per time step graphical model for the structured VRNN we use. In addition to the continuous latent variable z_t, we include a discrete latent variable y_t corresponding to an interpretable label, for which ground truth is sometimes available in the dataset. Dashed lines indicate the inference inference network, and solid lines the generative model.

signals. A semi-supervised VAE typically achieves performance superior to its unsupervised version.

4.2.3 Variational Recurrent Neural Network

The variational RNN (VRNN) [Chung et al., 2015] is a deep latent generative model that is an extension in the VAE family. It can be viewed as an instantiation of a VAE at each timestep, with the model factorised in time overall. We use a structured VRNN variant, where a discrete latent variable y is also present at each time step, as depicted in Figure 4.1. We also refer the read back to Chapter 2−Section 2.3.4.

4.3 Caffeinated Wake Sleep

Here we derive a general purpose training objective used for semi-supervised generative modelling. As we will show in subsequent experiments, both the objective and training method can be used to effectively train sequential models. As such, we introduce CWS in the context of time-series modelling, but note that it can trivially be used for non-sequential models by considering them to have a single time-step. In all subsequent sections, we denote $ab_t := (a_t, b_t)$, $ab_{\leq t} := (a_{1:t}, b_{1:t})$ and $ab_{<t} := (a_{1:t-1}, b_{1:t-1})$ if $t > 1$ and $\varnothing$ if $t = 1$.

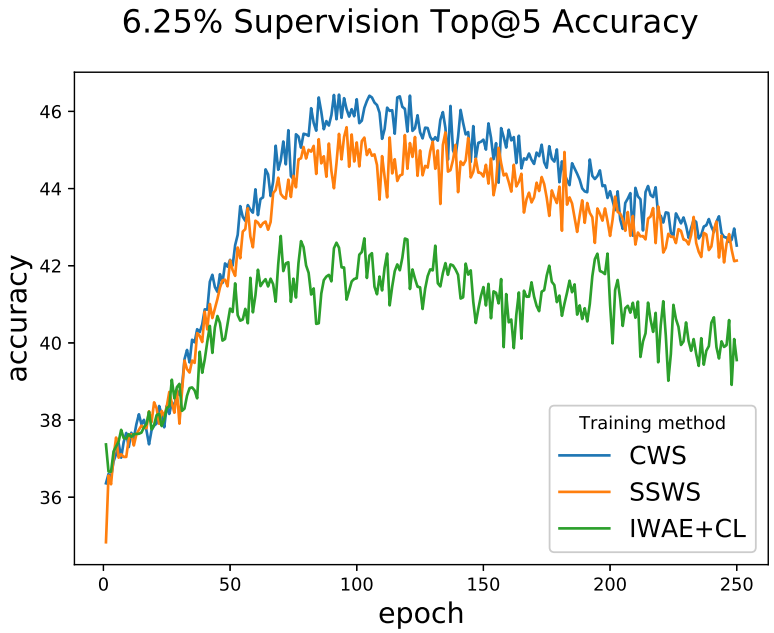


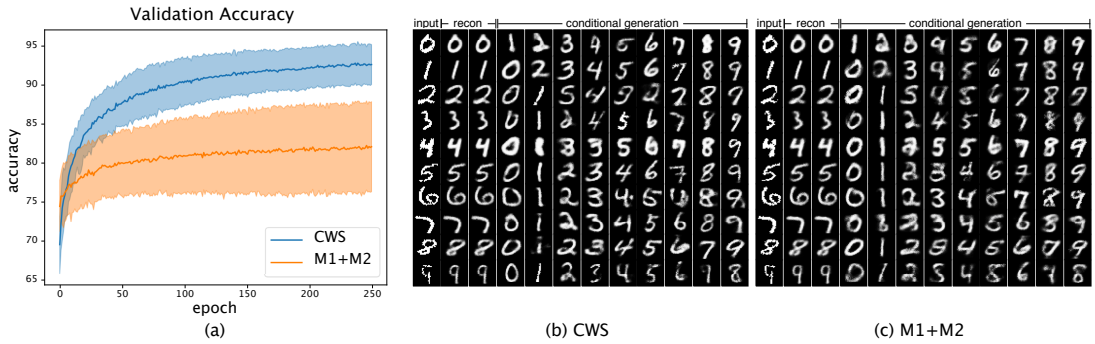
Figure 4.2: Classification accuracy for a 70 class character model given handwriting trajectory output. We train the same model with only 6.25% supervision, using three different training methods. IWAE+CL denotes a baseline we compute by training the generative model and inference with the supervised IWAE bound and the classifier loss, $q_\phi(y_S|x_{\leq T})$, using only the 6.25% of the data for which we have corresponding labels for. We see that at this supervision rate, CWS outperforms SSWS and the baseline.

4.3.1 Generative Model

First, we define a generative model over a sequence of observed variables, $x_{1:T}$, and a corresponding sequence of continuous latent variables, $z_{1:T}$, and a discrete class, $y_{1:T}$.

$$p_\theta(x_{\leq T}, y_{\leq T}, z_{\leq T}) = \prod_{t \leq T} p_\theta(x_t|z_{\leq t}, y_{\leq t}) p_\theta(z_t|x_{<t}, y_{<t}, z_{<t}) p_\theta(y_t|x_{<t}, y_{<t}, z_{<t}) \quad (4.3)$$

If the categorical variable y_t is always treated as a latent variable in the fully unsupervised case, we can define a variational distribution, $q(z_{\leq T}, y_{\leq T}|x_{\leq T})$ and maximize the ELBO with respect to $\{\theta, \phi\}$. Instead, assume we are given $x_{\leq T}$ and a subset $S \subseteq 1:T$ of labeled y_t, denoted $y_S := \{y_t : t \in S\}$. Denote unlabeled $y_U := \{y_t : t \in U\}$ where $U := (1:T) \setminus S$.



(a) (b) CWS (c) M1+M2

Figure 4.3: (a) Validation accuracy of a MNIST classification model trained with 4 different objectives. The objectives CWS and M2+M1 denote two approaches for semi-supervised learning, while M1 and RWS are their unsupervised analogies. As seen, CWS trains to the highest validation accuracy and faster on a held out test set of 10000 examples. (b) Visual reconstructions of MNIST digit in final trained model using CWS (c) Reconstructions using the final trained model by M2+M1 method. In (b-c), the first column denotes the input image, the second denotes reconstruction where z, y are both inferred using $q(y, z|x)$, the third denotes reconstruction where z is inferred given the correct label y, and columns 4-12 denote reconstruction where we infer the style, z, fix it, and vary the y label to conditionally generate.

4.3.2 Inference

Given the generative model defined above and a partially labeled dataset in the y space, our variational distribution is the following:

$$q_\phi(y_U, z_{\leq T}|x_{\leq T}, y_S) = \prod_{t \in U} q_\phi(y_t|x_{\leq T}, y_{<t}, z_{<t}) \prod_{t \leq T} q_\phi(z_t|x_{\leq T}, y_{\leq t}, z_{<t}) \tag{4.4}$$

We will not need to infer given y_S for any sequence, but at any time-step when y_t is available, we use it instead of a sampled y_t from the variational distribution. This is made clear by writing the ELBO as:

$$\begin{aligned} &\log p(x_{\leq T}, y_S) \geq \\ &\mathbb{E}_{q_\phi(y_U, z_{\leq T}|x_{\leq T}, y_S)} \left[\log \left(\prod_{t \in S} \frac{p_\theta(x_t, z_t, y_t,)}{q_\phi(z_t|x_{\leq T}, y_{\leq t}, z_{<t})} \prod_{t \in U} \frac{p_\theta(x_t, z_t, y_t)}{q_\phi(y_t, z_t|x_{\leq T}, z_{<t}, y_{<t})} \right) \right] \end{aligned} \tag{4.5}$$

4.3.3 Optimisation

Having defined the model and inference network, we now specify the optimisation objective. A simple way to derive such an objective would be to mirror the

derivation of Kingma et al. [2014] or Siddharth et al. [2017] in a sequential context. Unfortunately, since their objective is based on the ELBO, we would run into a problem with discrete latent variables y_t. Kingma et al. [2014] deal with this by summing over all possible values of y_t, but the same approach does not work in the sequential setting since the number of possible states of $y_{1:T}$ is exponential in T. Furthermore, their objective requires separate proposal networks, $q(y|x)$ and $q(z|x)$, for labelled and unlabelled data. In the sequential setting, this method would require an exponential number of proposal networks, making it intractable. As we will have different configurations of supervision in terms of which time steps have labels, it is unclear how to directly apply the method given in Kingma et al. [2014]. In the later section, we use a simple approximation of treating each latent variable set as conditionally independent given the immediate previous time-step *only* for the semi-supervision case. This allows us to avoid the exploding number of recognition networks, while maintaining good performance despite the approximation. We note that inference is still performed as a filtering objective, i.e. to take into account all previous time-steps.

Semi-Supervised Wake-Sleep

Before introducing our new objective, we first show how to use the reweighted wake-sleep algorithm [Bornschein and Bengio, 2014] with the semi-supervised objective introduced by Kingma et al. [2014]. Following the recommendation of Le et al. [2018], extending reweighted wake-sleep to include a supervision term can be used to effectively avoid problems with discrete variables. Below we derive the semi-supervised wake-sleep (SSWS) objective within that framework. Both of our objectives will be evaluated by sampling from q_ϕ, so for each $k \in 1:K$ we have $y_t^k \sim q_\phi(y_t|\cdot)$ for all $t \in U$ and $z_t^k \sim q_\phi(z_t|\cdot)$ for all $t \in 1:T$ with a convention $y_t^k = y_t$ for $t \in S$.

For learning the generative model, we maximise the IWAE bound with respect

to parameters θ:

$$\mathcal{L}_{\mathrm{p}}(x_{\leq T}, y_S) := \mathbb{E}_{q_\phi}\left[\log \frac{1}{K}\sum_{k=1}^{K} \frac{p_\theta(y_U^k, z_{\leq T}^k, x_{\leq T}, y_S)}{q_\phi(y_U^k, z_{\leq T}^k | x_{\leq T}, y_S)}\right]$$
$$q_\phi = \prod_k q_\phi(y_U^k, z_{\leq T}^k | x_{\leq T}, y_S) \tag{4.6}$$

This is a lower bound to $\log p_\theta(x_{\leq T}, y_S)$ which is tight when $q_\phi(y_U, z_{\leq T}|x_{\leq T}, y_S) = p_\theta(y_U, z_{\leq T}|x_{\leq T}, y_S)$. Sampling from $q_\phi(y_U, z_{\leq T}|x_{\leq T}, y_S)$ and evaluating its density is simple since it is factorised as in (4.4) where both $q(y_t|\cdot)$ and $q_\phi(z_t|\cdot)$ are given, and all values to the right of the conditioning bar are always available at sampling at time step t (previous $y_{<t}$ are either sampled or given supervision and $z_{<t}$ are sampled).

For learning the inference network parameters, we need to consider the unsupervised and the supervised cases. For unsupervised inference learning, we minimise the expected KL-divergence between the true posterior and the variational posterior given by q_ϕ under the generative model, $p_\theta(x_{\leq T}, y_S)$:

$$\mathbb{E}_{p_\theta(x_{\leq T}, y_S)}[\mathrm{KL}(p_\theta(y_U, z_{\leq T}|x_{\leq T}, y_S) \parallel q_\phi(y_U, z_{\leq T}|x_{\leq T}, y_S))] \tag{4.7}$$
$$= \mathbb{E}_{p_\theta(\cdot)}[\mathbb{E}_{p_\theta(y_U, z_{\leq T}|\cdot)}\left[p_\theta(y_U, z_{\leq T}|\cdot) - \log q_\phi(y_U, z_{\leq T}|\cdot)\right]]$$

Given some $(x_{\leq T}, y_S)$ from the data distribution, $\hat{p}_\theta(x_{\leq T}, y_S)$, the inner expectation is approximated using samples from the inference network, q_ϕ:

$$\mathcal{L}_{\mathrm{q}}(x_{\leq T}, y_S) := \mathbb{E}_{q_\phi}\left[\sum_{k=1}^{K} \bar{w}_k \left(-\log q_\phi(y_U^k, z_{\leq T}^k | x_{\leq T}, y_S)\right)\right], \tag{4.8}$$

where $\bar{w}_k$ is a normalised version of

$$w_k := \frac{p_\theta(y_U^k, z_{\leq T}^k, x_{\leq T}, y_S)}{q_\phi(y_U^k, z_{\leq T}^k | x_{\leq T}, y_S)}, \quad \bar{w}_k = \frac{w_k}{\sum_l w_l} \tag{4.9}$$

and

$$q_\phi = \prod_k q_\phi(y_U^k, z_{\leq T}^k | x_{\leq T}, y_S)$$

Evaluating (4.8) requires evaluating the joint p density given in (4.3), the q density given in (4.4), and sampling again from (4.4).

Finally, we need to include a supervised loss term. There is only supervision signal for some given y_t for some $t \in S$ per data example. As such, we want to

maximise the log-likelihood of the supervised pairs $(y_S, x_{\leq T})$ under the variational distribution, $q_\phi(y_{\leq T}|x_{\leq T})$, marginalised over unsupervised time-steps:

$$\begin{aligned} q_\phi(y_S|x_{\leq T}) &= \int q_\phi(y_{\leq T}, z_{\leq T}|x_{\leq T}) dy_U dz_{\leq T} \\ &= \int q_\phi(y_S|x_{\leq T}, z_{\leq T}, y_U) q_\phi(y_U, z_{\leq T}|x_{\leq T}) dy_U dz_{\leq T} \\ &= \mathbb{E}_{q_\phi(y_U, z_{\leq T}|x_{\leq T})}[q_\phi(y_S|x_{\leq T}, z_{\leq T}, y_U)] \end{aligned} \tag{4.10}$$

We lower bound this with Jensen's inequality and obtain our supervised loss term:

$$\mathcal{L}_{\mathrm{s}}(y_S) := \mathbb{E}_{q_\phi(y_U, z_{\leq T}|x_{\leq T})}[\log q_\phi(y_S|x_{\leq T}, z_{\leq T}, y_U)] \tag{4.11}$$

For each minibatch the objectives, (4.6), (4.8), and (4.11), are computed using the same set of samples and we perform the relevant gradient steps in ϕ and θ by alternating between the two parameter updates:

$$\theta^* = \theta + \alpha \nabla_\theta \mathcal{L}_{\mathrm{p}}(x_{\leq T}, y_S) \tag{4.12}$$

$$\phi^* = \phi - \alpha(\nabla_\phi \mathcal{L}_{\mathrm{q}}(x_{\leq T}, y_S) + \nabla_\phi \mathcal{L}_{\mathrm{s}}(y_S)) \tag{4.13}$$

Caffeinated Wake-Sleep

Although the SSWS approach can be used to train time-series models, in practice there will be a bias introduced by unevenly supervised sequences when performing mini-batch SGD. As mentioned earlier, sequences of observations may contain variable amounts of supervision, the most extreme example being datasets containing both sequences with fully observed and fully unobserved labels. Because of this, the magnitude of the supervised and unsupervised terms in the q_ϕ loss will vary per data sequence, and this would need to be corrected for when averaging over the mini-bath. Furthermore, the need to maintain two separate terms in inference network learning is cumbersome.

Therefore, we obviate the need to maintain two different supervision terms by minimising the expected KL-divergence between the true posterior and the *full*

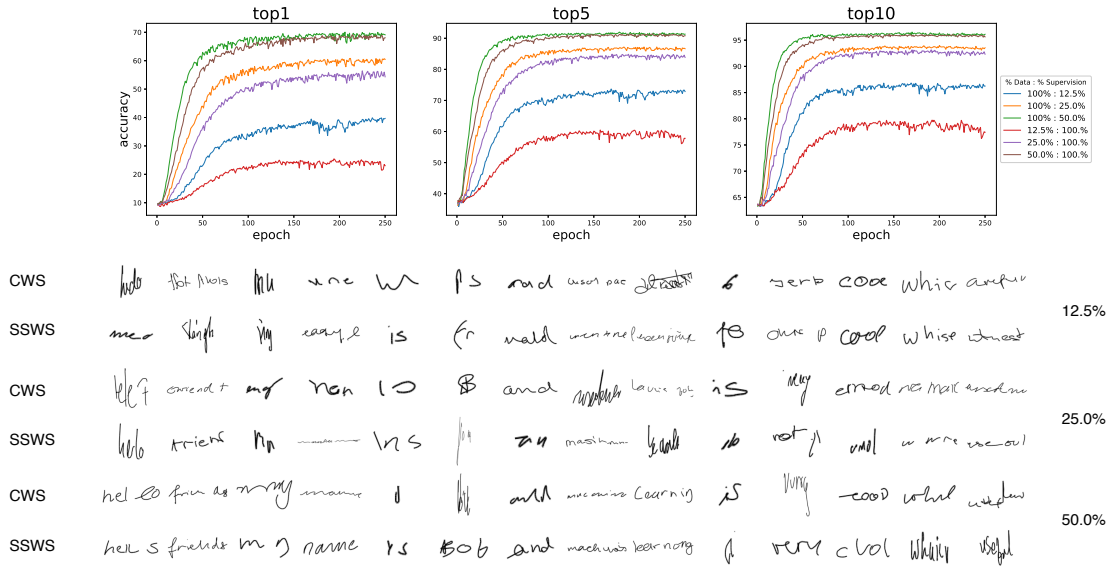


Figure 4.4: We plot the top1, top5, and top10 classification accuracies for a 70-class character classification task comparing models trained with a semi-supervision using CWS against models trained with full supervision but on a corresponding fraction of the original dataset. Below the plots, we show visual reconstructions of sample trajectories of handwriting. Each pair of rows is trained with a supervision rate (right) and a method (left). In these samples, CWS and SSWS both give reasonable results despite inferring missing labels during training.

variational posterior under the generative likelihood, $p_\theta(x_{\leq T})$. Our new single gradient estimator for q_ϕ becomes:

$$\begin{aligned}
&\nabla_\phi \mathbb{E}_{p_\theta(x_{\leq T})}[\mathrm{KL}(p_\theta(y_{\leq T}, z_{\leq T}|x_{\leq T}) \parallel q_\phi(y_{\leq T}, z_{\leq T}|x_{\leq T}))] \\
&= \mathbb{E}_{p_\theta(x_{\leq T})}[\mathbb{E}_{p_\theta(y_U, y_S, z_{\leq T}|x_{\leq T})}[-\nabla_\phi \log q_\phi(y_U, y_S, z_{\leq T}|x_{\leq T}))]] \\
&= \mathbb{E}_{p_\theta(x_{\leq T})}[\mathbb{E}_{p_\theta(y_S|x_{\leq T})}[\mathbb{E}_{p_\theta(y_U, z_{\leq T}|x_{\leq T}, y_S)}[-\nabla_\phi \log q_\phi(y_U, y_S, z_{\leq T}|x_{\leq T})]]]
\end{aligned}$$

Given some $x_{\leq T}$ from the data distribution $\hat{p}(x_{\leq T})$, we now sample y_S in the second expectation with the corresponding supervision signal $\hat{p}(y_S|x_{\leq T})$. As before, we estimate the innermost expectation with samples from (4.4), giving us the overall gradient estimator:

$$\mathcal{L}_{\mathrm{q}}^{\mathrm{CWS}}(x_{\leq T}, y_S) := \mathbb{E}_{q_\phi}\left[\sum_{k=1}^{K} \tilde{w}_k \left(-\log q_\phi(y_U^k, y_S, z_{\leq T}^k|x_{\leq T})\right)\right] \tag{4.14}$$

$$w_k^{\mathrm{cws}} := \frac{p_\theta(y_U^k, z_{\leq T}^k, x_{\leq T}, y_S)}{q_\phi(y_U^k, y_S, z_{\leq T}^k|x_{\leq T})}, \quad \tilde{w}_k = \frac{w_k^{\mathrm{cws}}}{\sum_l w_l^{\mathrm{cws}}}$$

Finally, we formally introduce the caffeinated wake-sleep algorithm. Model parameters are learned with the importance-weighted ELBO as before. However, crucially, we use a unified gradient estimator in (4.14) to update ϕ parameters:

$$\theta^* = \theta + \alpha \nabla_\theta \mathcal{L}_{\mathrm{p}}(x_{\leq T}, y_S) \tag{4.15}$$

$$\phi^* = \phi - \alpha \nabla_\phi \mathcal{L}_{\mathrm{q}}^{\mathrm{CWS}}(x_{\leq T}, y_S) \tag{4.16}$$

In comparison to SSWS losses, CWS no longer explicitly maximises log-likelihood of the supervised pairs $(y_S, x_{\leq T})$ under the variational posterior. Instead, it takes the view of treating supervised latent variables as having been sampled with probability 1 during a normal sequential variational inference rollout.

In Figure 4.2, we find that training with CWS results in higher accuracy than SSWS when using small supervision rates on a generative model of handwriting and learning to classify individual strokes to the high-level character they belong to. We further find that both approaches are able to train this model to higher overall accuracy than the baseline, which we will further elaborate in the experimental section.

4.4 Experiments

We evaluate CWS on an variety of tasks within generative modelling by running experiments on three separate datasets, MNIST, IAM On-Line Handwriting Database (IAM-OnDB) [Liwicki and Bunke, 2005], and Fly-vs-Fly [Eyjolfsdottir et al., 2014]. In our experiments, we consider the training accuracy of the inference network in classification, the conditional and unconditional generation of new data, and the uncertainty captured by our generative models.

4.4.1 Semi-supervised MNIST

We start with a toy MNIST experiment, which semi-supervises a discrete latent variable corresponding to digit class. We use the same model and VAE architecture detailed in Kingma et al. [2014] and compare against their training method, denoted as M2+M1 with the supervised weight set to 60. For supervision, we use only 100 labeled digits out of a total dataset of 50000. We briefly note that CWS for

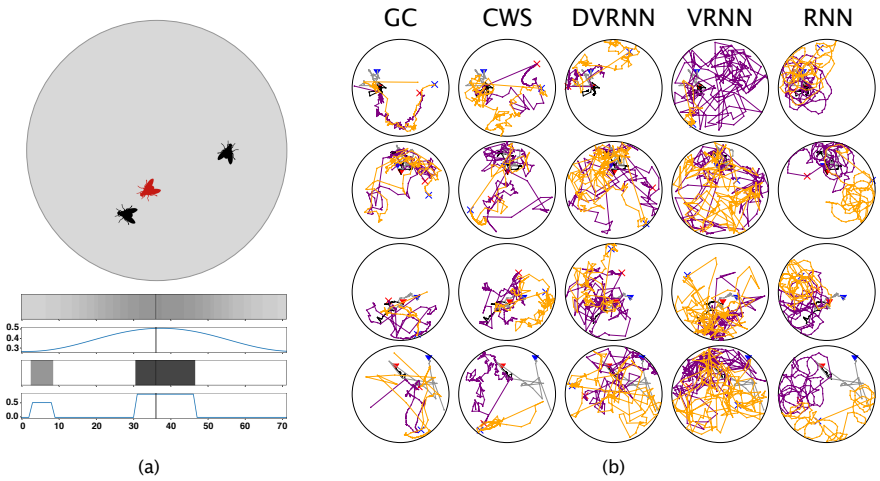


Figure 4.5: (a) Diagram of the red fly's field of view encoding in a petri dish environment containing two other flies. The middle line in the plots denotes the fly's direct line of sight. Each pair of plots indicates the agent's field of view with respect to walls and other flies, respectively. The fly closer to the red agent contributes more mass to the encoding vector. (b) Comparison of each model's continuation tracks for four distinct seed sequences. Each row shows a model's continuation sampled from the generative model compared against the bottom row ground truth continuation. Each row shows one of four seed sequences used to seed the generative models. Within the petri dish, both flies' locomotion tracks are shown. For the first fly, the red arrow indicates the starting position of the seed sequence, the black markers and line indicate the seed tracks, the purple markers and line indicate the sampled continuation tracks, and the red 'x' indicates the final position of the fly after 200 timesteps. For the second fly, these indicators are blue arrow, gray markers, orange markers, and blue 'x', respectively.

the single time-step case is trivial given the objectives defined above and SSWS recovers the M2+M1 objective.

In Figure 4.3a, we compare validation accuracy of digit classification between CWS and M2+M1 trained models. We find that our method both trains faster and better despite using an identical generative model, inference network, learning rate, and optimizer. Furthermore, our approach requires setting one fewer hyper-parameter when compared to M2+M1.

In Figure 4.3b, we conditionally generate samples using our trained models by first inferring the style using $q_\phi(z|\hat{x}, \hat{y})$. Then, fixing the style, we change y and reconstruct using $p_\phi(x|z, y)$. We find that CWS is able to conditionally generate as well as prior art.

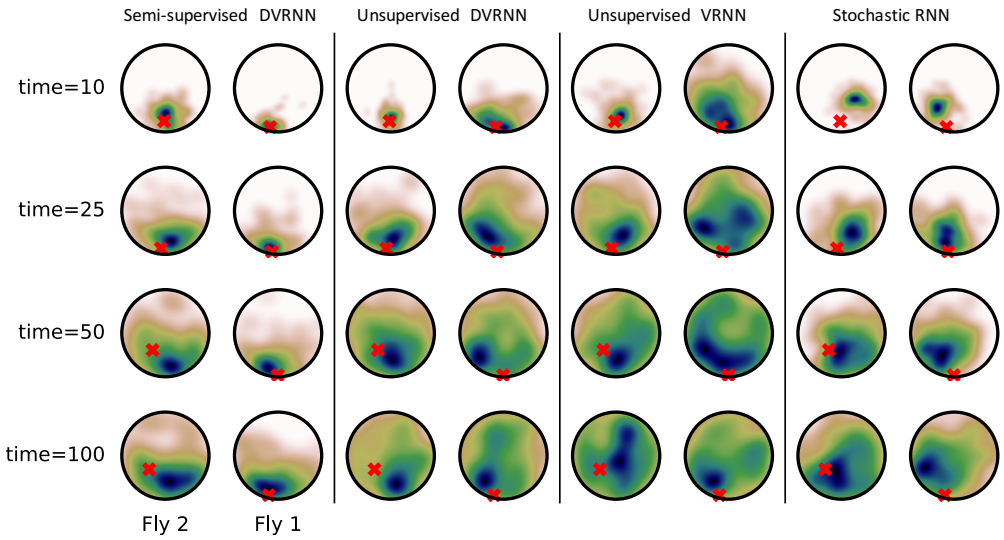


Figure 4.6: Comparison of uncertainty estimates over future trajectories of two flies' interactions in the environment between different models. Each model is seeded with the same ground truth sequence each time and 100 continuations are sampled for 100 timesteps into the future. We show the kernel density estimate of all fly positions at the indicated timestep across the 100 continuations. Each model's estimate is split by fly for clarity (two columns per model indicated by separators), where the red 'x' indicates the true position of the fly at that timestep. The future trajectories across all models show greater uncertainty about the fly's position as the model evolves in time. We find the model trained with CWS is the least noisy as it evolves, still captures the actual position within a high probability region, and has the most mass close to the true position of fly 1 at time 100 among the 4 models.

4.4.2 Handwriting

Next, we run an experiment on the IAM-OnDB dataset. The generative model is over $x_t := \{c_t^i, c_t^j, \text{pen}_t, \text{eoc}_t\}$, where c_i, c_j denotes a single time-step vectorised stroke of the pen and the pen_t and eoc_t denote pen-up status and end-of-character binary values. The latent space in this model is comprised of continuous style variable, z_t, and a discrete character label corresponding to a sequence of observed strokes that make up a valid character, y_t. For the IAM-OnDB data, the valid alphabet is comprised of 70 different characters: upper and lower case letters, digits, and special characters. The large latent space makes this a very challenging time-series problem.

The VRNN architecture we use is taken from the DeepWriting model introduced by Aksan et al. [2018]. At a high level, we use a VRNN over $x_{\leq T}, z_{\leq T}$, and a BiLSTM network for $q_\phi(y_{\leq T}|x_{\leq T})$. Because the VRNN can be optimized solely using reparameterisation, we train $q_\phi(z_{\leq T}|y_{\leq T}, x_{\leq T})$ using IWAE and $q_\phi(y_{\leq T}|x_{\leq T})$ using CWS. This mirrors the optimisation in Aksan et al. [2018], except that now, we can train with semi-supervision. For more detailed experimental setup, we refer the reader to the cited work.

As previously mentioned, techniques for semi-supervised learning do not trivially extend to time-series models. Instead, we compare CWS against SSWS, the latter of which can be viewed as the M2+M1 objective for sequential models. In all experiments, we use a training set of 26560 sequences and a validation set of 512 sequences, all of length 200.

In Figure 4.4, we present results of training with a fixed architecture, varying only the training method. First, we find that validation accuracy of the classifier trained with CWS using additional unlabelled data greatly outperforms the IWAE baseline using the same amount of partial labels from the dataset but without the additional data. This baseline is generated by using the optimisation technique used from Aksan et al. [2018] on a corresponding fraction of the dataset. Next, we show reconstructed samples from trained models where we force the network to attempt to generate the phrase, "hello friends my name is Bob and machine learning is very cool while useful". We find that even at 12.5% supervision, reconstructed samples resemble the target sentence using either of our methods.

4.4.3 Fly tracking

In our final experiment, we run CWS on a dataset of two male fruit flies interacting in a petri dish Eyjolfsdottir et al. [2014]. We use a VRNN which attempts to mirror biological plausibility by modelling movement, x_t, as velocities of body position and wing/thoracic configurations Cueva and Wei [2018]. We also inject as input to the RNN at each time-step, the visual encoding for a given fly shown

in Figure 4.5a Clandinin and Giocomo [2015] along with knowledge of its own state Cueva and Wei [2018].

The dataset uses high level actions, which we model as a 6-valued discrete variable, y_t, corresponding to possible semantic labels: "lunge", "charge", "tussle", "wing threat", "hold", and "unknown". These annotations are provided by human experts at time-steps dispersed throughout the dataset without a clear pattern or regularity. In all experiments we compare four models: the RNN used by Eyjolfsdottir et al. [2016] outputting a probability vector over discretized action space at each time step, the standard VRNN without y and with continuous valued z, the discrete VRNN (DVRNN) with y_t trained unsupervised, and the same DVRNN trained with CWS. While CWS was able to train this model, we ran into optimization difficulties with the same model but training with a separate supervision term. We fine-tune and report results of the best performing model during training: CWS-DVRNN with 30-dimensional z and 6-valued y, RWS-DVRNN with 50-dimensional z and 10-valued y, and VRNN with 120-dimension z.

In Figure 4.5b, we condition the model on an initial sequence of actions, then sample a continuation and visually inspect how it compares with the true continuation that was not shown to the model. We find that under the RNN model, the flies tend to move in circular patterns with relatively constant velocity. In contrast, real flies tend to alternate between fast and slow movements, changing their directions much more abruptly. All three variants of our VRNN model qualitatively recover this behaviour, however the continuous VRNN also tends to behave too erratically. In the DVRNN models, we were not able to identify any other clear visual artefacts in the generated trajectories that disagrees with the real data, although DVRNN without semi-supervision appears to generate trajectories where flies move too much.

In Figure 4.6, we investigate the quality of uncertainty estimates produced by various models. For this purpose we again seed the model with an initial sequence of actions, then observe how the probability mass over the flies' projected future positions evolves over time and compare it with the actual positions in the dataset.

Table 4.1: KDE of ground truth position under model

	KDE $\log p$	
model	**fly 1**	**fly 2**
CWS+DVRNN	**−780.1**	**−851.5**
RWS+DVRNN	−901.8	−853.9
VRNN	−962.0	−879.8
RNN	−931.0	−921.2

Figure 4.6 visualises the results and Table 4.1 provides the log likelihood of each fly's true position under this density estimate. Again, we find that CWS trained DVRNN performs best, followed by the unsupervised DVRNN, the continuous VRNN, and the RNN.

4.4.4 Experimental Details

Data Cleaning For the most part, the data is exhaustive, but a small percentage of tracking data is missing, which we fill in with a linear interpolation between the previous and next known frames. For example, if there is data missing for the flies position for 2 frames, we assume it walked in a straight line from its previous known location to the next known location. When the missing data is rotational, we interpolate with the assumption that the fly rotated along the shorter of the two possible arcs to the known orientation.

Exact measurements are given for $y_{t,f}$ and $\{\hat{o}_{t,f}\hat{w}^{ll}_{t,f}, \hat{w}^{la}_{t,f}, \hat{w}^{rl}_{t,f}, \hat{w}^{la}_{t,f}, \hat{b}^{maj}_{t,f}, \hat{b}^{min}_{t,f}\}$, but $s^{fly}_{t,f}$ and $s^{fly}_{t,f}$ are manually calculated. Sensory data for the other fly is approximated with the opposite fly being an exact circle with radius 12.95. In reality, these flies are ellipses.

Training Each VRNN model is trained with discrete and continuous latents using reweighted wake-sleep. The wake loss of θ (update of θ using IWAE) is annealed to a constant multiplier of 1.0 on the regularisation term (KL between $q(z|y)$ and $p(z)$) over 1000 steps. We use Adam with learning rate of 2e-5 with 25 particles

for training and train VRNN-DST model to 16000 iterations and VRNN-II to 12000. For the RNN baseline, we use RMSprop with 0.5 weight decay and train to 1500 iterations with a learning rate of 2e-5. All models are trained with a batch size of 32 (or 16 per fly).

The dataset we use for training and testing consists of 200 length sequences that comprise at least 25 percent well defined behaviour. that is, our dataset is labeled with actions consisting of lunges, wing threats, charges, holds, and tussles. Erratic behaviour is also labeled as well as idle behaviour. We define interesting actions as not idle nor erratic. the training dataset consists of 3292 sequences pairs or 6584 total sequences of length 200 behaviour. Of the 1,316,800 frames, 717188 are idle, 3567 are erratic, 83990 are tussles, 7946 are holds, 337 are charges, 485335 are wing threats, and finally, 24924 are lunges. (the labeled behaviour sums to 1,323,287 because there are overlapping frames, i.e. flies may charge or hold as part of tussling)

Model architecture The neural network architecture for the RNN uses two parallel GRU cells with hidden dimension of 150. Input to the first is 160 dimensional. The output from the second GRU is used as input to a 2-layer MLP with ReLU activations, to produce the final action output which is 9-dimensional.

For VRNN models, we use 10-dimensional one-hot categorical variables and 50-dimensional Gaussians as latents and 150 dimensional hidden units with a 2-Layer GRU. All link functions in the above graphical models are n-Layer MLPs with ReLU activations. The inference network link functions work similarly, and take as input variables and hidden outputs from the generative model. When multiple link functions enter a single node, we concatenate all the incoming vectors.

4.5 Discussion

We have introduced a new method for semi-supervised learning in deep generative time-series models and we have shown that it achieves better performance than unsupervised learning and fully supervised learning with a fraction of the data.

In future work, we are planning to extend CWS to a more general class of semi-supervised models, taking inspiration from the ideas of Siddharth et al. [2017].

We have now introduced two more examples of sequential generative modelling and inference. As shown, the ideas of this framework can be applied to model biologically plausible behaviour better than pure RNN-based methods, while giving better uncertainty estimates as well. The added discrete latent variable also gives us a way to enable structure models, where VI is able to still work under partial supervision. In the following Chapters, we move on to the domain of reinforcement learning.

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

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

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

Title of Paper	Semi-supervised Sequential Generative Models
Publication Status	□Published ▣ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Teng, M., Le, T. A., Scibior, A., & Wood, F. (2020, August). Semi-supervised Sequential Generative Models. In *Conference on Uncertainty in Artificial Intelligence* (pp. 649-658). PMLR.

Student Confirmation

Student Name:	Michael Teng		
Contribution to the Paper	Frank Wood and Adam Scibior had the idea to target semi-supervised learning. I derived the SSWS and CWS objectives with Adam Scibior, while Tuan Anh Le helped with implementation for fly experiments. The paper is written mostly by me. I also implemented the handwriting and MNIST experiments.		
Signature		Date	Jan 11, 2022

Supervisor Confirmation

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

Supervisor name and title:	Dr. Frank Wood, Associate Professor		
Supervisor comments:	I concur with Mr. Teng's assessment of his contributions to this work.		
Signature		Date	11 / Jan / 2022

5

KL-Regularised Reinforcement Learning as Inference

Contents

5.1 Introduction 105
5.2 Background 107
5.2.1 Markov Decision Processes 108
5.2.2 Soft Actor Critic 108
5.2.3 Reinforcement Learning as Inference 110
5.3 Policy Gradient Estimators Via The Reverse KL Divergence 112
5.4 Reweighted Policy Gradient Estimators Via the Forward KL 115
5.5 Experiments 118
5.6 Related Work 119
5.7 Discussion 121

5.1 Introduction

The recent success of deep reinforcement learning (RL) across a wide range of applications is largely attributable to advances in actor-critic algorithms [Haarnoja et al., 2018c, Fujimoto et al., 2018, Gu et al., 2016, Lillicrap et al., 2015, Schulman et al., 2015b, Mnih et al., 2016, Schulman et al., 2017, Kurutach et al., 2018]. While versatile, many of these algorithms are difficult to implement and require careful

hyper-parameter tuning [Engstrom et al., 2019]. Additionally, the most performant implementations of these algorithms rely on heuristics to stabilise learning that are not explicitly referenced within the algorithm definition, or even informed by the relevant theory [Engstrom et al., 2019, Tucker et al., 2018]. In some cases these implementation details become so crucial to performance that reproducing the original results becomes almost impossible [Islam et al., 2017, Duan et al., 2016, Henderson et al., 2017].

In light of these issues, some researchers refocused on algorithms informed by probabilistic methods, most notably a popular class of algorithms derived within the 'RL as Inference' (RLAI) framework [Neumann et al., 2011, Levine, 2018, Le et al., 2018, Ziebart, 2010, O'Donoghue et al., 2020, Fellows et al., 2019, Abdolmaleki et al., 2018, Song et al., 2019, Chow et al., Piché et al., 2019]. Under this framework, policy learning in a Markov decision process can be framed as an approximate posterior inference task, which has several key benefits over alternative approaches. First, policies trained using this approach can produce agents that transfer skills between tasks or even entire environments [Fellows et al., 2018]. Second, RLAI couches the task of model learning (i.e. learning the dynamics of the environment) within the larger literature of generative modelling and can take advantage of existing tools and techniques [Clavera et al., 2018, Abdolmaleki et al., 2018, Igl et al., 2018, Chow et al.], allowing us to reason in the even wider class of differentiable simulation [Metz et al., 2021, Freeman et al., 2021, Ainsworth et al., 2021]. Lastly, Kullback-Leibler (KL) regularisation introduces the notion of "default policies", which induce different behaviours in the learned policy that are useful in solving certain environments and improving exploration [Galashov et al., 2019, Haarnoja et al., 2018b, 2017].

Practically speaking, RLAI provides insights into how to construct various policy losses which interact more intelligently with modern computation graph software [Paszke et al., 2019, Abadi et al., 2015, Bradbury et al., 2018]. In framing RL as an approximate inference task, we force this consideration in deciding what divergence we wish to minimise, and how gradients of that expectation will be

estimated. This is doubly important in settings where the the simulator itself can be fully or partially differentiable [Bradbury et al., 2018, Bezanson et al., 2012]. While we will not consider how gradient estimates of the forward and backwards messages can be better estimated in general, we will look at how to provide learning signal to policies in a general setting (e.g. for anonymous policy distributions given only log-probability of samples). We argue that closer consideration of this class of gradient estimators will allow for more composable reinforcement learning software, which in turn would allow for more stable policy learning over all model or distribution classes.

This chapter focuses on the following: (1) gradient estimators of the forward and reverse KL divergences with respect to the policy parameters, (2) the practical implementations of popular algorithms based on these estimators, and (3) an empirical comparison of the two resulting policy losses. We begin with relevant background, then derive policy gradient estimators based on divergence minimisation, and finally show simple experiments to demonstrate how the performance of each objective differs under modified hyper-parameters. We conclude with a discussion on the utility of each policy loss, and consider future directions.

5.2 Background

There are three important ideas required to understand the differences between gradient estimators and target objectives in RLAI. The first is the probabilistic structure of a Markov Decision process, which will allow us to construct a corresponding inference task and model dependencies within various gradient estimators. For background on this, we refer the reader to Sections 2.4.1. The second is the is how actor-critic algorithms amortise many of the previous costly sampling operations required in classical inference based methods and on-policy algorithms. Lastly, we show how one can reframe the RLAI task as divergence minimisation.

5.2.1 Markov Decision Processes

Recall in Sections 2.2.3&2.4.1, where we define a Markov decision process (MDP), the generative (forward) model of the MDP in a RLAI setting, and the variational posterior used for policy learning. More generally, we can extend this notion to infinite horizon problems through the use of a discount factor which signifies the non-zero probability of termination after every time-step. In this setting it is common to arrive at a policy through the solution to a fixed point equation [Bertsekas, 2019, Sutton and Barto, 2018], which can be rewritten as the following optimization problem with respect to the policy parameters ϕ:

$$\phi^* = \max_{\phi \in \Phi} \mathbb{E}_{s \sim d^{\pi_\phi}, a \sim \pi_\phi}[Q^{\pi_\phi}(a, s)], \quad \text{where} \quad d^{\pi_\phi}(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t q_{\pi_\phi}(s_t = s), \tag{5.1}$$

$$\text{and} \quad Q^{\pi_\phi}(a, s) = \mathbb{E}_{p(s'|s,a)}\left[r(s, a) + \gamma \mathbb{E}_{\pi_\phi(a'|s')}\left[Q^{\pi_\phi}(a', s')\right]\right]. \tag{5.2}$$

Here, the distribution over states, $d^{\pi_\phi}(s)$, is referred to as the *state occupancy*. Methods which simultaneously learn the policy π_ϕ and critic Q^{π_ϕ} are generally called actor-critic algorithms [Mnih et al., 2016, Haarnoja et al., 2018c, Schulman et al., 2017, 2015a], and represent the most popular class of reinforcement learning algorithms currently available.

5.2.2 Soft Actor Critic

One popular variant of this framework is known as soft actor-critic (SAC). This algorithm is similar to most off-policy actor critic algorithms, in that it learns to estimate the expected reward ahead given the state and action (the Q-function), using the bellman error:

$$F(\theta) = \mathop{\mathbb{E}}_{(a_t, s_t, s_{t+1}) \sim \mathcal{D}}\left[\left(Q_\theta^{\pi_\phi}(a_t, s_t) - \left(\hat{r}_t + \mathop{\mathbb{E}}_{a_{t+1} \sim \pi_\phi(a_{t+1}|s_{t+1})}\left[\gamma \bar{Q}_\theta^{\pi_\phi}(a_{t+1}, s_{t+1})\right]\right)\right)^2\right] \tag{5.3}$$

Here, D defines the set of examples gathered from the environment under a behavioural policy, or a sequence of behavioural policies. The primary distinction between SAC and other AC algorithms [Mnih et al., 2016, Fujimoto et al., 2018]

however, is that r_t is replaced with a surrogate $\hat{r}_t = r_t - \mathcal{H}_t$, where $\mathcal{H}_t$ indicates the policy entropy at state s_t. Additionally in this setting, the bar over Q indicates a target network, generally defined by the use of parameter averaging [Mnih et al., 2013, Wang et al., 2016], a model ensemble, or both [Hessel et al., 2018]. Using this Q function estimate, we can iteratively update our policy to match the so called "softmax-optimal" policy [Haarnoja et al., 2018b,c, Levine, 2018]. This approach defines its policy objective based on a divergence between the current policy and a normalised target Q function:

$$J_\theta(\phi) = \mathop{\mathbb{E}}_{s_t \sim \mathcal{D}} \left[\mathop{\mathbb{E}}_{a_t \sim \pi_\phi(a_t|s_t)} \left[Q_\theta^{\pi_\phi}(a_t, s_t) + \alpha \log \left(\frac{\pi_0(a_t|s_t)}{\pi_\phi(a_t|s_t)} \right) \right] \right], \tag{5.4}$$

Here the gradient is evaluated using the reparametrisation trick [Le et al., 2017b] in cases where the distribution over actions is reparameterisable (e.g. normal, gamma ect.), and with REINFORCE [Williams, 1992] otherwise. A description of SAC is included in Algorithm 4. To ensure a fair comparison of the two divergences described within this chapter, we use SAC as the base algorithm, and modify only the policy update with the corresponding gradient estimator under consideration (coloured blue in Algorithm 4).

Algorithm 4 Soft Actor Critic with Entropy Tuning

Initialize Q-functions, $Q_{\theta_1}(s,a)$, $Q_{\theta_2}(s,a)$ and policy weights $\pi_\phi(a_t|s_t)$
Initialize target networks $Q_{\bar{\theta_1}}(s,a)$, $Q_{\bar{\theta_2}}(s,a)$
Initialize replay buffer D
for each iteration **do**
  for each environment step **do**
    Sample action form policy π
    store transition into replay buffer
  end for
  for each gradient step **do**
    Update Q-function using $J_Q(\theta_i)$ for $i \in \{1,2\}$ $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$
    Update policy weights using $J_\pi(\phi)$ $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    Adjust temperature using $J(\alpha)$ $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$
    Update target network weights using Polyak averaging $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$
  end for
end for

5.2.3 Reinforcement Learning as Inference

Recall that RL can be posed as a posterior inference problem over the set of trajectories conditioned on how well we would like the agent to do, as we covered in Chapter 2−Section 2.4. This breaks down into a set of latent random variables (the trajectories), and a set of observed random variables describing how well the agent is performing in the classical sense (e.g. the reward). We will refer to the observed random variables as "optimality" variables, denoted O_t. These optimality variables indicate if a given state action pair was optimal or not, and are assumed to be Bernoulli distributed. We say that a state action was optimal by indicating $O_t = 1$, while we say that a state action was sub-optimal by indicating $O_t = 0$. In RLAI, O_t is distributed as,

$$p(O_t = 0|a_t, s_t) = 1 - \frac{1}{Z_0} \exp r(a_t, s_t), \quad \text{and} \quad p(O_t = 1|a_t, s_t) = \frac{1}{Z_0} \exp r(a_t, s_t). \tag{5.5}$$

In the later sections, we will perform approximate inference in this model *conditioned* on observed rewards being optimal ($O_t = 1, \ \forall t \in 1...T$). As such, we assume the agent is optimally interacting with the environment during trajectory rollouts. However, this assumption does not carry over to the real-world, where agents are often behaving suboptimality while learning. Although we do not consider the case of suboptimality, the design of O_t as an observed Bernoulli random variable in the graphical model is nonetheless interesting for future work in RL. First, evidence from neuroscience suggests that humans may still be performing Bayesian inference for planning and decision making, despite acting suboptimally at times (i.e. overly habitual behaviour; Schwartenbeck et al. [2015]). In this work, the authors suggest that suboptimality can be seen as suboptimality of an individual's generative model of the world and the priors therein. Then, planning in this model can still be done via "control as inference". Reddy et al. [2018] explore this idea using simulation studies, and subsequently propose a model learning framework that can recover suboptimal behaviour and be leveraged by inverse RL algorithms.

Finally, although beyond the scope of this work, we give some intuition for the design choice of O_t. Conceptually, our goal is to frame RL as inference in a principled probabilistic model. In practice, we will be dealing with an energy based model, in that the normalisation constant remains unknown and the exponentiated reward given above is what we use for the energy function. However, this does not invalidate the choice of optimality variable, O_t, given the above discussion above and for future work that builds on this probabilistic modelling perspective. For example, we can already give the intuition for cases where optimal behaviour is intrinsically high variance. In the RLAI framework, high variance optimality would result in highly diffuse policies, due to each action being equally likely to give optimal control. In the toy example where $p(O_t = 1|a_t, s_t) = 0.5 \;\; \forall t \in 1...T$, the optimal policy is trivially uniform random sampling.

With optimality defined, we can consider the probability that a given trajectory, a sequence of state action pairs sampled from a policy interacting with the environment, is itself optimal. For finite horizon problems, this joint distribution over both latent and observed random variables is defined,

$$p(\tau, O_{1:T-1}) = p(s_1) \prod_{t=1}^{T-1} \Big[p(s_{t+1}|s_t, a_t) \pi_0(a_t|s_t) p(O_t|\tau_t) \Big], \tag{5.6}$$

where $\pi_0(a_t|s_t)$ denotes the 'default' policy or prior over actions given states. For clarity, probabilistic graphical models with different reward and state assumptions are given in Figure 5.1. However, in control, we do not just want a metric on how likely a given trajectory is to be optimal, we want to produce a policy which itself induces trajectories that are optimal: $\pi^*(a_t|s_t) = \pi(a_t|s_t, o_t, ..., o_{T-1})$. Unfortunately, arriving at an exact $\pi^*(a_t|s_t)$ is intractable in all but the simplest cases. Whats more, both our graphical model and our resulting joint distribution are defined in terms of entire trajectories. This means that not only must we perform a posterior inference task (e.g. determine what the distribution over states and actions given optimality), but we must also find a way recover the marginal distribution over actions given the state and optimality for any state which we might see through environment interaction. We address the first issue (intractability

of the posterior inference problem) by using variational inference to produce an approximate posterior efficiently. We will then show through manipulation of the expected gradient and structure of the graphical model, we can define an objective who's optimum characterises the same policy as is defined by $\pi(a_t|s_t, o_t, ..., o_{T-1})$.

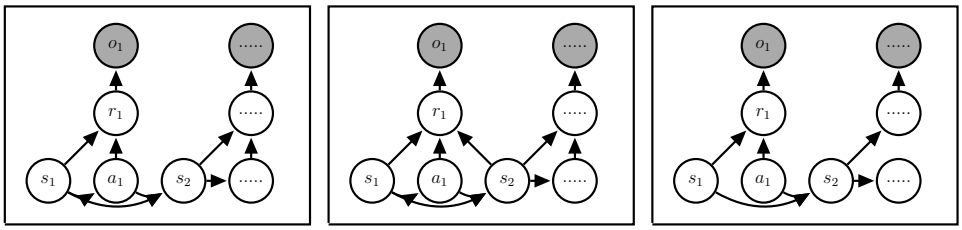


Figure 5.1: Three potential probabilistic graphical models associated with RLAI. In the graphs, shaded nodes indicate observed random variables, while unobserved or *latent* random variables are transparent. Note that on the left, as is usually the case the reward is not dependent on the resulting transition. The centre graphical model gives an example of this, and is often not used as it tends to exacerbate over-optimistic behaviour in the policy. The case on the right is given in the "maximum entropy" setting where the default policy is assumed to be uniform and independent of the state.

5.3 Policy Gradient Estimators Via The Reverse KL Divergence

This section contains derivations relevant to the reverse KL applied to RLAI. More specifically, this section gives information on both the un-marginalised and marginalised objective and its associated gradient estimator. This section will rely on classic tools found in variational inference, and makes frequent use of the REINFORCE trick [Williams, 1992], and the score function trick [Sutton and Barto, 2011].

The Un-marginalised Reverse KL Objective

For the following we assume that $\tau_t := (a_t, s_t, s_{t+1}, r_t)$, and $(s_{0:T}, a_{0:T-1}, r_{0:T-1}) := \tau$. We begin by looking at the objective derived by integrating over all time-steps simultaneously and differentiating. To derive this objective we start with the

functional form of the reverse KL divergence,

$$KL(q_\phi(\tau)||p(\tau|O)) = \int q_\phi(\tau) \log \left(\frac{q_\phi(\tau)}{p(\tau|O)} \right) d\tau$$
$$= \int q_\phi(\tau) \log \left(\frac{q_\phi(\tau)}{p(\tau, O)} \right) d\tau - \int q_\phi(\tau) \log p(O) d\tau = \int q_\phi(\tau) \log \left(\frac{q_\phi(\tau)}{p(\tau, O)} \right) d\tau - \log p(O)$$

Next we can look at the log term and note, that it can be broken up into each time step, and additionally that the dynamics between the the joint and approximate distribution cancel out. We can then simplify the integral to exclude terms which are constant with respect to the policy parameters. This allows us to convert a difficult to compute objective which included the optimal policy, into an objective we can estimate through Monte-Carlo sampling and interaction with the environment:

$$\min_\phi \left[\int_\tau q_\phi(\tau) \log \left(\frac{q_\phi(\tau)}{p(\tau, O)} \right) d\tau \right] = \min_\phi - \mathop{\mathbb{E}}_{\tau \sim q_\phi} \left[\sum_{t=0}^{T-1} r(s_t, a_t) - \log \left(\frac{\pi_0(a_t|s_t)}{\pi_\phi(a_t|s_t)} \right) \right].$$

A Marginalised Reverse KL Gradient Estimator

Unfortunately, the naive gradient estimator using REINFORCE on this loss will have variance which grows with the time-horizon T. However by considering its gradient estimator we can infer an objective whose minimum is the same as the one above up to a constant factor. In a similar fashion to the policy gradient theorem [Sutton and Barto, 2018] we consider the expected gradient of our loss and simplify:

$$\nabla_\phi \mathop{\mathbb{E}}_{q(\tau)} \left[\sum_{t=0}^{T-1} r(s_t, a_t) - \log \left(\frac{\pi_\phi(a_{t'}|s_{t'})}{\pi_0(a_{t'}|s_{t'})} \right) \right] = \sum_{t=0}^{T-1} \mathop{\mathbb{E}}_{q_\phi(\tau_{t:T-1})} \left[\hat{\nabla}_\phi \left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) - \log \frac{\pi_\phi(a_{t'}|s_{t'})}{\pi_0(a_{t'}|s_{t'})} \right) \right],$$

where $\hat{\nabla}_\phi = \nabla_\phi \log \pi_\phi(s_t, a_t)$. We start by passing the gradient into the expectation under the assumption that the integrand is bounded over the domain of interest,

$$\nabla_\phi J_{q,p}(\phi) = \int \nabla_\phi \left[\left(\sum_{t=0}^{T-1} r(s_t, a_t) - \log \frac{\pi_\phi(a_t|s_t)}{\pi_0(a_t|s_t)} \right) q_\phi(\tau) \right] d\tau$$
$$= \int \left(\sum_{t=0}^{T-1} \nabla_\phi \log \pi_\phi(a_t|s_t) \right) \left(\sum_{t'=0}^{T-1} r(s_{t'}, a_{t'}) - \log \frac{\pi_\phi(a_{t'}|s_{t'})}{\pi_0(a_{t'}|s_{t'})} - 1 \right) q_\phi(\tau) d\tau$$

Note that s_t and s'_t correspond to the *same* random variable (as do a_t and a'_t for all t). The primes are introduced only to index the double sum below correctly.

Here we use the score function trick to transform the integral of a gradient into the expectation over a gradient. We then combine terms and simplify, noting that the gradient of the dynamics is zero. We then convolve these sums and further simplify using d separation of the probabilistic graphical model.

$$\begin{aligned} J_{q,p}(\phi) &= \int \sum_{t=0}^{T-1} \sum_{t'=0}^{T-1} \nabla_\phi \log \pi_\phi(a_t|s_t) \left(r(s_{t'}, a_{t'}) - \log \frac{\pi_\phi(a_{t'}|s_{t'})}{\pi_0(a_{t'}|s_{t'})} - 1 \right) q_\phi(\tau) d\tau, \\ &= \sum_{t=0}^{T-1} \sum_{t'=t}^{T-1} \int \nabla_\phi \log \pi_\phi(a_t|s_t) \left(r(s_{t'}, a_{t'}) - \log \frac{\pi_\phi(a_{t'}|s_{t'})}{\pi_0(a_{t'}|s_{t'})} \right) q_\phi(\tau) d\tau. \end{aligned}$$

Where the last line follows from the Markov property of the graphical model which gives $t' < t \rightarrow r(s_{t'}, a_{t'}) - \log \pi_\phi(a_{t'}|s_{t'}) + \log \pi_0(a_{t'}|s_{t'})$ is independent of $\nabla_\phi \log \pi_\phi(a_t|s_t)$. This means that we can decompose the expectation such that for all $t' < t$,

$$\int \nabla_\phi \log \pi_\phi(a_t|s_t) f(a_{t'}, s_{t'}) q_\phi(\tau) d\tau = \int f(a_{t'}, s_{t'}) \, (0) \, \frac{q_\phi(\tau)}{\pi_\phi(a_t|s_t)} d\tau \setminus a_t = 0$$

Therefore the inner expectation is a constant for all t' less then t, and the score function estimator is zero in expectation at these terms in the series. This sum of (entropy regularised) rewards ahead is often referred to as the Q function or the advantage (if we include a baseline or value function which integrates over all actions). This term can be defined more explicitly as,

$$Q(s_t, a_t) = \mathop{\mathbb{E}}_{q_\phi(\tau_{t+1:T}|a_t, s_t)} \left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) - \log \frac{\pi_\phi(a_{t'}|s_{t'})}{\pi_0(a_{t'}|s_{t'})} \right].$$

Using this estimate of the expected rewards ahead, we can re-scale the log probability of certain actions and give them more weight in the gradient estimator. Crucially, the variance of this gradient estimator is strictly lower, while maintaining the correct target distribution of interest. The full marginalised objective and its estimator are given as the following:

$$\nabla_\phi J_{q,p}(\phi) = \sum_{t=0}^{T-1} \mathop{\mathbb{E}}_{q_\phi(a_t, s_t)} [\nabla_\phi \log \pi_\phi(a_t|s_t) Q(s_t, a_t)] \approx \frac{1}{NM} \sum_{t=0}^{T} \sum_{m,n} \left[\nabla_\phi \log \pi_\phi(a_t^m|s_t^n) Q(s_t^n, a_t^m) \right],$$
$$\text{where} \quad a_t^m \sim \pi_\phi(a_t|s_t^n), \quad \text{and} \quad s_t^n \sim q_\phi(s_t).$$

5.4 Reweighted Policy Gradient Estimators Via the Forward KL

This section details derivations for how to produce an approximately optimal policy under the forward KL, and like the previous section we will produce a lower variance gradient estimator which targets the original distribution of interest. Here we take advantage of tools from self normalised importance weighting to produce a biased but consistent estimator, and briefly discuss practical issues.

Un-marginalised Objective Derivation for the Forward KL

This objective, represents the expected KL divergence between a target distribution, and an approximate distribution under samples from the target distribution given by $KL(p(\tau|O)||q_\phi(\tau))$. Again, our latent variables are given as the associated state-action pairs, while the observed variables represents the optimality distribution. In order to evaluate the expression above, we apply a standard importance weighting scheme, beginning with the removal constant terms within the expectation.

$$KL(p(\tau|O)||q_\phi(\tau)) = \int p(\tau|O)\log\frac{p(\tau|O)}{q_\phi(\tau)}d\tau = -\int p(\tau|O)\log q_\phi(\tau)d\tau + \int p(\tau|O)\log p(\tau|O)d\tau$$
$$= \int \frac{1}{p(O)}\log q_\phi(\tau)\frac{p(\tau,O)}{q_\phi(\tau)}q_\phi(\tau)d\tau + c = \frac{1}{Z}\mathop{\mathbb{E}}_{\tau\sim q_\phi}\left[\log q_\phi(\tau)\frac{p(\tau,O)}{q_\phi(\tau)}\right] + c$$

Un-marginalised gradient of the Forward KL Objective

Crucially, because the original problem is over a distribution not dependent on ϕ, we can directly pass the gradient into the expectation without the REINFORCE trick, or reparameterisation:

$$\nabla_\phi KL(p(\tau|O=1)||q_\phi(\tau)) = \mathop{\mathbb{E}}_{\tau\sim q_\phi}\left[\frac{p(\tau,O)}{q_\phi(\tau)}\frac{\nabla_\phi \log q_\phi(\tau)}{Z}\right].$$

This actually gives us a simple algorithm that can evaluate the gradient, and thereby minimise the expected KL between our two distributions following samples generated from our policy interacting with the simulator. Based upon this

approach, we can apply self normalised importance weighting to avoid explicitly computing the constant Z, at the cost of a biased but consistent estimator:

$$\nabla_\phi KL(p(\tau|O)||q_\phi(\tau)) \approx \sum_{i=1}^{m} [w_i \nabla_\phi \log q_\phi(\tau_i)] = \frac{1}{Z} \sum_{i=1}^{m} \left[\frac{p(\tau_i, O_i)}{q_\phi(\tau_i)} \nabla_\phi \log q_\phi(\tau_i) \right].$$

Where for optimal trajectories ($O = O_{t=0:T-1} = 1$), the un-normalised weights are defined as,

$$w_i = \prod_{t=0}^{T-1} \frac{\frac{1}{Z_0} \exp r_t(a_t, s_t) \pi_0(a_t|s_t)}{\pi_\phi(a_t|s_t)} = \frac{1}{\hat{Z}_0} \exp \left[\sum_{t=0}^{T-1} r(a_t^i, s_t^i) - \log \left(\frac{\pi_\phi(a_t^i|s_t^i)}{\pi_0(a_t^i|s_t^i)} \right) \right]$$

We also include reward normalisation constant Z_0 for generality. In order to avoid computation of Z explicitly, we take advantage of the consistent but biased self normalised importance weighting estimator. This means weights w_i are replaced with $\hat{w}_i$ using $\tau_i \sim q_\phi(\tau)$:

$$\hat{w}_i = \frac{w_i}{\sum_{j=1}^{m} w_j} = \frac{\exp \left[\sum_{t=0}^{T-1} r(a_t^i, s_t^i) - \log \left(\frac{\pi_\phi(a_t^i|s_t^i)}{\pi_0(a_t^i|s_t^i)} \right) \right]}{\sum_{j=1}^{m} \exp \left[\sum_{t=0}^{T-1} r(a_t^j, s_t^j) - \log \left(\frac{\pi_\phi(a_t^j|s_t^j)}{\pi_0(a_t^j|s_t^j)} \right) \right]}. \tag{5.7}$$

In this case we now have a biased estimator of the gradient, but also one that is generally low variance when compared to its un-marginalised RKL counterpart [Le et al., 2018],

$$\nabla_\phi KL(p(\tau|O)||q_\phi(\tau)) \approx \sum_{i=1}^{n} \frac{\exp \left[\sum_{t=0}^{T-1} r(a_t^i, s_t^i) - \log \left(\frac{\pi_\phi(a_t^i|s_t^i)}{\pi_0(a_t^i|s_t^i)} \right) \right]}{\sum_{j=1}^{n} \exp \left[\sum_{t=0}^{T-1} r(a_t^j, s_t^j) - \log \left(\frac{\pi_\phi(a_t^j|s_t^j)}{\pi_0(a_t^j|s_t^j)} \right) \right]} \nabla_\phi \log q_\phi(\tau_i) \tag{5.8}$$

Marginalization Proof for forward KL

Like the reverse KL, we can further reduce variance of our gradient estimator. In order to show that the objective can be marginalised with respect to actions given states and optimality, we start with the original definition of the objectives gradient. Using d separation of the probabilistic graphical model to ignore the optimality variables prior to the state conditioned on we have that:

$$\nabla_\phi J_{p,q}(\phi) = \sum_{t=0}^{T-1} \int_\tau p(a_0, s_0, ...s_T|o_t, ..., o_{T-1}, s_t) p(s_t|o_0, ..., o_{T-1}) \nabla_\phi \log \pi_\phi(a_t|s_t) d\tau.$$

Next looking at just one element of this sum at index t,

$$
\begin{aligned}
&= \int_{s_t} ... \int_{s_T} \int_{a_t} ... \int_{a_{T-1}} p(s_t|O_0...O_{T-1}) \nabla_\phi \log \pi_\phi(a_t|s_t) \\
&\quad ... \int_{s_0} ... \int_{s_{t-1}} \int_{a_0} ... \int_{a_{t-1}} p(a_0, s_0, ...s_T|o_t, ..., o_T, s_t) d\tau \\
&= \int_{s_t} ... \int_{s_T} \int_{a_t} ... \int_{a_{T-1}} p(s_t|O_0...O_{T-1}) \nabla_\phi \log \pi_\phi(a_t|s_t) p(a_t, s_{t+1}...s_T|o_t, ..., o_T, s_t) d\tau \\
&= \int_{s_t} ... \int_{s_T} \int_{a_t} ... \int_{a_{T-1}} p(s_t|O_0...O_{T-1}) \nabla_\phi \log \pi_\phi(a_t|s_t) \frac{p(a_t, s_{t+1}...s_T, o_t, ..., o_T|s_t)}{p(o_t, ..., o_T|s_t)} d\tau \\
&= \int_{s_t} ... \int_{s_T} \int_{a_t} ... \int_{a_{T-1}} \frac{p(s_t|O_0...O_{T-1})\pi_0(a_t|s_t)}{p(o_t, ..., o_T|s_t)} \nabla_\phi \log \pi_\phi(a_t|s_t) p(s_{t+1}...s_T, o_t, ..., o_T|s_t, a_t) d\tau \\
&= \int_{s_t} ... \int_{s_T} \int_{a_t} ... \int_{a_{T-1}} \frac{p(s_t|O_0..O_{T-1})\pi_0(a_t|s_t)}{p(o_t, ..., o_T|s_t)} \nabla_\phi \log \pi_\phi(a_t|s_t) p(s_{t+1}...s_T, o_t, ..., o_T|s_t, a_t) d\tau \\
&= \mathop{\mathbb{E}}_{p(s_t|O_{0:T})} \left[\frac{1}{p(O_{t:T}|s_t)} \mathop{\mathbb{E}}_{a_t \sim \pi_0(a_t|s_t)} \left[\nabla_\phi \log \pi_\phi(a_t|s_t) \mathop{\mathbb{E}}_{q_\phi(\tau_{t+1:T}|a_t, s_t)} \left[\frac{p(\tau_{t+1:T}, O_{t:T}|a_t, s_t)}{q_\phi(\tau_{t+1:T}|a_t, s_t)} \right] \right] \right]
\end{aligned}
$$

Notably, we can define a specific expectation for convenience, which can be either be estimated through sampling or a learned function approximator as is done in practice [Abdolmaleki et al., 2018, Piché et al., 2019]:

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{q_\phi(\tau_{t+1:T}|s_t, a_t)} \left[\frac{p(\tau_{t+1:T}, O_{t:T}|a_t, s_t)}{q(\tau_{t+1:T}|a_t, s_t)} \right] = \mathop{\mathbb{E}}_{q_\phi(\tau_{t+1:T}|s_t, a_t)} \left[\frac{p(\tau_{t+1:T}, O_{t:T}|a_t, s_t)}{q_\phi(\tau_{t+1:T}|a_t, s_t)} \right] \\
&= \mathop{\mathbb{E}}_{q_\phi(\tau_{t+1:T}|s_t, a_t)} \left[\exp \left[r(a_t, s_t) + \sum_{t'=t+1}^{T} r(a_{t'}, s_{t'}) - \frac{\pi_\phi(a_{t'}|s_{t'})}{\pi_0(a_{t'}|s_{t'})} \right] \right] = \exp(\hat{Q}(a_t, s_t))
\end{aligned}
$$

We then note the following about the normalization constant sitting outside the action expectation:

$$
p(O_{t:T}|s_t) = \int p(O_{t:T}, a_t|s_t) da_t = \int p(O_{t:T}|a_t, s_t) \pi_0(a_t|s_t) da_t = \mathop{\mathbb{E}}_{a_t \sim \pi_0(a_t|s_t)} \left[\exp(\hat{Q}(a_t, s_t)) \right]
$$

This definition allows us to define our marginalized expectation more simply, and identify the crucial issue with direct estimation of the marginalized forward KL:

$$
\nabla_\phi J_{p,q}(\phi) = \mathop{\mathbb{E}}_{p(s_t|O_{0:T})} \left[\mathop{\mathbb{E}}_{\pi_\phi(a_t|s_t)} \left[\frac{\nabla_\phi \log \pi_\phi(a_t|s_t) \exp \left[\hat{Q}(a_t, s_t) - \log \left[\frac{\pi_\phi(a_t|s_t)}{\pi_0(a_t|s_t)} \right] \right]}{\mathbb{E}_{a'_t \sim \pi_\phi(a'_t|s_t)} [\exp \hat{Q}(a'_t, s_t) - \log \left[\frac{\pi_\phi(a_t|s_t)}{\pi_0(a_t|s_t)} \right]]} \right] \right]
$$

We can deal with the outer expectation in terms of $p(s_t|O_{0:T})$ as well, though it induces a less satisfactory estimator whose variance again grows with time-horizon:

$$\mathop{\mathbb{E}}_{p(s_t|O_{0:T})}[f(s_t)] = \int f(s_t)p(s_t|O_{0:T})ds_t = \int f(s_t)\frac{p(s_t, O_{0:T})}{p(O_{0:T})}ds_t = \int f(s_t)p(s_t)\frac{p(O_{0:T}|s_t)}{p(O_{0:T})}ds_t$$
$$= \frac{1}{p(O_{0:T})}\mathop{\mathbb{E}}_{p(s_t)}[p(O_{0:T}|s_t)f(s_t)] = \frac{\int p(O_{0:T}, s_t)f(s_t)ds_t}{\int p(O_{0:T}, s_t)ds_t} = \frac{\int p(O_{0:T}, \tau_{0:T})f(s_t)d\tau_{0:T}}{\int p(O_{0:T}, \tau_{0:T})d\tau_{0:T}}$$
$$= \mathop{\mathbb{E}}_{q_\phi(\tau_{0:T})}\left[\frac{p(O_{0:T}, \tau_{0:T})}{q_\phi(\tau_{0:T})}f(s_t)\right] \Big/ \mathop{\mathbb{E}}_{q_\phi(\tau_{0:T})}\left[\frac{p(O_{0:T}, \tau_{0:T})}{q_\phi(\tau_{0:T})}\right]$$

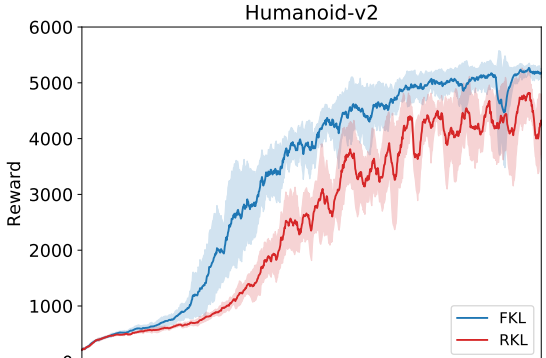
Such an estimator would function by sampling N independent trajectories from q_ϕ, and then for the specified time, reweigh each function f at its respective state by its associated regularised cumulative return. With this consideration in mind, we can define importance weights over states and actions for each time-step which produce a consistent estimate of the forward KL gradient:

$$\nabla_\phi J_{p,q}(\phi) \approx \sum_{t=0}^{T}\sum_{i=1}^{M}\bar{y}_i^t\sum_{k=1}^{N}\bar{x}_{i,k}^t\nabla_\phi \log \pi_\phi(a_t^k|s_t^i), \quad \text{with } \tau^i \sim q_\phi(\tau),\ a_t^k \sim \pi_\phi(a_t|s_t^i)$$
$$\text{where} \quad \bar{y}_i^t = \frac{y_i^t}{\sum_{j=1}^{M} y_j^t} \quad \text{with} \quad y_i^t = \exp\left[\sum_{t'=0}^{T-1} r(a_{t'}^i, s_{t'}^i) - \log\left(\frac{\pi_\phi(a_{t'}^i|s_{t'}^i)}{\pi_0(a_{t'}^i|s_{t'}^i)}\right)\right]$$
$$\text{and} \quad \bar{x}_{i,k}^t = \frac{x_{i,k}^t}{\sum_{j=1}^{N} x_{i,j}^t} \quad \text{with} \quad x_{i,k}^t = \exp\left[\hat{Q}(a_t^k, s_t^i) - \log\left[\frac{\pi_\phi(a_t^k|s_t^i)}{\pi_0(a_t^k|s_t^i)}\right]\right]$$

For our experiments, as done in practice in [Abdolmaleki et al., 2018, Piché et al., 2019], we reweigh the loss each at each state uniformly (i.e. $\bar{y}_i^t = \frac{1}{M}$), to avoid high variance of the gradient estimator, ensure proper coverage of the state-space for better generalization, and to maintain sample efficiency within the larger RL algorithm. Additionally we target the classic Q function instead of the soft Q function $\bar{Q}$, as we found that using this lower bound on $\bar{Q}$ tended to produce more stable learning within the policy as in prior work [Abdolmaleki et al., 2018].

5.5 Experiments

To display the relative merits of both objectives, we include experiments where a native implementation of soft actor critic was taken, and modified



to include the forward KL objective (FKL). We begin with an illustration of an instance where you can find improvement by replacing the more common RKL for the FKL objective. In Figure 5.2, we plot results of training on the Humanoid benchmark from MuJoCo using RKL and FKL. Here, the FKL improves sample efficiency over the RKL, demonstrating approximately 60% increase in sample efficiency. Humanoid is the most challenging of the five tasks considered from the MuJoCo suite.

Additional advantages of the FKL policy objective include its ease of implementation and its distribution agnostic property. This property means that in discrete control, or for hierarchical policies with discrete branching, the RKL objective would require further modification, while the FKL functions as is. One drawback that we note, however, is that the FKL estimator scales with computation, i.e. its bias and thus its performance is dependent on the number of N re-samples from the policy. Finally, for continuous control, we additionally show that the FKL can also improve the algorithm's robustness to changes in hyper-parameters. All experiments are run with at 10 replicates and plotted with one-half standard deviation and smoothed using a 0.9 weighted EMA. As seen in Figure 5.3, the forward KL can maintain policy performance, even when memory is severely limited.

5.6 Related Work

There is a growing literature on various methods that incorporate the RLAI framework discussed within this paper [Abdolmaleki et al., 2018, Haarnoja et al., 2018b, Levine, 2018, Ziebart, 2010, Ziebart et al., 2008, Kappen et al., 2012, Neumann et al., 2011, Van De Meent et al., 2016]. In many cases these works use RLAI to give theoretical basis for the empirical effectiveness of well established implementation tricks such as entropy bonuses [Haarnoja et al., 2018b, Levine, 2018, Ziebart, 2010].

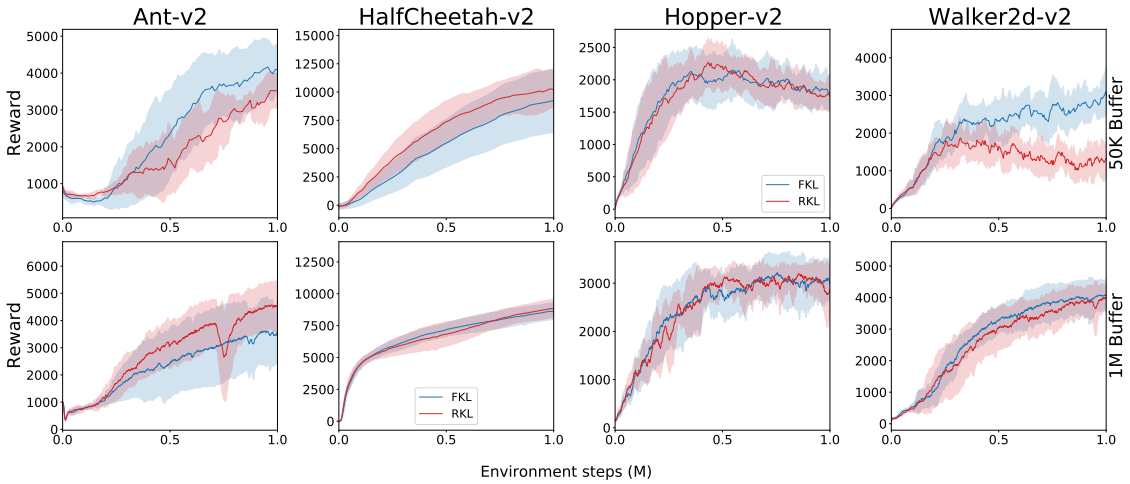


Figure 5.3: First row: comparison of RKL and FKL on MuJoCo benchmark tasks using a base implementation of SAC *without* hyper-parameter tuning for MuJoCo. The notable difference in hyper-parameters between these settings is the buffer sizes of 50K versus 1M for the first and second rows, respectively. We find under these conditions, FKL is always competitive or significantly better than the baseline with the exception of the Ant benchmark using 1M replay buffer size. In all figures, error bars denote one-quarter standard deviation.

Very few works however, have considered the comparison between the forward and reverse KL as well as the relative merits of each in the context of reinforcement learning objectives [Neumann et al., 2011, Lyu and Chen, 2019]. Even fewer produce scalable and sample efficient implementations of both objectives, making a proper comparison difficult. We note for completeness that work has also been done in formalizing methods relating to the original policy gradient algorithm [Williams, 1992, Schulman et al., 2017, 2015a] using tools from the optimisation literature [Ghosh et al., 2020, Vaswani et al., 2021], though these analysis do not directly relate the the RLAI framework. Work by [Chan et al., 2021] more closely relates to this work in their analysis of improvement under the FKL, however they do not focus on practical gradient estimation. Similarly there has been work analysing critic and gradient estimation [Chung et al., 2021] in average return MDPs, as well as empirical studies for the more commonly utilised discounted MDPs discussed above [Ilyas et al., 2018].

Algorithmically, the closest class of algorithms to our work in the classical literature comes from policy search methods [Neumann et al., 2011, Montgomery

and Levine, 2016, Levine and Koltun, 2013, Colomé and Torras, 2017, Peters et al., 2010, Deisenroth et al., 2013]. Of the subset of these that have been made scalable, closely related algorithms to our work come from Abdolmaleki et al. [2018], Song et al. [2019], and Chow et al.. Unlike Abdolmaleki et al. [2018], we do not require an alternating update and directly learn the policy via the forward KL. Additionally while we do not provide monotonic improvement guarantees which require additional assumptions and a trust-region update, we do provide a full proof of marginalisation of the forward KL as it applies to RLAI in a finite horizon setting. A similar proof for a marginalisation can be found in Piché et al. [2019], however this assumes access to the steady state distribution of the Markov chain as well as a model of the environment. We do note that the objective for the target policy given in Song et al. [2019] and Peng et al. [2019] come very close to our own, however each requires additional parameterisation of a value function, as well as a more complicated alternating update for the temperature coefficient α, and a different inference formulation which culminates in an advantage based objective. Most importantly, while these works represent end-to-end algorithms, our estimators can be applied a wide variety of existing frameworks.

5.7 Discussion

This chapter describes in detail the differences in policy objective which occur both mathematically, and practically under a divergence minimisation perspective of RLAI. Crucially, we show that there are very real advantages to considering other objectives besides the classic policy gradient methods which are most common. While these alternatives like the FKL are generally not going to drastically change performance, their ease of implementation, robustness to hyper-parameters, and their distribution agnostic nature make them a very desirable alternative to classic methods. In the future we would like to test how these objectives function on a wider variety of benchmarks and policy distributions. Additionally, its possible that better non-uniform sampling methods might be derived following $p(s_t|O_{0:T})$. Though estimating an expectation under such a distribution might be difficult,

and potentially require investigation into the particle smoothing literature [Doucet et al., 2001].

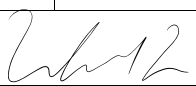
Statement of Authorship for joint/multi-authored papers for PGR thesis

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

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

Title of Paper	A Closer Look at Gradient Estimators in RL as Inference
Publication Status	☐Published ☐ Accepted for Publication ☒Submitted for Publication ☐Unpublished and unsubmitted work written in a manuscript style
Publication Details	Lavington, J. W., Teng, M., Schmidt, M., & Wood, F. (2021). A Closer Look at Gradient Estimators in RL as Inference. *NeurIPS DeepRL Workshop*

Student Confirmation

Student Name:	Michael Teng		
Contribution to the Paper	The paper is mostly written by Jonathan Wilder Lavington. He and I went through the derivations together, and Frank Wood and Mark Schmidt helped with paper writing. I implemented all experiments for the paper, as well as algorithms.		
Signature		Date	Jan 11, 2022

Supervisor Confirmation

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

Supervisor name and title:	Dr. Frank Wood, Associate Professor		
Supervisor comments:	I concur with Mr. Teng's assessment of his contributions to this work.		
Signature		Date	11 / Jan / 2022

6

Exploration with Multi-Sample Target Values for Distributional Reinforcement Learning

Contents

6.1 Introduction 125
6.2 Background 128
6.2.1 Soft Actor-Critic 129
6.2.2 Implicit Quantile Networks 130
6.2.3 Distributional SAC 131
6.3 Extra Exploration with Distributional Critics 132
6.3.1 Multi-sample Target Value 134
6.3.2 UCB Exploration 135
6.3.3 Algorithm Description 135
6.4 Experiments 136
6.4.1 Distribution Matching Study 137
6.4.2 MuJoCo and Box2d 138
6.4.3 Humanoid Ablation Study 139
6.5 Discussion 141

6.1 Introduction

Actor-critic methods generally follow two main design principles: (1) a neural network parametrised estimate of the expectation of the value function (critic)

and (2) a stochastic policy to encourage exploration (actor). While this set of methods have been shown to be robust, estimating the expectation of the value function is at odds with the intrinsic stochasticity of the cumulative reward ahead introduced by the policy (Figure 6.2; bottom-left). To address this, the field has recently turned to modelling the full distribution of returns, introducing the 'distributional RL' framework.

We also motivate this work from the RLAI framework. Although we treat the control problem as an inference task, we are still left with point estimates for the sum of expected rewards ahead. However, this leaves us with a free choice over the design of the value-functions used, even in RLAI. Here we explore a different approach, where even these estimates incorporate uncertainty estimates via distributional RL. Considering the form of gradient estimators given in Chapter 5, we may also be able to utilise this for less biased estimates in the FKL objective. Additionally, distributional RL is known to be more sample efficient than classical TD-learning.

Along with the inefficiency of fitting a point estimate to the empirical distribution of returns, only capturing the first moment introduces additional drawbacks for tackling exploration in the RL setting. When RL is viewed as a bandit problem, the stochastic nature of the observed return of each action at a given state has multiple sources, including the sampling policy used during training to take subsequent actions, and the stochastic nature of the observed rewards and state transitions. When the parametrised value estimate does not quantify uncertainty over the return, exploration techniques used in bandit literature cannot readily be ported to actor-critic algorithms. Recently, work has begun to address this issue by leveraging uncertainty estimates of the estimated value-function to direct exploration in the RL setting. Ciosek et al. [2019] computes an upper and lower bound using an ensemble of value-functions, Chen et al. [2017] utilise an ensemble of actor-critic agents to compute an upper confidence bound (UCB) for exploration, Lee et al. [2020] introduce a state-of-the-art ensemble method called SUNRISE, Tang and Agrawal [2018] apply posterior sampling for exploration with distributional critics,

and finally, both Moerland et al. [2017] and Mavrin et al. [2018] use the uncertainty of distributional value-functions for additional exploration using Thompson sampling and UCB, respectively.

Despite these recent advances that leverage either a distributional critic or an ensemble of critics for exploration purposes, there remains a gap in designing a simple, principled approach for additional exploration based on value-function estimation. This is particularly true for continuous control tasks and being able to achieve stronger performance on standard benchmarks. To the best of our knowledge, the exploration problem in state-of-the-art actor-critic methods for continuous control has largely been left to adding randomness to the policy, either with entropy regularization in Soft Actor-Critic (SAC; Haarnoja et al. [2018b]), adding noise to a deterministic policy in deep deterministic policy gradients (DDPG; Lillicrap et al. [2015], TD3; Fujimoto et al. [2018]), or to use more expressive policies with hierarchies or normalizing flows [Haarnoja et al., 2018a, Ward et al., 2019]. Otherwise, recent advances to leverage critics for exploration purposes must train ensembles of both policies and their respective value-functions, which is computationally expensive and performs comparably to RL methods that are not designed for sample efficiency. This is reflected in Table 6.1, where the ensemble-based SUNRISE method [Lee et al., 2020] performs roughly on par with a distributional method such as IQN [Dabney et al., 2018a].

In this chapter, we address this gap by taking advantage of the advances in distributional RL, which rely on estimating the full distribution of returns at each state. The distributional RL framework has been initially applied to discrete control tasks, and a series of improvements have been made since. First, Bellemare et al. [2017] introduce the C51 algorithm using a KL-divergence loss to fit a distributional estimate of the value-function. Dabney et al. [2018b] improves upon this work by replacing the KL-divergence loss with quantile regression to learn a better estimate called Quantile Regression Deep Q-Networks (QR-DQN). Finally, Dabney et al. [2018a] introduce an architectural improvement to the value-function estimator called Implicit Quantile Networks (IQN). Subsequently, continu-

ous control counterparts have been used successfully, with distributed distributional DDPG (D4PG) adding QR-DQN to the DDPG algorithm [Barth-Maron et al., 2018] and distributed SAC (DSAC) integrating IQN into the SAC algorithm [Ma et al., 2020]. We continue in the direction of these works via a principled multi-sample improvement to the use of the distributional estimates, in the context of continuous control. We then show that these can be effectively combined with bandit-like UCB-based approaches to further exploration.

The rest of this chapter is organised as follows. Section 2 reviews relevant background including the SAC, IQN, and DSAC algorithms. Section 3 introduces our novel algorithm called Extra Exploration with Distributional Critics (E2DC), which modifies the way in which IQN captures uncertainty and couples this with UCB for exploration. Section 4 gives experimental results for our algorithm and Section 5 discuses future work.

6.2 Background

In this section, we review relevant background starting with the RL problem setup and the algorithms we build on including SAC, IQN, and DSAC.

Recall from Chapter 2−Section 2.2.3 the definitions of our MDP, and the sum of expected rewards ahead. Our MDP considers states $s \in S$, actions $a \in A$, initial state distribution $p_0(s_0)$, transition function $p(s_{t+1}|s_t, a_t)$, and the random reward $R : S \times A \to \mathbb{R}$, distributed as $R(s,a) \sim r(\cdot|s,a)$. Our goal is to learn a parametrised policy, $\pi_\phi(a_t|s_t)$, to maximise $\mathbb{E}_{\pi,p,r}\left[\sum_t \gamma^t R(a_t, s_t)\right]$.

Once, again, we have the state-value, $V^\pi(s)$, and the action-value, $Q^\pi(s)$, and the full distribution of each:

$$Q^\pi(s,a) := \mathop{\mathbb{E}}_{\pi,p,r}\left[\sum_t \gamma^t R(a_t, s_t) \quad | \quad a_0 = a,\ s_0 = s\right] \tag{6.1}$$

$$V^\pi(s) := \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a)\right] \tag{6.2}$$

$$Z^\pi(s,a) :\overset{D}{=} \sum_t \gamma^t R(a_t, s_t) \quad | \quad a_0 = a,\ s_0 = s, s_{t+1} \sim p(\cdot|s_t, a_t), a_t \sim \pi(\cdot|s_t), \tag{6.3}$$

$$Z^\pi(s) :\overset{D}{=} Z^\pi(s,a) \quad | \quad a \sim \pi(\cdot|s) \tag{6.4}$$

6.2.1 Soft Actor-Critic

Soft Actor-Critic is a state-of-the-art, model-free algorithm for continuous control. Haarnoja et al. [2018b] introduce SAC as soft policy iteration with a dueling critic architecture combined with a reparametrized stochastic policy based on the maximum entropy framework. In the following notation, we assume the entropy bonus as part of the stochastic reward at each timestep for clarity. They show that their algorithm converges to an optimal policy for maximizing the expected return subject to a maximum entropy policy by iteratively performing soft policy evaluation and soft policy improvement.

We first define the soft policy evaluation step. For a fixed policy, π, and a initial parametrization of a Q-function, Q_θ, a soft Bellman operator, $\mathcal{T}$ is defined:

$$(\mathcal{T}^\pi Q)(s,a) = R(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim p} \Big[V(s') \Big]. \tag{6.5}$$

Because this operator is contractive in the space of possible Q-functions, i.e. $||\mathcal{T}^\pi Q_1 - \mathcal{T}^\pi Q_2||_\infty \leq \gamma ||Q_1 - Q_2||_\infty$, repeated applications of the operator to the initial Q_θ converges to the fixed point solution of Q^π [Haarnoja et al., 2017]. This leads to a gradient update which minimizes the MSE distance between the current estimate produced by Q_θ and the application of the Bellman operator, referred to as the *target*. This is referred to as Q-learning, using samples from a replay buffer, $\mathcal{D}$, in the off-policy setting:

$$\mathcal{L}_\theta^Q(s,a) = \mathop{\mathbb{E}}_{s,a \sim \mathcal{D}} \Bigg[\mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a)} \Bigg[\Big(R(s,a) + \gamma \mathbb{E}_{a' \sim \pi} [Q_{\bar{\theta}}(s',a')] - Q_\theta(s,a) \Big)^2 \Bigg] \Bigg], \tag{6.6}$$

where $\bar{\theta}$ denotes frozen parameters (either synchronized with θ at some fixed iterations or updated using a moving average of θ. Haarnoja et al. [2018b] show that learning a separate parametrized estimate of $V(s)$ can be done to stabilize training, however, in practice, the value-estimate of s' in the above equation is often approximated with a single sample of the Q-function, i.e. $V(s') \approx Q(s', \text{sample}\{\pi(\cdot|s')\})$.

Next, for soft policy improvement, we update the parametrized policy, π_ϕ, by minimizing the expected KL-divergence between current policy and the exponential

Table 6.1: Performance of trained agents at 200K timesteps. The results shown are mean and standard deviation averaged over 5 seeds and the first two rows are taken from Lee et al. [2020].

Easy tasks	Cheetah	Walker	Hopper	Ant
SAC	4035.7 ± 268.0	-382.5 ± 849.5	2020.6 ± 692.9	836.5 ± 68.4
SUNRISE	5370.6 ± 483.1	1926.5 ± 694.8	$\mathbf{2601.9 \pm 306.5}$	$\mathbf{1627.0 \pm 292.7}$
SAC (ours)	5547.2 ± 261.2	556.8 ± 115.8	1833.7 ± 614.8	1551.5 ± 254.0
IQN	6489.7 ± 655.8	$\mathbf{1964.5 \pm 594.8}$	1604.4 ± 757.7	1557.6 ± 602.9
E2DC	$\mathbf{7004.8 \pm 856.5}$	1789.9 ± 979.3	1689.5 ± 615.8	1087.5 ± 374.1
Hard tasks	Swimmer	Humanoid	HStand	Biped (H)
SAC (ours)	41.4 ± 0.59	862.3 ± 149.1	149K $\pm$4.6K	-97.8 ± 14.1
IQN	$\mathbf{48.9 \pm 0.81}$	1154 ± 249.4	140K $\pm$30.8K	-0.05 ± 43.5
E2DC	48.8 ± 2.3	$\mathbf{2218.3 \pm 1394.5}$	**176K $\pm$21.5K**	$\mathbf{7.93 \pm 52.3}$

of the Q-value. The loss derived from this can use the reparametrization trick when assuming Gaussian policies:

$$\mathcal{L}^{\pi}_{\phi}(s) = \mathop{\mathbb{E}}_{s\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{a\sim\pi_{\phi}(\cdot|s)}\Big[Q_{\theta}(a,s) + \alpha \log \pi_{\phi}(a|s)\Big]\right], \tag{6.7}$$

where α is a hyperparameter controlling how entropic the policy should be. Finally, more recent implementations of SAC also tunes a temperature coefficient using,

$$\mathcal{L}^{\alpha}(s) = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}\left[\alpha \log \pi(a|s) - \alpha\bar{\mathcal{H}}\right]\right], \tag{6.8}$$

which targets a fixed entropy, $\bar{\mathcal{H}}$ defined as the expected negative log of the policy $\bar{\pi}$, under sampled actions throughout optimization. We refer the reader to Haarnoja et al. [2018c] for details.

6.2.2 Implicit Quantile Networks

Implicit Quantile Networks is a distributional RL method developed first for discrete action spaces. In the distributional critic setting, we first define a distributional Bellman operator:

$$(\mathcal{T}^{\pi}Z)(s,a) \overset{D}{:=} R(s,a) + \gamma Z(s', \arg\max_{a'\in\mathcal{A}} Q(s',a')) \quad | \quad s' \sim p(\cdot|s,a), \tag{6.9}$$

where $Q(s,a)$ is obtained via Equation 2.83. In Dabney et al. [2018b], the authors replace mean-valued Q-functions with quantile functions, Z^{τ}, which approximate

the inverse CDF, $F_Z^{-1}(\tau) := \inf\{z \in R : \tau \leq F_Z(z)\}$, where $F_Z(z) := Pr(Z < z)$ denotes the CDF. For a given quantile fraction sampled uniformly on the interval, $\tau \sim [0, 1]$, then Z^τ is a sample of the random variable Z.

Given a parametrized quantile function, $Z_\theta^\tau(s, a)$, the authors show that using the Huber quantile regression loss gives a contractive distributional Bellman operator (Equation 6.9) in the p-Wasserstein metric for inverse CDFs, i.e. $d_{\mathcal{Z}}(\mathcal{T}^\pi Z_1, \mathcal{T}^\pi Z_2) \leq \gamma d_{\mathcal{Z}}(Z_1, Z_2)$, where:

$$d_{\mathcal{Z}} := W_p(U, V) = \left(\int_0^1 |F_U^{-1}(w) - F_V^{-1}(w)|^p dw \right)^{1/p}, \tag{6.10}$$

Finally, this leads to a loss that minimizes the Wasserstein distance between the parametrized quantile function, Z_θ and its Bellman target, referred to as Z-learning:

$$\mathcal{L}_\theta^Z(s, a) = \mathop{\mathbb{E}}_{s,a\sim\mathcal{D}} \left[\mathop{\mathbb{E}}_{s'\sim p} \left[\frac{1}{M} \sum_{i=1}^{N} \sum_{j=1}^{M} \rho_\tau^H \left(R(s, a) + \gamma Z_\theta^{\tau_j}(s', \arg\max_{a'} Q(s', a')) - Z_\theta^{\tau_i}(s, a) \right) \right] \right],$$
$$\rho_\tau^H(u) = |\tau - \mathbb{1}_{u<0}| \mathcal{L}_H^\kappa(u), \quad \mathcal{L}_H^\kappa(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \leq \kappa \\ \kappa(|u| - \frac{1}{2}\kappa), & \text{otherwise} \end{cases} \tag{6.11}$$
$$\{\tau_i\}_{i=1}^N, \{\tau_j\}_{j=1}^M \sim \text{Uniform}([0, 1])$$

6.2.3 Distributional SAC

In Distributional SAC, Ma et al. [2020] update the SAC algorithm with a quantile critic. Instead of Equation 6.9, the authors define the distributional soft Bellman operator:

$$\begin{aligned} (\mathcal{T}_{\text{soft}}^\pi Z)(s, a) &\overset{D}{:=} R(s, a) + \gamma Z(s') \\ &\overset{D}{:=} R(s, a) + \gamma \Big[Z(s', a') - \alpha \log \pi(a'|s') \Big] \quad | \quad s' \sim p(\cdot|s, a),\ a' \sim \pi(\cdot|s') \end{aligned} \tag{6.12}$$

In practice, Equation 6.12 is approximated with a single sample, $a' \sim \pi(\cdot|s')$, as is the case in the classic SAC algorithm. Our contribution here takes advantage of distributional critics, in that samples from multiple action-value distributions can be pooled to compute a better estimate of $Z(s)$. Additionally, DSAC retains the advances of SAC with entropy tuning and a dueling Z-network estimate, $Z^\tau := \min(Z_{\theta_2}^\tau, Z_{\theta_1}^\tau)$. Finally, DSAC performs policy improvement step that with an expectation of the Q-value by taking the mean of the Z-distribution

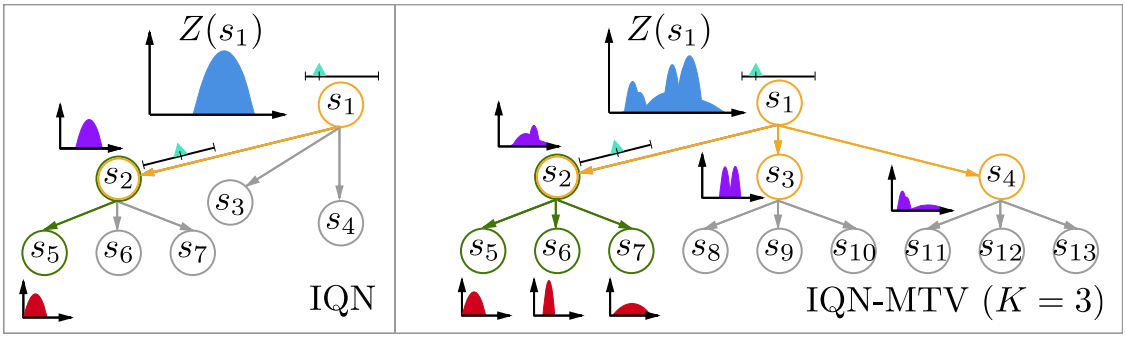


Figure 6.1: Propagation of information in temporal difference Z-learning for IQN and IQN-MTV. In this illustrative MDP, there are 3 possible actions at each state: left, center, right. For each layer in the tree, the green and yellow path denotes subsequent actions that are pooled in the Bellman target. The teal triangles denote the reward obtained by arriving at states $R(s_1), R(s_2)$, which are added to a discounted Z-values from either single states in IQN or multiple states for IQN-MTV. Intuitively, higher K in IQN-MTV leads to easier learning of multimodal Z-networks.

6.3 Extra Exploration with Distributional Critics

In this section, we introduce our method for additional exploration, but first introduce two concepts that motivated our algorithm. First, we distinguish between uncertainty quantified by using ensembles of critics and distributional critics. The former relies on capturing parametric uncertainty, which is uncertainty over the parameters given in the beginning of training, while the latter relies on capturing return uncertainty, i.e. the uncertainty of returns based on a stochastic policy. Moerland et al. [2017] compare using distributional critics for exploration using return uncertainty against using Bayesian neural networks for exploration leveraging parametric uncertainty. In 4 of the 5 difficult tasks they considered, exploration based on the return uncertainty outperformed all others. In this work, we do not focus on ensembles or Bayesian neural networks to capture parametric uncertainty, as several works have already explored this question [Lee et al., 2020, Wiering and Van Hasselt, 2008]. Instead, we address the gap in the literature posed by Moerland et al. [2017], which is to better understand the advantages of using return uncertainty to drive exploration and learning.

Alongside the clear utility of distributional critics in capturing return uncertainty, several works have posited that the uncertainty associated with the return distribution is also subject to bootstrapping error in temporal difference settings [Flennerhag et al., 2020, Mavrin et al., 2018]. As a result, distributional methods may also capture *parametric* uncertainty through the accumulation of these errors, especially so at the start of training [Mavrin et al., 2018]. In the offline, non-distributional RL setting, Kumar et al. [2019] show that when using a target value-function in the start of training, the estimates given by the target estimate will be off due to sampling actions outside of the behaviour policy. Without being able to explore these actions, the errors themselves are propagated throughout the Q-function and usually, this error would manifest as over-estimation [Van Hasselt et al., 2016, Wang et al., 2016, Lan et al., 2020]. Therefore, it is likely that these bootstrap errors are attributable to the agent's parameter uncertainty. While this may be true intuitively, Flennerhag et al. [2020] directly try to decouple parametric uncertainty from the return uncertainty. The authors take this idea to the distributional setting, and show that in addition to return uncertainty, part of the uncertainty estimate of the Q-value, *when using TD methods*, is attributable to parametric uncertainty of the agent's policy. Although they find it to be intractable to completely decouple, we begin to explore this idea as well by asking if using the uncertainty quantified by distributional RL can be leveraged in the same as the uncertainty quantified by Bayesian neural networks or ensemble methods. A more thorough analysis of this idea is required to draw further conclusions. In particular, we note that it would be particularly interesting to compare the exploration conferred by leverage uncertainty from ensembles of distributional critics with our method below, and we leave this for future work.

Next, we introduce two novel methodological improvements to the DSAC algorithm. When taken together, these changes comprise our novel algorithm which we refer to as Extra Exploration with Distributional Critics (E2DC).

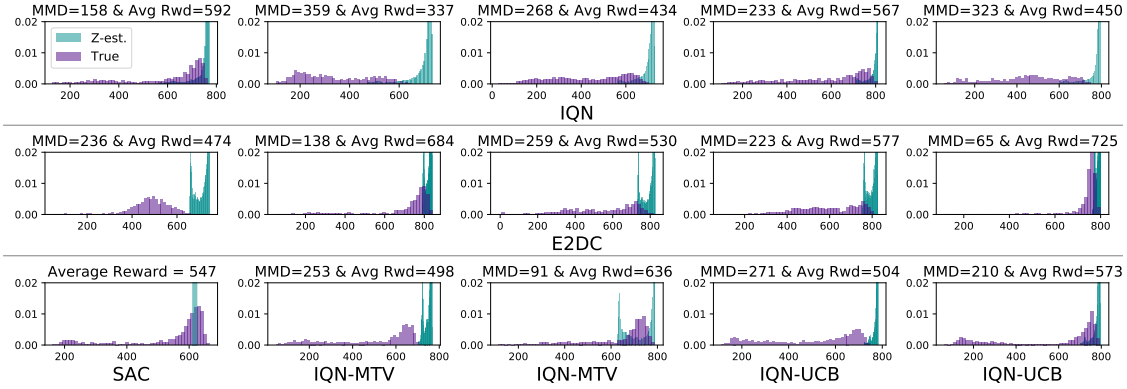


Figure 6.2: Histograms of true empirical return distributions following a trained policy (purple) compared against distributions given by a trained Z-network (cyan) on the Humanoid-v2 tasks. Each plot is a independent actor-critic pair. We highlight results from the baseline algorithms, SAC and IQN, along with our algorithm, E2DC, and its ablations, IQN-UCB and IQN-MTV. In the first two rows, we show 5 histogram plots each for IQN and E2DC, respectively, and in the third row, we show one example for SAC, followed by two each for IQN-MTV and IQN-UCB.

6.3.1 Multi-sample Target Value

In the original formulation in the SAC algorithm, the soft Bellman operator is defined for a value function, $V(s)$, which can either be learned as a separate parametrized value network distinct from the action-value network, $Q(s,a)$, or it can be approximated with a single sample from the policy and passed through the Q-function. Similarly, the value-distribution, $Z(s)$, is almost always approximated with a single sample, i.e. $Z(s) \approx Z(s, \text{sample}\{\pi(\cdot|s)\})$.

In the distributional case, we present a multi-sample approximation of $Z(s)$, obtained by ancestrally resampling from the policy and then the conditional action-value distribution. In this setup, our value-distribution can instead be approximated by the empirical distribution:

$$\begin{aligned} a_{1:K} \sim \pi(\cdot|s), \;\; Z^k_{1:M} \sim Z_\theta(s, a_k) \\ Z(s) := \frac{1}{MK} \sum_{j=1}^{M} \sum_{k=1}^{K} \delta_{Z^k_j} \end{aligned} \tag{6.13}$$

The final loss we use in the multisample quantile loss is then:

$$\mathcal{L}^{\text{MTV}}_\theta(s,a) = \mathop{\mathbb{E}}_{s,a\sim\mathcal{D}} \left[\mathop{\mathbb{E}}_{s'\sim p} \left[\frac{1}{MK} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{K} \rho^H_\tau \left(R(s,a) + \gamma Z^{\tau^k_j}_\theta(s', a'_k) - Z^{\tau_i}_\theta(s,a) \right) \right] \right], \tag{6.14}$$

where $a'_k \sim \pi(\cdot|s')$, $\{\tau_j^k\}_{j=1}^M \sim \text{Uniform}([0,1]) \quad \forall k \in 1...K$, and $\{\tau_i\}_{i=1}^N \sim \text{Uniform}([0,1])$. Note that in the case where $K = 1$, we recover the single sample estimate of the original IQN algorithm. We call this a Multi-sample Target Value (MTV) method.

Figure 6.1 provides a schematic comparison of IQN (equivalently, single-sample MTV) and multi-sample MTV. Intuitively, the additional samples contribute to parametric uncertainty in the propagation of Bellman update errors at the start of training, and leading to a better approximation of the intrinsic return uncertainty at the end of training.

6.3.2 UCB Exploration

Our method here extends the work of Tang and Agrawal [2018], which uses posterior sampling with distributional critics. We improve upon this by leveraging a distributional critic for UCB, similar to the way in which Chen et al. [2017] replaces posterior sampling used in Osband et al. [2016] with UCB exploration for further improvements. We extend the idea of continuous action space UCB exploration from Lee et al. [2020]. We call this extension IQN-UCB.

While exploring, we maintain an UCB for each action at a given state based on the current Z-distribution, $\{Z^{\tau_i}\}_{i=1}^N \sim Z_\theta(s,a)$. We define the empirical mean $\tilde{\mu}(s,a) := \frac{1}{N}\sum_i Z^{\tau_i}$ and the empirical standard deviation $\tilde{\sigma}(s,a) := \sqrt{\frac{1}{N}\sum_i (Z^{\tau_i} - \tilde{\mu})^2}$. Following Lee et al. [2020], we use this in the continuous control setting by first sampling L candidate actions for a given state, $a_t^{1:L} \sim \pi(\cdot|s_t)$. We then choose the action from the candidate set that maximizes the UCB:

$$a_t = \arg\max_{a_t^l} \{\tilde{\mu}(s_t, a_t^l) + \lambda \tilde{\sigma}(s_t, a_t^l)\}, \tag{6.15}$$

where λ is a hyperparameter to control the importance of each action's distributional critic variance.

6.3.3 Algorithm Description

Finally, we outline our algorithm E2DC, which adds the two improvements to the DSAC algorithm. The first is the MTV sample estimate which adds diversity to the

Bellman targets. During UCB exploration, we exactly want to target the action-values that have been fitted to these higher variance Z-distributions to explore further, as they are the ones with the highest parametric uncertainty in the start, but decreasing as training progresses. Furthermore we empirically validate this in the Humanoid task. While UCB itself does help, adding the MTV improves the stability of learning (Figure 6.4).

Our algorithm is similar to DSAC in implementation, which uses two parameterized Z-networks, θ_1 and θ_2 and their respective targets, $\bar{\theta}_{1,2}$. We use the minimum of the Z-samples for the gradient estimates and update the targets using a $\bar{\tau}$ parameter to control the exponential moving average (EMA) update of $\theta_{1,2}$ throughout optimization. The full pseudocode is given in Algorithm 5.

Algorithm 5 Extra Exploration with Distributional Critics

Input: θ_1, θ_2, ϕ, $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$, $D \leftarrow \{\}$, Hyperparameters $\alpha, \lambda, \bar{\tau}, N, M, K, L$,
Learning rates $\lambda_\alpha, \lambda_\phi, \lambda_\theta$
while $n \leq$ max timesteps **do**
  for each environment step **do**
    sample candidate actions $a_t^{1:L} \sim \pi_\phi(a_t|s_t)$
    select action to maximize the UCB $a_t = \arg\max_{a_t^l}\{\tilde{\mu}(s_t, a_t^l) + \lambda\tilde{\sigma}(s_t, a_t^l)\}$
    do update $s_{t+1} \leftarrow p(s_{t+1}|s_t, a_t)$, receive reward, $r_t \leftarrow r(s_t, a_t)$
    store transition in replay buffer $D \leftarrow D \cup \{(a_t, s_t, r_t, s_{t+1})\}$
  end for
  for each gradient step **do**
    sample minibatch from replay buffer D
    $\phi \leftarrow \phi - \lambda_\phi \nabla_\phi \mathcal{L}_\phi^\pi$ using $Q(s,a) = \mathbb{E}[Z_\theta(s,a)]$
    $\theta_i \leftarrow \theta - \lambda_\theta \nabla_\theta \mathcal{L}_{\theta_i}^{\text{MTV}}$ using N, M, K samples drawn from Z_θ, Bellman targets, and π_ϕ
    $\bar{\theta}_i \leftarrow (1-\bar{\tau})\bar{\theta}_i + \bar{\tau}\theta_i$, update EMA of target networks for $i \in \{1, 2\}$
    $\alpha \leftarrow \alpha - \nabla_\alpha \lambda_\alpha \mathcal{L}^\alpha$, adjust entropy coefficient towards task-specific target entropy
  end for
end while

6.4 Experiments

We test the effectiveness of E2DC on a host of benchmark tasks in continuous control and compare against both conventional and distributional state-of-the-art

RL algorithms. The baseline distributional RL algorithm we use is DSAC with an IQN loss for Z-learning, and we refer to it as IQN in our experiments for clarity. The baseline classical RL algorithm we compare against is SAC. An implementation of both baselines is provided in the code released by Ma et al. [2020], based on RLKit [Pong et al.], which we further modify to implement E2DC. For all training runs using IQN-MTV or E2DC, we set $M = 8, K = 8$, while runs using IQN or IQN-UCB, we set $M = 64, K = 1$ for a fair comparison. All runs use $N = 64$, and UCB settings $L = 12$ and $\lambda = 50$. We note that higher values for K tend to increase performance, but incur a slight computational cost.

We perform experiments using the MuJoCo suite [Todorov et al., 2012] and Box2d tasks from OpenAI Gym [Brockman et al., 2016]. All experiments are run with 3 seeds for SAC and 5 seeds for distributional methods, IQN and E2DC, unless otherwise stated. Reward curves are plotted with one standard deviation and smoothed using a 0.98 weighted EMA, unless otherwise stated. All MuJoCo agents are trained to 5 million timesteps and all Box2d agents are trained to 3 million timesteps. Additional implementation and experimental details are provided in the supplementary material.

6.4.1 Distribution Matching Study

In Table 6.2, we quantify the 1-Wasserstein Distance (i.e. Earth Mover Distance; EMD), between true returns generated by a trained policy at 5 million timesteps and those generated by the corresponding trained Z-network on actor-critic pairs learning on Humanoid-v2. For a comparison, we fix the initial state and action sampled by the policy and rollout 500 times independently under the stochastic policy and environment transition. Therefore, the returns garnered with 500 roll-outs should approximate that which is captured by the Z-network for the same initial state-action pair. We find that IQN-MTV and E2DC provide the closest matched distributions, while leading to higher average returns under the stochastic policy. Meanwhile, IQN is the worst performing algorithm when evaluating the average return given by a fully trained RL agent.

Table 6.2: Row 1: Wasserstein distance between the true distribution of returns following a trained stochastic policy, π, and the distribution given by a corresponding trained Z-network to approximate Z^π. Row 2: Average returns for different trained agents following the stochastic policy from the start.

	SAC	IQN	IQN-UCB	IQN-MTV	E2DC
EMD (W_p ; $p = 1$)	-	265 ± 75	228 ± 36	$\mathbf{164 \pm 75}$	183 ± 72
Average Return	519 ± 42	480 ± 98	551 ± 40	575 ± 63	$\mathbf{600 \pm 93}$

Figure 6.2 provides a visualization of these policies, with additional tasks summarized in the supplementary material. We qualitatively note that the true returns under the policy rollouts across all algorithms, including SAC, tend to be more bimodal than unimodal, while only the multisample variants, E2DC and IQN-MTV, are able to capture the bimodal behavior in estimating the returns.

We find that evaluation of these agents on the Humanoid-v2 task under the stochastic policy closes the gap in performance from SAC to the distributional RL algorithms. This is in contrast to the Humanoid-v2 results given in Figures 6.3 and 6.4, which use a deterministic policy to evaluate the total reward, without discounting or entropy bonus. In Figure 6.2 we see for a SAC agent (bottom-left) that while sometimes the policy achieves very low return, most of the rollouts are within the higher return mode to the right. In contrast, IQN agents (top row) also achieve low return at times, but tends to have less mass concentrated its the higher return region, despite much higher deterministic returns.

6.4.2 MuJoCo and Box2d

In Table 6.1 and Figure 6.3, we present results on the full MuJoCo Suite as well as the hardest Box2d task. The other tasks in the Box2d suite are solved easily by all algorithms, although E2DC achieves a slight gain in sample efficiency under 500K timesteps. The top row is comprised of the more difficult tasks, as measured by the dimensionality of the state and action spaces in the Humanoid-v2 and HumanoidStandup-v2 tasks as well as by the difficulty for SAC to solve these tasks. We note that these achieve a performance gain in final convergence and sample efficiency when using E2DC over all baselines. However, in the bottom row

among the easier tasks, we find that while IQN and E2DC both outperform the SAC baseline, E2DC does not improve upon IQN in Walker2d-v2 and Ant-v2.

We provide a brief intuition for why E2DC is beneficial on harder tasks. We hypothesize that this phenomenon can be attributed to easier tasks being more readily solved using only the entropy bonus for exploration. We know quantitatively, the final entropy coefficient, α, is about 2-5x times higher for each of the easier tasks than for Humanoid-v2. Additionally, both Humanoid tasks and the Swimmer-v2 task are thought to require more exploration [Gangwani et al., 2018]. When maximum-entropy RL is enough to solve the task reasonably well, using E2DC may hinder performance by focusing computational resources on exploring when there is less need. Finally, this is reflected in the sometimes improved performance of IQN over a multi-sample bound, as the former is also more 'exploitative' in Z-learning.

6.4.3 Humanoid Ablation Study

In Figure 6.4, we evaluate the two ablations, IQN-MTV and IQN-UCB, of E2DC on the Humanoid-v2 task. We find that for this task, E2DC still outperforms IQN and either ablation in average returns (right). In the left plot, we show the variance of each ablation and the baselines' Z-estimate throughout training. We find that the variance of the IQN-MTV Z-estimate remains highest in the initial stages of training, which we attribute to initial parametric uncertainty due to accumulation of Bellman errors.

We therefore posit that the additional methods we introduce accomplish the goal we want in terms of exploration using return uncertainty. In the MTV method, we see that the Z-distribution remains higher variance in the beginning of training over IQN for a short period, before converging to much lower variance towards the end of training. IQN, however, has equally high variance throughout training as the IQN algorithm with UCB exploration, except towards the end (similarly reflected in equal performance between IQN and IQN-UCB on average return, except at the end). This suggests that the E2DC method helps the most, as it explores best throughout training. We believe this is because UCB targets the highly uncertain

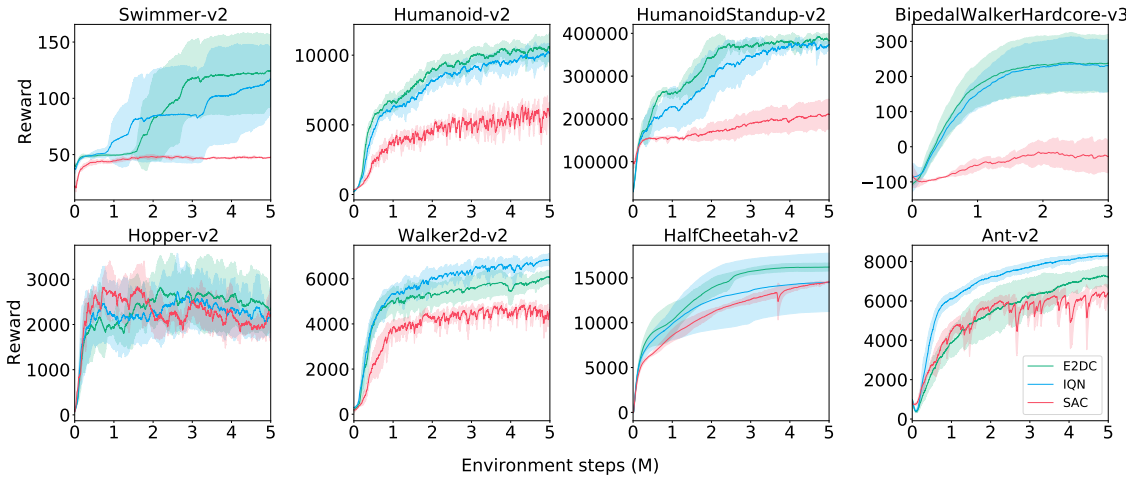


Figure 6.3: Performance on MuJoCo control suite tasks (v2) and the most difficult Box3d task (other tasks are solved easily by all algorithms). The top row are the 4 most difficult tasks for SAC, while the bottom row comprise easier tasks. Humanoid-v2 IQN and E2DC use 10 seeds (others use 5).

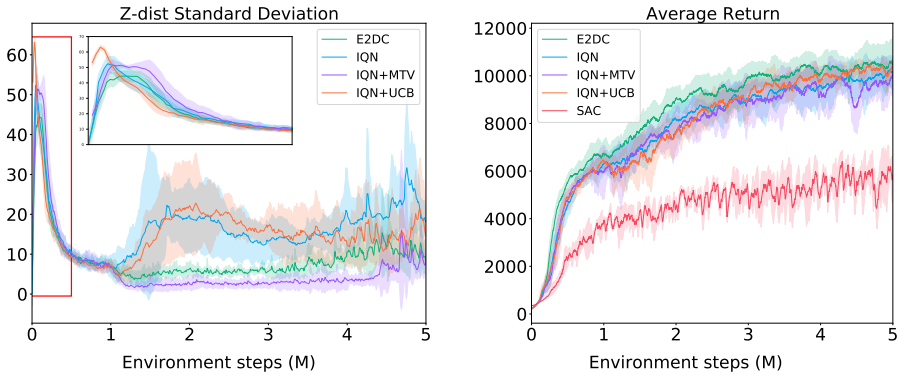


Figure 6.4: Left: Z-distribution standard deviations throughout training for IQN, E2DC, and ablations, IQN-MTV and IQN-UCB smoothed using a 0.5 EMA. The red box is zoomed in on the first 500K timesteps. Right: average returns of each method along with SAC baseline using the deterministic policy rollout for evaluation at each timestep throughout training.

actions produced by the MTV estimator only in the beginning of training, while converging better to the true return uncertainty later in training. When IQN-MTV is then used with UCB in E2DC, these initial uncertain states are better explored, leading to higher sample efficiency and overall return shown in the right plot.

6.5 Discussion

In this work, we have introduced the multi-sample target values using pooling of quantile-based networks and leveraged this mechanism for additional exploration with UCB. Our resulting algorithm, E2DC, shows performance gains in difficult continuous control tasks and improvements in distribution matching. Future work includes integrating our changes with orthogonal advances in distributional RL [Kuznetsov et al., 2020] and more expressive policies [Yue et al., 2020, Ward et al., 2019]. Due to our method potentially learning more accurate distributions of the true returns, our work here can be leveraged for specific use cases, such as risk-seeking policies in stock market trading strategies, or risk-averse learning for robotics. As with other advances in RL, the societal impact of this work is largely dependent on the RL practitioner and should be used with care in safety-critical applications.

Statement of Authorship for joint/multi-authored papers for PGR thesis
To appear at the end of each thesis chapter submitted as an article/paper

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

Title of Paper	Exploration with Multi-Sample Target Values for Distributional RL
Publication Status	☐Published ☐ Accepted for Publication ☒Submitted for Publication ☐Unpublished and unsubmitted work written in a manuscript style
Publication Details	Teng, M., Van de Panne, M., & Wood, F. (2022). Exploration with Multi-Sample Target Values for Distributional RL. *Preprint*

Student Confirmation

Student Name:	Michael Teng		
Contribution to the Paper	I came up with the idea. Frank Wood and I derived the objectives. I implemented the algorithm and experiments. Michiel van de Panne and I co-wrote the paper.		
Signature		Date	Jan 11, 2022

Supervisor Confirmation

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

Supervisor name and title:	Dr. Frank Wood, Associate Professor		
Supervisor comments:	I concur with Mr. Teng's assessment of his contributions to this work.		
Signature		Date	11 / Jan / 2022

We cross our bridges when we come to them and burn them behind us, with nothing to show for our progress except a memory of the smell of smoke, and a presumption that once our eyes watered.

— Tom Stoppard

7 Conclusion

We have applied deep sequential modelling and inference to a number of tasks both in custom domains and the specific domain of reinforcement learning. Our goal is to show different ways in which the overall framework of probabilistic machine learning could be useful towards achieving algorithmic intelligence.

In Chapter 3, we introduce BDSGD, a high-throughput algorithm for training deep neural networks on parallel computing clusters. This algorithm uses amortised inference in a deep generative model to perform joint posterior predictive inference of mini-batch gradient computation times in a compute cluster specific manner. Specifically, our algorithm mitigates the straggler effect in synchronous, gradient-based optimisation by choosing an optimal cutoff beyond which mini-batch gradient messages from slow workers are ignored. The principle novel contribution and finding of this work goes beyond this by demonstrating that using the predicted run-times from a generative model of cluster worker performance improves over the static-cutoff prior art, leading to higher gradient computation throughput on large compute clusters. In our experiments we show that eagerly discarding the mini-batch gradient computations of stragglers not only increases throughput but sometimes also increases the overall rate of convergence as a function of wall-clock time by virtue of eliminating idleness. In order for this to work, we rely

on amortising inference over the entire system faster than any local computation of cluster nodes.

In Chapter 4, we introduce a novel objective for training deep generative time-series models with discrete latent variables for which supervision is only sparsely available. This instance of semi-supervised learning is challenging for the existing methods, since the exponential number of possible discrete latent configurations results in high variance gradient estimators. We first overcome this problem by combining the reweighted wake-sleep algorithm with a standard semi-supervised objective extended to the time-series setting, and we then improve upon this by introducing a unified objective that no longer separates unsupervised and supervised learning signals. We call the resulting approach caffeinated wake-sleep (CWS) to emphasise its additional dependence on real data. We demonstrate its effectiveness with experiments on MNIST, handwriting data, and fruit fly trajectory data. We showed that the added training signal in the latent space helped learning, generative capability, and uncertainty quantification for fly behaviour.

In Chapter 5, we turn to the RL setting, and we investigate the benefits and drawbacks of existing policy gradient algorithms based on their objectives derived from the forward and reverse KL divergences. These objectives follow from the "RL as inference" framework and reframe the control problem as an inference task. We show that many existing methods are compatible with this framework. In Chapter 6, we learn a value-network that predicts the full distribution of the returns for a given state, modelled via a quantile-based critic. This approach has been successfully integrated into common RL methods for continuous control, giving rise to algorithms such as Distributional Soft Actor-Critic (DSAC). We improve this algorithm to investigate an orthogonal approach towards improving decision making in agents, and our implementation on top of SAC shows that it is still compatible with the RLAI framework.

In general, we are interested in showing the impact of sequential generative modelling, and we present a series of arguments in this dissertation for different situations in which they may be useful. Although not presented here, we conclude

by saying that a clear future direction for these works is to drill deeper into *model-based* reinforcement learning, where we simultaneously learn a model of the environment while solving the control task within our model. This directly relates to the goal of realistic agents that can solve challenging real-world tasks with sample efficiency and minimal supervision.

Bibliography

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

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

Alessandro Achille, Tom Eccles, Loic Matthey, Chris Burgess, Nicholas Watters, Alexander Lerchner, and Irina Higgins. Life-long disentangled representation learning with cross-domain latent homologies. *Advances in Neural Information Processing Systems*, 31, 2018.

Samuel Ainsworth, Kendall Lowrey, John Thickstun, Zaid Harchaoui, and Siddhartha Srinivasa. Faster policy learning with continuous-time gradients, 2021.

Emre Aksan, Fabrizio Pece, and Otmar Hilliges. Deepwriting: Making digital ink editable via deep generative modeling. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 205. ACM, 2018.

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

Idan Amir, Tomer Koren, and Roi Livni. Sgd generalizes better than gd (and regularization doesnŠt help). In *Conference on Learning Theory*, pages 63–92. PMLR, 2021.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016.

Christophe Andrieu and Arnaud Doucet. Particle filtering for partially observed gaussian state space models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(4):827–836, 2002.

Jimmy Ba, Ruslan R Salakhutdinov, Roger B Grosse, and Brendan J Frey. Learning wake-sleep recurrent attention models. In *Advances in Neural Information Processing Systems*, pages 2593–2601, 2015.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

Matthew J Beal, Zoubin Ghahramani, and Carl Edward Rasmussen. The infinite hidden markov model. *Advances in neural information processing systems*, 1:577–584, 2002.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

J Bernardo, J Berger, APAFMS Dawid, A Smith, et al. Regression and classification using gaussian process priors. *Bayesian statistics*, 6:475, 1998.

Dimitri P Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.

Christophe Biernacki, Gilles Celeux, and Gérard Govaert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE transactions on pattern analysis and machine intelligence*, 22(7):719–725, 2000.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. *arXiv preprint arXiv:1406.2751*, 2014.

Matthew Botvinick, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement learning, fast and slow. *Trends in cognitive sciences*, 23(5):408–422, 2019.

Matthew M Botvinick, Yael Niv, and Andew G Barto. Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective. *Cognition*, 113(3):262–280, 2009.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in beta-vae. *arXiv preprint arXiv:1804.03599*, 2018.

Alan Chan, Hugo Silva, Sungsu Lim, Tadashi Kozuno, A. Rupam Mahmood, and Martha White. Greedification operators for policy optimization: Investigating forward and reverse kl divergences, 2021.

Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981*, 2016.

Richard Y Chen, Szymon Sidor, Pieter Abbeel, and John Schulman. Ucb exploration via q-ensembles. *arXiv preprint arXiv:1706.01502*, 2017.

Yanping Chen, Bing Hu, Eamonn Keogh, and Gustavo EAPA Batista. Dtw-d: time series semi-supervised learning from a single example. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 383–391. ACM, 2013.

Yinlam Chow, Brandon Cui, MoonKyung Ryu, and Mohammad Ghavamzadeh. Variational model-based policy optimization. *arXiv preprint arXiv:2006.05443*.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

Wesley Chung, Valentin Thomas, Marlos C. Machado, and Nicolas Le Roux. Beyond variance reduction: Understanding the true impact of baselines on policy optimization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1999–2009. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/chung21a.html`.

Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. Better exploration with optimistic actor critic. In *Advances in Neural Information Processing Systems*, pages 1787–1798, 2019.

Thomas R Clandinin and Lisa M Giocomo. Neuroscience: Internal compass puts flies in their place. *Nature*, 521(7551):165, 2015.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization, 2018.

Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. Scale out for large minibatch sgd: Residual network training on imagenet-1k with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291*, 2017.

Adriá Colomé and Carme Torras. Dual reps: A generalization of relative entropy policy search exploiting bad experiences. *IEEE Transactions on Robotics*, 33(4):978–985, 2017.

Adrien Corenflos, James Thornton, George Deligiannidis, and Arnaud Doucet. Differentiable particle filtering via entropy-regularized optimal transport. In *International Conference on Machine Learning*, pages 2100–2111. PMLR, 2021.

Christopher J Cueva and Xue-Xin Wei. Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. *arXiv preprint arXiv:1803.07770*, 2018.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018a.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. *A survey on policy search for robotics*. now publishers, 2013.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control, 2016.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2019.

Tolga Ergen and Suleyman S Kozat. Online training of lstm networks in distributed systems for variable length data sequences. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

Eyrun Eyjolfsdottir, Steve Branson, Xavier P Burgos-Artizzu, Eric D Hoopfer, Jonathan Schor, David J Anderson, and Pietro Perona. Detecting social actions of fruit flies. In *European Conference on Computer Vision*, pages 772–787. Springer, 2014.

Eyrun Eyjolfsdottir, Kristin Branson, Yisong Yue, and Pietro Perona. Learning recurrent representations for hierarchical behavior modelling. In *International Conference on Learning Representations*, 2016.

Matthew Fellows, Anuj Mahajan, Tim G. J. Rudner, and Shimon Whiteson. Virel: A variational inference framework for reinforcement learning, 2018.

Matthew Fellows, Anuj Mahajan, Tim GJ Rudner, and Shimon Whiteson. Virel: A variational inference framework for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 7120–7134, 2019.

Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine learning*, 32(1):41–62, 1998.

Sebastian Flennerhag, Jane X Wang, Pablo Sprechmann, Francesco Visin, Alexandre Galashov, Steven Kapturowski, Diana L Borsa, Nicolas Heess, Andre Barreto, and Razvan Pascanu. Temporal difference uncertainties as a signal for exploration. *arXiv preprint arXiv:2010.02255*, 2020.

David Forsyth and Jean Ponce. *Computer vision: A modern approach.* Prentice hall, 2011.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax-a differentiable physics engine for large scale rigid body simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL `https://openreview.net/forum?id=VdvDlnnjzIN`.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

Alexandre Galashov, Siddhant M Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in kl-regularized rl. *arXiv preprint arXiv:1905.01240*, 2019.

Tanmay Gangwani, Qiang Liu, and Jian Peng. Learning self-imitating diverse policies. *arXiv preprint arXiv:1805.10309*, 2018.

Jonas Geiping, Micah Goldblum, Phillip E Pope, Michael Moeller, and Tom Goldstein. Stochastic training is not necessary for generalization. *arXiv preprint arXiv:2109.14119*, 2021.

Dibya Ghosh, Marlos C. Machado, and Nicolas Le Roux. An operator view of policy gradient methods, 2020.

Adam Golinski, Frank Wood, and Tom Rainforth. Amortized monte carlo integration. In *International Conference on Machine Learning*, pages 2309–2318. PMLR, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

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

Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2005.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv preprint arXiv:1711.00123*, 2017.

Thomas Griffiths, Charles Kemp, and Joshua Tenenbaum. Bayesian models of cognition. 2008.

Thomas Griffiths, Joseph Austerweil, and Vincent Berthiaume. Comparing the inductive biases of simple neural networks and bayesian models. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 34, 2012.

Shixiang Gu, Zoubin Ghahramani, and Richard E Turner. Neural adaptive sequential monte carlo. *arXiv preprint arXiv:1506.03338*, 2015.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies, 2017.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 1851–1860. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018b.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018c.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2017.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1729–1739, 2017.

Matthew D Hoffman and Matthew J Johnson. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, 2016.

Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*, pages 2117–2126. PMLR, 2018.

Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. A closer look at deep policy gradients. *arXiv preprint arXiv:1811.02553*, 2018.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Hilbert J Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. *Machine learning*, 87(2):159–182, 2012.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013a.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013b.

Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

David C Knill and Alexandre Pouget. The bayesian brain: the role of uncertainty in neural coding and computation. *TRENDS in Neurosciences*, 27(12):712–719, 2004.

Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE, 2004.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

Rahul G Krishnan, Uri Shalit, and David Sontag. Structured Inference Networks for Nonlinear State Space Models. In *AAAI*, pages 2101–2109, 2017.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*, pages 5556–5566. PMLR, 2020.

Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*, 2020.

T.A. Le, A.G. Baydin, and F. Wood. Inference Compilation and Universal Probabilistic Programming. In *20th International Conference on Artificial Intelligence and Statistics*, 2017a.

Tuan Anh Le, Maximilian Igl, Tom Rainforth, Tom Jin, and Frank Wood. Auto-encoding sequential monte carlo. *arXiv preprint arXiv:1705.10306*, 2017b.

Tuan Anh Le, Adam R Kosiorek, N Siddharth, Yee Whye Teh, and Frank Wood. Revisiting reweighted wake-sleep. *arXiv preprint arXiv:1805.10469*, 2018.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. *arXiv preprint arXiv:2007.04938*, 2020.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6): 335–346, 2020.

Marcus Liwicki and Horst Bunke. Iam-ondb-an on-line english sentence database acquired from handwritten text on a whiteboard. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 956–961. IEEE, 2005.

Xubo Lyu and Mo Chen. Ttr-based rewards for reinforcement learning with implicit model priors. *arXiv preprint arXiv:1903.09762*, 2019.

Xiaoteng Ma, Qiyuan Zhang, Li Xia, Zhengyuan Zhou, Jun Yang, and Qianchuan Zhao. Distributional soft actor critic for risk sensitive learning. *arXiv preprint arXiv:2004.14547*, 2020.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Chris J Maddison, John Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Teh. Filtering variational objectives. In *Advances in Neural Information Processing Systems*, pages 6573–6583, 2017.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

Borislav Mavrin, Hengshuai Yao, Linglong Kong, et al. Exploration using distributional rl and ucb. 2018.

Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*, volume 38. M. Dekker New York, 1988.

Brendan McMahan and Matthew Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems*, pages 2915–2923, 2014.

Richard J Meinhold and Nozer D Singpurwalla. Understanding the kalman filter. *The American Statistician*, 37(2):123–127, 1983.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

Luke Metz, C. Daniel Freeman, Samuel S. Schoenholz, and Tal Kachman. Gradients are not all you need, 2021.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

Andriy Mnih and Danilo J Rezende. Variational inference for monte carlo objectives. *arXiv preprint arXiv:1602.06725*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Efficient exploration with double uncertain value networks. *arXiv preprint arXiv:1711.10789*, 2017.

William H Montgomery and Sergey Levine. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016, 2016.

Christian A Naesseth, Scott W Linderman, Rajesh Ranganath, and David M Blei. Variational sequential monte carlo. *arXiv preprint arXiv:1705.11140*, 2017.

Gerhard Neumann et al. Variational inference for policy search in changing situations. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pages 817–824, 2011.

Brendan O'Donoghue, Ian Osband, and Catalin Ionescu. Making sense of reinforcement learning and probabilistic inference. *arXiv preprint arXiv:2001.00805*, 2020.

Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz E Khan, Anirudh Jain, Runa Eschenhagen, Richard E Turner, and Rio Yokota. Practical deep learning with bayesian principles. *Advances in neural information processing systems*, 32, 2019.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*, 2016.

Thomas Parr, Geraint Rees, and Karl J Friston. Computational neuropsychology and bayesian inference. *Frontiers in human neuroscience*, 12:61, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett,

editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

Alexandre Piché, Valentin Thomas, Cyril Ibrahim, Yoshua Bengio, and Chris Pal. Probabilistic planning with sequential monte carlo methods. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByetGn0cYX.

Vitchyr Pong, M Dalal, S Lin, and A Nair. Rlkit: Reinforcement learning framework and algorithms implemented in pytorch, 2019. *URL https://github. com/vitchyr/rlkit*.

Tom Rainforth, Adam R. Kosiorek, Tuan Anh Le, Chris J. Maddison, Maximilian Igl, Frank Wood, and Yee Whye Teh. Tighter variational bounds are not necessarily better. In *International Conference on Machine Learning*, 2018.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

Sid Reddy, Anca Dragan, and Sergey Levine. Where do you think you're going?: Inferring beliefs about dynamics from behavior. *Advances in Neural Information Processing Systems*, 31, 2018.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Daniel Ritchie, Paul Horsfall, and Noah D Goodman. Deep amortised inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016.

JP Royston. Algorithm AS 177: Expected normal order statistics (exact and approximate). *Journal of the royal statistical society. Series C (Applied statistics)*, 31 (2):161–165, 1982.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242, 2016.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Philipp Schwartenbeck, Thomas HB FitzGerald, Christoph Mathys, Ray Dolan, Friedrich Wurst, Martin Kronbichler, and Karl Friston. Optimal inference with suboptimal models: addiction and active bayesian inference. *Medical hypotheses*, 84 (2):109–117, 2015.

Adam Ścibior, Vaden Masrani, and Frank Wood. Differentiable particle filtering without modifying the forward pass. *arXiv preprint arXiv:2106.10314*, 2021.

N Siddharth, Brooks Paige, Alban Desmaison, Jan-Willem van de Meent, Frank Wood, Noah D Goodman, Pushmeet Kohli, and Philip HS Torr. Learning disentangled representations in deep generative models. 2016.

N. Siddharth, T Brooks Paige, Jan-Willem Van de Meent, Alban Desmaison, Noah Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr. Learning disentangled representations with semi-supervised deep generative models. In *Advances in Neural Information Processing Systems*, pages 5925–5935, 2017.

Samuel Smith, Erich Elsen, and Soham De. On the generalization benefit of noise in stochastic gradient descent. In *International Conference on Machine Learning*, pages 9058–9067. PMLR, 2020.

Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

H Francis Song, Abbas Abdolmaleki, Jost Tobias Springenberg, Aidan Clark, Hubert Soyer, Jack W Rae, Seb Noury, Arun Ahuja, Siqi Liu, Dhruva Tirumala, et al. V-mpo: On-policy maximum a posteriori policy optimization for discrete and continuous control. *arXiv preprint arXiv:1909.12238*, 2019.

Yang Song and Stefano Ermon. Generative modelling by estimating gradients of the data distribution. *arXiv preprint arXiv:1907.05600*, 2019.

Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *arXiv preprint arXiv:2006.09011*, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 2011.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

Yunhao Tang and Shipra Agrawal. Exploration by distributional reinforcement learning. *arXiv preprint arXiv:1805.01907*, 2018.

Terence Tao. *An introduction to measure theory*, volume 126. American Mathematical Society Providence, RI, 2011.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems*, pages 1195–1204, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pages 2627–2636, 2017.

George Tucker, Surya Bhupatiraju, Shixiang Gu, Richard E Turner, Zoubin Ghahramani, and Sergey Levine. The mirage of action-dependent baselines in reinforcement learning. *arXiv preprint arXiv:1802.10031*, 2018.

Jan-Willem Van De Meent, Brooks Paige, David Tolpin, and Frank Wood. Black-box policy search with probabilistic programs. 2016.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

A_P Varga and Roger K Moore. Hidden markov model decomposition of speech and noise. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 845–848. IEEE, 1990.

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

Sharan Vaswani, Olivier Bachem, Simone Totaro, Robert Mueller, Matthieu Geist, Marlos C Machado, Pablo Samuel Castro, and Nicolas Le Roux. A functional mirror ascent view of policy gradient methods with function approximation. *arXiv preprint arXiv:2108.05828*, 2021.

Jack M Wang, David J Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2007.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003, 2016.

Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving exploration in soft-actor-critic with normalizing flows policies. *arXiv preprint arXiv:1906.02771*, 2019.

Andrew Warrington, Jonathan W Lavington, Adam Scibior, Mark Schmidt, and Frank Wood. Robust asymmetric learning in pomdps. In *International Conference on Machine Learning*, pages 11013–11023. PMLR, 2021.

Li Wei and Eamonn Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 748–753. ACM, 2006.

Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, 2008.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017a.

Yang You, Zhao Zhang, C Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. *CoRR, abs/1709.05011*, 2017b.

Yuguang Yue, Zhendong Wang, and Mingyuan Zhou. Implicit distributional reinforcement learning. *arXiv preprint arXiv:2007.06159*, 2020.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems*, pages 685–693, 2015.

Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, USA, 2010.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.