Probabilistic Machine Learning: Methods and Applications to Continuous Control

Leonard Hasenclever
St Peter's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2018

Acknowledgements

First and foremost, I am indebted to my supervisor, Professor Yee Whye Teh, for his support and advice during my time at Oxford. It has been inspiring to work with one of the leading researchers in Bayesian machine learning and I have gained an astonishing amount of knowledge about machine learning and statistics, although much still remains mysterious to me. I also gratefully knowledge the support of the Engineering and Physical Sciences Research Council (EPSRC) through the Oxford-Warwick Statistics Programme, of which I was fortunate enough to be part. I also thank DeepMind for giving me the opportunity to work with so many brilliant researchers and access to computational resources without which the continuous control part of this thesis would have been impossible.

During my time at Oxford I have met many inspiring people whose friendship I value tremendously, among them fellow DPhil students at the Department of Statistics as well as other graduate students at St Peter's. Thanks for many interesting and stimulating discussions, your company and support. I am equally grateful to my friends from Cambridge and from Germany.

I am also grateful to my co-authors. In no particular order, Xiaoyu Lu, Valerio Perrone, Sebastian Vollmer, Stefan Webb and Thibaut Lienart at Oxford; Max Welling, Jakub Tomczak and Rianne van den Berg at the University of Amsterdam where I spend a few months in 2017; and finally, my co-authors at DeepMind: Josh Merel, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Nicolas Heess, Wojciech Czarnecki, Siddhant Jayakumar, Max Jaderberg, Jonathan Schwarz, Guillaume Desjardins, Charles Blundell, Balaji Lakshminarayanan and Razvan Pascanu. I am particularly grateful to Nicolas Heess for his patient explanations while I tried to grapple with Reinforcement Learning.

Finally, I would like to thank my family and my girlfriend Jackie, for their support and faith in me during my DPhil and putting up with me during the writing of this thesis. Thank you!

Abstract

Probabilistic inference is at the core of many recent advances in machine learning. Unfortunately, exact inference is intractable in all but the simplest models. Thus, approximate inference methods are required use probabilistic methods for large and complex models. Broadly speaking, there are two different paradigms for approximate inference, sampling methods and variational methods. Sampling methods attempt to construct approximate samples from the target distribution, which can then be used to approximate expectations with respect to the posterior. Variational methods instead rephrase inference as an optimisation problem and form parametric approximations to the target distribution.

In this thesis, we present contributions to sampling methods and variational methods with a focus on scalability. Firstly, we introduce a novel sampling technique based on Hamiltonian Monte Carlo that uses a relativistic kinetic energy to improve robustness to hyperparameters. We then describe a novel algorithm for distributed Bayesian learning based on expectation propagation techniques. In addition, we present a novel normalising flow that can be used to form more flexible variational approximations within variational inference.

We then describe two applications of probabilistic thinking and variational techniques to the field a continuous control. Firstly, we describe how reinforcement learning can be viewed as probabilistic inference and introduce a novel algorithm for learning priors in reinforcement learning leading to substantial improvement in learning speed and final performance in certain settings. Lastly, we describe a probabilistic model that can be used to compress thousands of expert policies trained to reproduce motion capture data into one model that is capable of one-shot imitation. We further demonstrate that it is possible to reuse our model, resulting in naturalistic movements on challenging control tasks.

Publications

This thesis is an integrated thesis and most chapters are publications at machine learning conferences or in machine learning journals. At the end of each chapter we include a description of my contribution. For convenience we detail the relevant publications here. * denotes equal contribution.

Chapter 3 contains

Lu*, X., Perrone*, V., **Hasenclever, L.**, Teh, Y.W. and Vollmer, S., 2017, April. Relativistic Monte Carlo. In Artificial Intelligence and Statistics (pp. 1236-1245).

Chapter 4 contains

Hasenclever, L., Webb, S., Lienart, T., Vollmer, S., Lakshminarayanan, B., Blundell, C. and Teh, Y.W., 2017. Distributed Bayesian learning with stochastic natural gradient expectation propagation and the posterior server. The Journal of Machine Learning Research, 18(1), pp.3744-3780.

Chapter 5 contains

van den Berg*, R., **Hasenclever*, L.**, Tomczak, J.M. and Welling, M., 2018. Sylvester normalizing flows for variational inference. Conference on Uncertainty in Artificial Intelligence, Oral presentation.

Chapter 7 contains results from

Galashov, A., Jayakumar, S., **Hasenclever, L.**, Tirumala, D., Schwarz, J., Desjardins, G., Czarnecki, W., Teh, Y.W., Pascanu, R. and Heess, N., 2019. Information Asymmetry in KL-regularized RL. International Conference on Learning Representations.

Chapter 8 contains

Merel*, J., **Hasenclever*, L.**, Galashov, A., Ahuja, A., Pham, V., Wayne, G., Teh, Y.W. and Heess, N., 2018. Neural Probabilistic Motor Primitives for Humanoid Control. International Conference on Learning Representations.

Contents

Chapter 1

Introduction

In recent years, machine learning has led to great advances in natural language processing and machine translation (e.g. Bahdanau et al. [2014]), speech recognition and synthesis (e.g. [Oord et al., 2016]), computer vision (e.g Krizhevsky et al. [2012]), reinforcement learning (e.g. Mnih et al. [2015]), and many other areas. Probabilistic models in general and Bayesian methods in particular offer rich modelling tools and a principled, coherent way to reason about uncertainty. Bayesian methods also naturally help to prevent overfitting. Despite these attractive properties, most of the recent successes in machine learning have used empirical risk minimisation or maximum likelihood. This tendency is due to the punitive cost of posterior inference. Learning becomes the problem of computing the posterior distribution over model parameters, which is intractable in all but the easiest models.

When exact posterior inference is impossible, we have to resort to approximate methods. Approximate methods for Bayesian inference broadly fall into two different categories: sampling methods and variational methods.

Sampling methods aim to produce samples from the posterior distribution to calculate expectations of statistics of interest. Examples include Markov chain Monte Carlo (MCMC) methods (see Roberts and Rosenthal [2004] for a good review), and Sequential Monte Carlo (SMC) methods [Del Moral et al., 2006], as well as combinations of both [Andrieu et al., 2010]. There is a lot of ongoing work to scale up and parallelise sampling methods to high-dimensional settings and large data sets. Recently stochastic gradient Markov chain Monte Carlo (SGMCMC), a class of approximate dynamics based MCMC methods that uses minibatches of data, has been introduced [Welling and Teh, 2011]. These methods can be understood as discretisations of stochastic differential equations (SDEs) whose invariant distribution is the target distribution. Crucially, only a minibatch of data is used per iteration, which dramatically reduces the per-iteration cost. However, some recent results suggest

that, if high accuracy posterior estimates are required, the overall computational cost remains similar to standard sampling methods [Nagapetyan et al., 2017].

Variational methods use a different paradigm (see Wainwright and Jordan [2008] for an excellent introduction). Instead of trying to produce approximate samples from the exact posterior distribution, inference is phrased as an optimisation problem. Variational methods approximate the posterior distribution with a distribution from a tractable, parametric family. The optimal parameters are determined by an optimisation problem. Examples of variational methods include expectation propagation [Minka, 2001a] and variational inference. Variation inference forms a parametric approximation q to the posterior p by minimizing the forward Kullback-Leibler divergence $\mathsf{KL}(q\|p)$. Expectation propagation instead leverages the factorisation of the model to form an exponential family approximation to each factor and refines it iteratively.

This thesis is an integrated thesis: most of it is in the form of self-contained papers. At the end of each section containing a paper there is a form describing my contributions to the paper. The thesis consists of two parts. Part I focuses on methodology for probabilistic machine learning, mostly through a Bayesian lens, while part II describes applications to reinforcement learning.

Part I begins in chapter 2 with a brief introduction to probabilistic machine learning and introduces the fundamental problem of (Bayesian) inference. We then review three different strands of inference methods: sampling methods, variational inference, and expectation propagation.

In the next chapters, we discuss three approaches to the inference problem. Chapter 3 contains Lu et al. [2017], which introduces Relativistic HMC, a new class of MCMC algorithms, and the corresponding stochastic gradient MCMC algorithm. Relativistic HMC introduces a maximum velocity in HMC and improves robustness. In addition, it has interesting connections with widely used optimizers such as Adam [Kingma and Ba, 2015]. Chapter 4 contains Hasenclever et al. [2017a], which introduces stochastic natural gradient expectation propagation (SNEP), an algorithm that combines expectation propagation and stochastic gradient MCMC methods to enable distributed Bayesian learning. Chapter 5 focuses on variational inference and contains van den Berg et al. [2018], which introduces a novel normalising flow for more flexible variational distributions.

During an internship at DeepMind, I developed an interest in reinforcement learning, particularly with continuous action spaces. This field is also known as continuous

control. Part II contains two applications of probabilistic thinking to continuous control. In chapter 7, we introduce reinforcement learning and how it can be understood as a probabilistic inference problem. Next, we present a method to learn a prior in continuous control and present experimental evidence of the benefit of doing so. We also present exploratory results with priors with latent variables. Chapter 8 contains Merel et al. [2019], which uses a similar architecture to compress thousands of single skill expert policies trained on motion capture data into a single model. It is now relatively straightforward to train agents to reproduce a single motion capture trajectory but combining many such experts was substantially more challenging. Our architecture, named Neural Probabilistic Motor Primitives (NPMP), is capable of remarkable one-shot imitation. Furthermore, it is possible to reuse NPMP as a low-level controller for challenging control tasks resulting in naturalistic movements. We end in a brief conclusion and discussion of possible future research directions in chapter 9.

Part I

Methods

Chapter 2

A Brief Introduction to Bayesian Inference

A self-driving car should know how certain it is that an object on the road is not a pedestrian. Similarly, machine learning systems aiding medical practitioners should quantify their uncertainty about diagnoses. More generally, in almost all applications that leverage machine learning models to aid decision making uncertainty is crucial.

Jaynes [2003] invokes Cox's theorem [Cox, 1946] to argue that probability is essentially[1] the only principled way to reason about degrees of belief of uncertain propositions.

In this thesis, we mostly view machine learning through the lens of Bayesian inference. In Bayesian machine learning, uncertainty about model parameters is expressed through probability distributions. A *prior distribution* $p(\theta)$ over model parameters θ encapsulates prior knowledge about the problem. The *likelihood* $p(\mathcal{D}|\theta)$ is the probability of data, given θ. Having observed the data, we can update our beliefs about the model parameters $p(\theta|\mathcal{D})$. This distribution is known as the *posterior* and is the primary object of interest in Bayesian machine learning. By Bayes' rule the posterior can be written as:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta} \tag{2.1}$$

The posterior expresses our beliefs about the model parameters after observing the data $\mathcal{D}$ and all other quantities of interest can be derived from it. For example, predictions will generally take the form of expectations of test functions with respect to the posterior $\mathbb{E}_{p(\theta|\mathcal{D})}[f(\theta)]$.

Unfortunately, although the posterior update in (2.1) looks deceptively simple, it generally is very hard to calculate. The normalisation constant in the denominator

[1] Up to monotonic transformations.

is generally analytically intractable in all but the most simple models. In addition, even if we knew the normalisation constant, calculating expectation with respect to the posterior would still be intractable in general. The rest of this chapter reviews the two major strands of methods that address these problems: sampling methods and variational methods. In chapters 3, 4, and 5 we present contributions to both of these approaches.

2.1 Sampling Methods

In this section, we give a brief overview of sampling methods for probabilistic inference.

2.1.1 Classical Methods of Integration

The problem of inference is intimately related to the more general problem of integration. Let us consider the general problem of calculating expectations of a test function $\phi : \mathcal{S} \to \mathbb{R}$ defined on some space $\mathcal{S}$ with respect to a distribution π on $\mathcal{S}$:

$$I = \mathbb{E}_\pi \left[\phi(X)\right].$$

One approach to this problem is to approximate I by evaluating ϕ at some points $x^{(i)} \in \mathcal{S}$ and forming a weighted sum:

$$I_N = \frac{1}{N} \sum_{i=1}^{N} w_i(x^{(i)})\phi(x^{(i)}). \tag{2.2}$$

There is a range of classic methods for numerical integration based on choosing deterministic points $x^{(i)}$, typically on a grid. Examples include the trapezoid rule, Simpson's rule, and Gauss-Hermite quadrature. These methods work very well for low-dimensional integrals, but generally scale badly with the dimensionality d of the problem: the rate of convergence of such methods is $\mathcal{O}(N^{-k/d})$ where $k > 0$ is a constant depending on the specific quadrature rule (e.g. Caflisch [1998]).

Thus, for moderately high dimensional problems, the convergence rate can be very slow indeed. The potentially very slow convergence of classical methods motivates methods based on sampling:

$$x^{(i)} \overset{i.i.d}{\sim} \pi, \qquad w_i = 1 \tag{2.3}$$

This method is known as Monte Carlo integration. The Central Limit Theorem implies that Monte Carlo integration converges at a rate of $\mathcal{O}\left(N^{-1/2}\right)$ independent

of d (e.g. Robert and Casella [2004]). Monte Carlo integration is an attractive choice for high dimensional problems if we are able to sample from π. Unfortunately, this is generally not the case in Bayesian inference. There exist a number of methods that attempt to address this problem, either by sampling from a similar distribution and adjusting the weights for the mismatch, or attempting to approximately sample from π.

2.1.2 Importance Sampling

Importance sampling is a simple method to help evaluate integrals of the form $\mathbb{E}_\pi[\phi(X)]$ when we cannot sample from π, but can evaluate its density $\pi(x)$. It relies on the following identity:

$$\mathbb{E}_\pi[\phi(X)] = \mathbb{E}_q\left[\frac{\pi(X)}{q(X)}\phi(X)\right], \tag{2.4}$$

provided that q has a larger support than π, i.e. $\pi(x) > 0 \Rightarrow q(x) > 0$. q is typically called a proposal distribution and we assume we can sample from it. The ratio $\frac{\pi(X)}{q(X)}$ is known as an importance weight. Importance sampling proceeds as follows:

$$x^{(i)} \overset{i.i.d}{\sim} \pi, \qquad w_i(x^{(i)}) = \frac{\pi(x^{(i)})}{q(x^{(i)})}. \tag{2.5}$$

The strong law of large numbers implies that:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{\pi(x^{(i)})}{q(x^{(i)})}\phi(x^{(i)}) \xrightarrow{a.s.} \mathbb{E}_\pi[\phi(X)], \quad \text{as } N \to \infty. \tag{2.6}$$

In principle, any proposal distribution q such that $\pi(x) > 0 \Rightarrow q(x) > 0$ defines a consistent estimator. However, different proposal distributions will lead to estimators with different variances. One proxy to assess different proposals is the effective sample size:

$$ESS = \frac{\left(\sum_{i=1}^{N} w_i\left(x^{(i)}\right)\right)^2}{\sum_{i=1}^{N} w_i\left(x^{(i)}\right)^2} \tag{2.7}$$

The ESS is bounded above by N by the quadratic mean arithmetic mean inequality, achieved if all importance weights are equal to one, i.e. $q \equiv \pi$, and bounded below by 1, if only one sample contributes. Other notions of effective sample size exist for Markov Chain Monte Carlo algorithms.

Classic importance sampling uses a fixed proposal distribution q. However, it can be difficult to construct a good proposal distribution q. To alleviate this problem, there is a substantial body of work on adaptive importance sampling [Cappé et al., 2004, Cappé et al., 2008, Liu, 2008, Cornuet et al., 2012, Lu et al., 2018].

2.1.3 Markov Chain Monte Carlo Methods

Another strategy to approximate $\mathbb{E}_\pi[\phi(X)]$ is to develop approximate methods of sampling from π. One popular class of algorithms for this is known as Markov chain Monte Carlo (MCMC) methods. These methods construct a Markov chain whose equilibrium distribution is the distribution from which we wish to sample. We briefly review standard algorithms in the next sections. For a more detailed review of MCMC methods, see Robert and Casella [2004], Green et al. [2015].

2.1.4 Markov Chains and Detailed Balance

Consider a system with random states $X_t,\ t = 0, \ldots$ that evolves in discrete time. Such a system is a Markov chain if its transition dynamics only depend on the current state $X_t = x_t$. More formally:

$$p(X_{t+1} = x_{t+1}|X_0 = x_0, \ldots, X_t = x_t) = p(X_{t+1} = x_{t+1}|X_t = x_t) \tag{2.8}$$

The joint distribution of $X_t, t \in \mathbb{N}$ is fully specified by its initial distribution $p(X_0)$ and its transition dynamics $p(X_{t+1} = x_{t+1}|X_t = x_t)$. An important special case are Markov chains that are stationary (or time-homogeneous), i.e. Markov chains which have the same transition dynamics independent of t. In this case, the transition dynamics are often referred to as a transition kernel $T(x \to y) = p(X_{t+1} = y|X_t = x)$ for all t.

For MCMC, we are interested in Markov chains that converge to an equilibrium or stationary distribution regardless of initial distribution. A stationary distribution π must be invariant under the transition kernel:

$$\pi(y) = \int_{\mathcal{X}} T(x \to y)\pi(x)dx \tag{2.9}$$

However, the existence of a stationary distribution does not guarantee its uniqueness or that the Markov chain will converge to a stationary distribution. A sufficient condition for uniqueness and convergence is ergodicity: A Markov chain is ergodic if it is irreducible, aperiodic, and positive recurrent. Irreducibility means that it is possible to transition from any state to any other state in finite time, that is:

$$\forall x,\ x'\ \ \exists n \text{ such that } p(X_n = x'|X_0 = x) > 0 \tag{2.10}$$

A state x is aperiodic if for all sufficiently large n,

$$p(X_n = x|X_0 = x) > 0. \tag{2.11}$$

A Markov chain is aperiodic if all of its states are. Finally, a state x is positive recurrent if the mean return time back to x is finite and a Markov chain is positive recurrent if all of its states are.

All Markov chains we consider in the following sections can be shown to be ergodic under weak conditions [Neal, 1993].

In the context of MCMC algorithms, we aim to construct an ergodic Markov chain with a specific stationary distribution. However, directly constructing a transition kernel T such that (2.9) holds is difficult. Luckily, there is a simple sufficient condition for (2.9) to hold: *detailed balance*. A Markov chain is said to be in detailed balance if

$$\forall x, x' \quad \pi(x)T(x \to x') = \pi(x')T(x' \to x). \tag{2.12}$$

Markov chains in detailed balance are also known as reversible.[2] Detailed balance for π implies that π is a stationary distribution of the Markov chain since:

$$\begin{aligned} \forall x, x' \quad \int_{\mathcal{X}} \pi(x)T(x \to x')dx &= \int_{\mathcal{X}} \pi(x')T(x' \to x)dx \\ &= \pi(x') \int_{\mathcal{X}} T(x' \to x)dx \\ &= \pi(x') \end{aligned} \tag{2.13}$$

Detailed balance allows us to easily construct Markov chains with a desired target distribution.

2.1.5 Metropolis-Hastings

The most generic class of MCMC algorithms is Metropolis-Hasting (MH) [Hastings, 1970]. Given a target distribution π, in every step, a proposal is drawn from a proposal distribution $q(x'|x)$. To satisfy detailed balance and thus guarantee the correct stationary distribution, Metropolis-Hastings uses an accept-reject step that accepts the proposed transition $x \to x'$ with probability

$$\mathbb{P}(\text{accept}) = \min\left(1, \frac{\pi(x')q(x|x')}{\pi(x)q(x'|x)}\right). \tag{2.14}$$

It is easy to see that the transition kernel given by q and the accept-reject step

[2] Recently some irreversible MCMC samplers have been proposed (e.g. Bouchard-Côté et al. [2018]). These samplers constitute an exciting research direction.

satisfies detailed balance:

$$\pi(x)T(x \to x') = \pi(x)q(x'|x)\min\left(1, \frac{\pi(x')q(x|x')}{\pi(x)q(x'|x)}\right) \tag{2.15}$$

$$= \min\left(\pi(x)q(x'|x), \pi(x')q(x|x')\right) \tag{2.16}$$

$$= \pi(x')q(x|x')\min\left(1, \frac{\pi(x)q(x'|x)}{\pi(x')q(x|x')}\right) \tag{2.17}$$

$$= \pi(x')T(x' \to x) \tag{2.18}$$

The generic algorithm is summarised in algorithm 1. Note that we can also use unnor-

Algorithm 1 Metropolis-Hastings.

(Possibly unnormalized) target distribution : $\pi(x)$
Proposal distribution: $q(x'|x)$
for t=1, ... **do**
  Sample proposed transition $x' \sim q(x'|x_t)$
  accept $\sim$ Bernoulli $\left(\min\left(1, \frac{\pi(x')q(x|x')}{\pi(x)q(x'|x)}\right)\right)$
  if accept **then**
    $x_{t+1} = x'$
  else
    $x_{t+1} = x_t$
  end if
end for

malised targets within Metropolis-Hastings because the target distribution appears in both the denominator and the numerator.

While Metropolis-Hastings offers a general recipe to construct Markov chains with the desired target distribution, it does not provide a recipe to construct good proposal distribution. In a low-dimensional setting, we can often use an isotropic Gaussian distribution centred on the current parameter value. Metropolis-Hastings with this choice of proposal distribution is known as Random Walk Metropolis-Hastings.

2.1.6 Gibbs Sampling

Another common MCMC algorithm is Gibbs sampling. Gibbs sampling can be seen as a special case of Metropolis-Hastings and can be used whenever we can sample from conditional distributions of our target. Let us assume we wish to sample from a distribution $\pi(x)$ where $x = \left(x^{(1)}, \ldots, x^{(n)}\right)$ and we can sample from the conditional distributions $\pi(x^{(i)}|x^{(-i)})$, where $x^{(-i)}$ is short-hand for $x^{(1)}, \ldots, x^{(i-1)}, x^{(i+1)}, \ldots, x^{(n)}$.

Gibbs sampling then cycles through the coordinates of x, sampling from the conditional distributions. See algorithm 2. It is easy to see that Gibbs sampling satisfies detailed balance.

Often the conditional distributions can be sampled from directly, whereas the the full distribution is intractable, rendering Gibbs sampling an effective algorithm. However, Gibbs sampling can often suffer from slow convergence to the stationary distribution (also known as mixing), especially when dimensions of x are highly correlated. In addition, its sequential nature scales badly to high dimensions.

Algorithm 2 Gibbs sampling.

Conditional target distributions: $\pi\left(x^{(k)}|x^{(-k)}\right)$
for t=1, ... **do**
  $x_t = x_{t-1}$
  for k=1, ... n **do** ▷ Iterate over conditionals
    Sample $x_t^{(k)} \sim p\left(x^{(k)}|x_t^{(-k)}\right)$
  end for
end for

2.1.7 Practicalities of MCMC

MCMC methods offer asymptotically exact samples from the target distribution. However, in practice, we do not have infinite computational resources and MCMC methods are not without pitfalls. One problem is the fact that samples are correlated, leading to potentially biased estimators. While the convergence results above ensure asymptotically unbiased estimators, it is very difficult to assess convergence in practice. In practice, practitioners often ignore a number of initial samples to remove the effect of the starting point. This is known as burn-in. In addition, many Markov chains targeting to the same distribution are typically run in parallel.

Multimodal target distributions are particularly problematic since MCMC samplers typically propose local transitions. While detailed balance ensures that all modes are visited eventually, it can often be very unlikely to transition from one mode to another and there is no way of knowing whether the sampler missed a mode or spend the correct fraction of time in each mode.

Often the particular parametrisation of the model can have a big impact on convergence. Perhaps surprisingly, it is often possibly to improve mixing of Markov chains by introducing *auxiliary variables* in such a way that the resulting larger space

is easier to sample from. One example of such an approach is Hamiltonian Monte Carlo, which takes inspiration from classical mechanics.

2.1.8 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo [Duane et al., 1987b, Neal, 2011, Betancourt, 2017] augments the sample space with momentum variables and uses measure-preserving, deterministic Hamiltonian dynamics to propose good, long-range transitions within MH. It is arguably the state of the MCMC for high-dimensional problems with continuous variables. There are several high-performance implementations available to scale HMC to very large problems [Carpenter et al., 2017, Tran et al., 2018].

To sample from a d-dimensional target distribution π, HMC introduces momentum variables $p \in \mathbb{R}^d$. Transitions are proposed by simulating Hamiltonian dynamics (see e.g. Arnol'd [1989]) based on the following Hamiltonian:

$$H(x, p) = -\log \pi(x) + \frac{1}{2} p^T M^{-1} p, \tag{2.19}$$

where the negative log target distribution forms a potential energy and the additional term $\frac{1}{2} p^T M^{-1} p$ is interpreted as a kinetic energy. M is a mass matrix that can be used to adapt the algorithm to the geometry of the problem [Betancourt, 2017, Barp et al., 2018]. HMC can be intuitively understood as a particular moving with a potential energy of $-\log p(x)$ and a certain kinetic energy. As the particle moves to more unlikely regions of the space, the kinetic energy decreases. However, provided that there is sufficient kinetic energy at the beginning of the trajectory, HMC can cross regions with little probability mass and transition to different modes. HMC's stationary distribution is, up to a normalisation constant, given by $\propto \exp\left(-H(x, p)\right)$.

Simulating Hamiltonian dynamics analytically is typically not possible and a numerical integrator has to be used. In principle, a numerical integrator could make the MH acceptance probability intractable. Fortunately, with a specific type of volume-preserving integrator called leapfrog integrator, the acceptance probability takes a simple form. HMC has at least two hyperparameters: the stepsize for the leapfrog integrator and the number of leapfrog steps L. The full algorithm is shown in algorithm 3.

In relatively high-dimensional settings probability mass tends to concentrate on thin shells or typical sets [3]. These typical sets are not necessarily where the density is highest but dominate expectations due to their volume. For example, consider

[3] This fact has been well known for a long time. For an excellent discussion see e.g. Betancourt [2017]

random variables X_d following a d-dimensional standard Gaussian distribution. In one dimension, most of the probability mass is concentrated around the mode. However as d increases the distribution becomes more and more concentrated around the unit sphere, where $\sum_{i=1}^{d} X_i^2 = d$, by the central limit theorem. Thus an efficient integration methods should focus most of its resources on this region. Following the gradient of the target distribution directly would propose moves towards the mode, leaving the typical set. The momentum within HMC allows it to propose transitions aligned with the typical set, where as MH algorithms with more local proposals typically do not. This partly explains the effectiveness of HMC relative to other MCMC samplers in relatively high-dimensional settings.

A more in-depth discussion of HMC is beyond the scope of this thesis. For an excellent introduction, see Betancourt [2017]. In chapter 3, we will discuss an extension of HMC that uses a different kinetic energy in the Hamiltonian.

Algorithm 3 Hamiltonian Monte Carlo.

(Possibly unnormalized) target distribution π
Mass matrix M
Leapfrog stepsize ϵ
Number of leapfrog steps L
for t=1, ... **do**
  Sample momentum $p \sim \mathcal{N}(0, M)$
  Simulate Hamiltonian dynamics for L leapfrog steps to obtain proposal x', p'
  accept $\sim$ Bernoulli $(\min(1, \exp(-(H(x', p') - H(x_t, p_t)))))$
  if accept **then**
    $x_{t+1} = x'$
    $p_{t+1} = p'$
  else
    $x_{t+1} = x_t$
    $p_{t+1} = p_t$
  end if
end for

2.2 Stochastic Gradient Markov Chain Monte Carlo

Traditional MCMC methods as described in the preceding sections often scale poorly to large datasets, since every iteration typically requires a full pass through the dataset to calculate the full gradient or the full log likelihood. Inspired by stochastic gradient descent, Welling and Teh [2011] proposed stochastic gradient Langevin dynamics (SGLD) as a novel MCMC algorithm with lower per-iteration cost. SGLD uses a

discretisation of a stochastic differential equation whose invariant distribution is the target distribution. It uses minibatches of data to estimate the gradients in every step. Since the introduction of SGLD, there has been a lot of interest in stochastic gradient MCMC algorithms. In this section, we review stochastic gradient MCMC (SGMCMC) algorithms.

Since stochastic gradient MCMC methods are primarily used for Bayesian, we will describe them in a Bayesian setting. For simplicity, let us assume a dataset $X = \{x_i\}_i^n$ of i.i.d. observations. Let $\theta \in \mathbb{R}^d$ be a parameter of interest and let $p(\theta)$ be its prior distribution. Let $p(x|\theta)$ denote the likelihood. The posterior distribution of θ is then given by:

$$p(\theta|X) \propto p(\theta) \prod_{i=1}^{N} p(x_i|\theta). \tag{2.20}$$

SGLD is based on the over-damped Langevin diffusion:

$$d\theta_t = \left(\nabla \log p(\theta_t) + \sum_{i=1}^{N} \nabla \log p(x_i|\theta_t) \right) dt + \sqrt{2} dW_t, \quad \theta_0 \in \mathbb{R}^d \tag{2.21}$$

where W_t is a d-dimensional Brownian motion. It is well known that Langevin dynamics are ergodic with respect to a suitably smooth target distribution $p(\theta|x)$ [Roberts et al., 1996]. While the direct simulation of Langevin dynamics would yield a sample from the posterior, exact solutions to (2.21) are typically unavailable. However, it is possible to discretise the dynamics to obtain an approximate solution. An application of the Euler scheme yields

$$\theta_{t+1} = \theta_t + h_t \left(\nabla \log p(\theta_t) + \sum_{i=1}^{N} \nabla \log p(x_i|\theta_t) \right) + \sqrt{2h_t} \xi_t \tag{2.22}$$

where ξ_t is a standard Gaussian random variable on $\mathbb{R}^d$. SGLD is related to stochastic gradient descent (SGD, Robbins and Monro [1951]) but differs from SGD because it injects a small amount of noise in each step. The injection of noise prevents SGLD converging to a mode of the target distribution as SGD would. The stochastic gradient Langevin dynamics algorithm (SGLD) reduces the per-iteration cost of the full Euler scheme by subsampling the sum of the N log likelihood gradient terms based on a minibatch of data with batchsize $n \ll N$. This yields the SGLD update:

$$\theta_{t+1} = \theta_t + h_t \left(\nabla \log p(\theta_t) + \frac{N}{n} \sum_{i=1}^{n} \nabla \log p\left(x_{\tau_i^t}|\theta_t\right) \right) + \sqrt{2h_t} \xi_t \tag{2.23}$$

where τ_s^t is a random subset of $\{1, \cdots, N\}$. While this update technically only yields a valid posterior sample for the usual conditions on the stepsizes

$$\sum_t h_t = \infty, \qquad \sum_t h_t^2 < \infty, \tag{2.24}$$

SGLD and related algorithms are typically used with a constant stepsize in practice, incurring a bias.

Since the introduction of SGLD, a large number of related SGMCMC algorithms have been proposed, and we discuss some of the most important ones here. One of the most common SGMCMC algorithms is stochastic gradient Hamiltonian Monte Carlo (SGHMC, Chen et al. [2014]). Similar to Hamiltonian Monte Carlo, SGHMC augments the parameter space with momentum variables $p \in \mathbb{R}^d$. SGHMC uses the following update:

$$\begin{aligned} p_{t+1} &= p_t + h_t \left(\nabla \log p(\theta_t) + \frac{N}{n} \sum_{i=1}^{n} \nabla \log p\left(x_{\tau_i^t} | \theta_t\right) \right) - h_t D p_t + \sqrt{2 h_t D} \xi_t \\ \theta_{t+1} &= \theta_t + h_t p_{t+1}. \end{aligned} \tag{2.25}$$

Note how the magnitude of the friction term $h_t D p_t$ is related to the magnitude of the injected noise term $\sqrt{2h_t D}\xi_t$. If full gradients are used and the system is damped just enough to have to correct invariant distribution. With stochastic gradients, this is not the case. Assuming that the stochastic gradient noise is approximately Gaussian, we can try to estimate the gradient noise and subtract its variance from the variance of the injected noise [Chen et al., 2014]. An alternative, adaptive approach is stochastic gradient Nosé-Hoover thermostats (SGNHT, Ding et al. [2014], Leimkuhler and Shang [2016]). SGNHT augments the system with an additional thermostat variable τ that adaptively changes the friction coefficient of SGHMC. The update equations are given by:

$$\begin{aligned} \tau_{t+1} &= \tau_t + \mu^{-1}(p^T p - d) \\ p_{t+1} &= p_t + h_t \left(\nabla \log p(\theta_t) + \frac{N}{n} \sum_{i=1}^{n} \nabla \log p\left(x_{\tau_i^t} | \theta_t\right) \right) - h_t \tau p_t + \sqrt{2 h_t D} \xi_t \\ \theta_{t+1} &= \theta_t + h_t p_{t+1}, \end{aligned} \tag{2.26}$$

where d is the dimensionality θ.

SGNHT takes inspiration from statistical physics and intuitively works in the following way: by construction, we know that the system without stochastic gradients should have $\mathbb{E}[p^T p] = d$, that is that the average kinetic energy of the system is d.

In the presence of stochastic gradients the kinetic energy of the system is generally larger. SGNHT uses d as the target value for the kinetic energy to adaptively 'cool' or 'heat' the system. How reactive the algorithm is to deviations of the kinetic energy from the target value is governed by a hyperparameter μ. SGNHT, when well-tuned, is remarkably effective and allows the use of larger stepsizes within SGMCMC algorithms. Since the introduction of SGNHT, other types of thermostat SGMCMC algorithms have been proposed such as multivariate stochastic gradient Nosé-Hoover thermostats [Li et al., 2016b], Nosé-Poincaré thermostats [Roychowdhury et al., 2016], and thermostats for variants of SGHMC with different kinetic energies (Lu et al. [2017], see chapter 3). Deriving valid SGMCMC algorithms is not trivial but Ma et al. [2015] provide framework for deriving algorithms, allowing them to derive stochastic gradient Riemannian HMC, which takes into account the geometry of the target distribution. We review and make use of their framework in chapter 3.

Stochastic gradient MCMC methods have been applied across a large range of machine learning applications, such as matrix factorisation models for recommender systems (Chen et al. [2014], Ding et al. [2014], Ahn et al. [2015]), topic models (Ding et al. [2014], Gan et al. [2015]), and neural networks (Li et al. [2016a], Chen et al. [2014]) with good results.

However, due to the complexity and high dimensionality of these models, it is hard to assess whether these promising results are due to accurate posterior sampling or simply due to averaging over different parameter values (akin to ensemble methods). Indeed, even averaging over samples from stochastic gradient descent results in a crude posterior approximation with often similar performance (Mandt et al. [2016]).

In addition, as we point out in Nagapetyan et al. [2017], a simple back-of-the-envelope calculation raises questions as to whether SGLD and other stochastic gradient MCMC schemes without mechanisms for variance reduction reduce the overall computational cost if we desire high accuracy estimates of the posterior. For SGLD to yield accurate posterior samples, the injected noise should dominate or at least be comparable to the additional variance due to the stochastic gradients.

$$var(\sqrt{2h}\xi) \approx var\left(h\left(\nabla \log p(\theta) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{\tau_i}|\theta)\right)\right).$$

The injected noise has variance of h whereas the variance in the stochastic gradients is model dependent, but scales as $N^2 \cdot h^2 \cdot \frac{N-n}{N \cdot n}$. Hence, up to a problem-dependent

constant

$$\begin{aligned} h &\propto N^2 \cdot h^2 \cdot \frac{N-n}{N \cdot n} \\ \Rightarrow n(1+hN) &\propto N^2 h \\ \Rightarrow n &\propto N^2 h/2. \end{aligned} \tag{2.27}$$

In particular, this calculation suggests that we need $n/h \approx \textit{constant}$ to satisfy (2.27). This means that the total computational cost of simulating the dynamics is independent of the batchsize, assuming that the cost per step is proportional to n and the required total number of steps is proportional to $1/h$ (implying a constant simulation time of the underlying SDE). Of course, this analysis ignores other factors such as discretization errors.

Our work in Nagapetyan et al. [2017] studies this question more rigorously for strictly log-concave models with Lipschitz gradients. We show that the computational cost to reach high posterior accuracy is indeed roughly independent of that batchsize in such models, because small stepsizes have to be used to control the variance and conduct a numerical study demonstrating this phenomenon.

Two ways to tackle the variance in the stochastic gradients have been proposed: the first is variance reduction schemes based on control variates [Dubey et al., 2016, Baker et al., 2018, Nagapetyan et al., 2017]; the second approach is augmenting the dynamics with thermostat variables based on ideas from molecular dynamics (see Ding et al. [2014], Gan et al. [2015], Leimkuhler and Shang [2016]) described above.

Chapter 3 introduces Relativistic Hamiltonian Monte Carlo [Lu et al., 2017], where one of my contributions was deriving a thermostat version of the algorithm.

2.3 Variational Methods

MCMC methods are asymptotically exact, but, as we have discussed, in practice it is often difficult to assess convergence or to estimate the resulting bias. In addition, standard MCMC methods typically perform a full pass through the dataset per iteration making them computationally expensive. Another problem is that MCMC methods do not immediately provide an estimate of the marginal likelihood that could be used for model comparison and model selection.

An alternative approach is to approximate the target distribution directly with a tractable parametric distribution. We will refer to such methods as variational methods. In section 2.3.1 we first review the Laplace approximation, a simple method that relies on a local expansion of the target distribution around a mode. We review the

class of variational inference methods that find a variational approximation q by minimising $\mathsf{KL}(q\|p)$ in section 2.3.2. We discuss variational autoencoders (VAEs, Kingma and Welling [2013]) in section 2.3.3. Chapters 5, 7 and 8 build on variational inference and techniques related to VAEs. Finally, in section 2.3.4 we introduce assumed density filtering and expectation propagation which forms the basis for chapter 4.

2.3.1 Maximum-A-Posteriori Inference and the Laplace Approximation

A classic method in statistics and machine learning is maximum likelihood learning: model parameters θ are found through optimisation of the log-likelihood of the data:

$$\theta^\star = \arg\max_\theta \log p(\mathcal{D}|\theta). \tag{2.28}$$

A variation of maximum likelihood learning in a Bayesian setting is maximum-a-posteriori learning (MAP), which takes into account the prior:

$$\theta^\star = \arg\max_\theta \left\{\log p(\mathcal{D}|\theta) + \log p(\theta)\right\}. \tag{2.29}$$

MAP inference approximates the posterior with a delta distribution centred at the posterior mode. MAP slightly regularises the maximum likelihood estimate towards the prior, but still does not take into account the uncertainty in the posterior. The Laplace approximation [Laplace, 1809] expands the target density to second order around the posterior mode $\theta^\star$ to construct a Gaussian approximation:

$$\log p(\theta^{MAP} + \Delta\theta|\mathcal{D}) = \log p(\theta^{MAP}|\mathcal{D}) + \frac{1}{2}\Delta\theta^T \underbrace{\frac{\partial^2 \log p(\theta|\mathcal{D})}{\partial\theta\partial\theta^T}\Big|_{\theta=\theta^\star}}_{:=H} \Delta\theta + \mathcal{O}(\Delta\theta^3). \tag{2.30}$$

The resulting approximation is:

$$\theta \sim \mathcal{N}\left(\theta^\star, H^{-1}\right) \tag{2.31}$$

Note that there is no first-order term since $\theta^\star$ is the posterior mode and we do not need to know the normalisation constant to form the Laplace approximation. This approximation will work best when the posterior is highly peaked around the mode, but is less well-suited to multimodal problems. The Bernstein-von-Mises theorem [Doob, 1949, Le Cam, 1986] guarantees that, under regularity conditions, in a large data limit the posterior will eventually become uni-modal and approximately Gaussian. Thus, the Laplace approximation is an excellent option in a large-data setting with a moderate number of parameters.

2.3.2 Variational Inference

Variational inference (VI) is a large class of methods that propose to approximate a target distribution p with a variational approximation from a family of tractable, parametric distributions $\mathcal{F}$ by minimising:

$$q^\star = \arg\min_{q\in\mathcal{F}} \mathsf{KL}(q\|p). \tag{2.32}$$

See Blei et al. [2016] for a recent review. Let us assume that we are interested in a model $p(x, z)$, where x are observed data with likelihood $p(x|z)$ and z are unobserved latent variables (which could be global parameters or and separate latent variables per datapoint x_i) with prior $p(z)$. The distribution of interest is the posterior $p(z|x)$. If we only know $p(z|x)$ up to a normalising constant, we cannot directly minimise (2.32). Luckily there is a simple and (up to a constant) equivalent objective that is tractable:

$$\begin{aligned}\mathsf{KL}(q\|p) - \log p(x) &= \mathbb{E}_q\left[\log q(z) - \log p(z|x) - \log p(x)\right] &(2.33)\\ &= \mathbb{E}_q\left[\log q(z) - \log p(x, z)\right] &(2.34)\end{aligned}$$

Note that since $\mathsf{KL}(q\|p)$ is non-negative we can interpret (2.34) as a lower bound

$$\begin{aligned}\log p(x) &\geq \mathbb{E}_q\left[\log p(x, z, \theta) - \log q(z)\right]\\ &= \mathbb{E}_q\left[\log p(x|z)\right] - \mathsf{KL}(q(z)\|p(z)) &(2.35)\end{aligned}$$

known as the evidence lower bound (ELBO). The ELBO contains two terms. Firstly, a likelihood or reconstruction term $\mathbb{E}_q\left[\log p(x|z)\right]$ that captures how well the approximate posterior q explains the data and a KL term $\mathsf{KL}(q(z)\|p(z))$ that penalises deviations from the prior.

Of course, the quality of the approximation depends on the choice of variational family $\mathcal{F}$. One common choice is mean-field variational inference, which posits a factorised approximation:

$$q(z) = \prod_{i=1}^{m} q_i(z_i). \tag{2.36}$$

While computationally efficient, mean-field variational inference severely restrict the variational family and does not allow any dependencies between latent variables in the variational posterior. Typically, exponential family distributions are used for the individual factors in (2.36).

Some authors propose more structured variational approximations (e.g. Saul and Jordan [1996]) or using additional auxiliary variables (e.g. Bishop et al. [1998]). While these methods can improve the approximation quality, they often lead to harder optimisation problems.

One simple idea to derive a practical algorithm based on the mean-field approximation is to update the individual mean-field factors sequentially. This is known as Coordinate Ascent Variational Inference (CAVI, Bishop [2006]) and we briefly derive it below.

In the context of the mean-field approximation, let us define z_{-i} to be all coordinates (or more generally, factors in the factorisation) other than i. The ELBO can be written as:

$$\int q_i(z_i) q_{-i}(z_{-i}) \left[\log p(x, z) - \log q_i(z_i) - \log q_{-i}(z_{-i})\right] dz \tag{2.37}$$

To find the optimal value of $q_i(z_i)$, holding q_{-i} fixed, we can frame this as a constrained optimisation problem:

$$\max_{q_i} \int q_i(z_i) q_{-i}(z_{-i}) \left[\log p(x, z) - \log q_i(z_i) - \log q_{-i}(z_{-i})\right] dz$$
$$\text{subject to } \int q_i(z_i) dz_i = 1. \tag{2.38}$$

Introducing a Lagrange multiplier λ for the equality constraint, differentiating and setting to zero, we find:

$$\int q_{-i}(z_{-i}) \log p(x, z) - \log q_i(z_i) - 1 + \lambda = 0$$
$$q_i(z_i) \propto \exp\left(\mathbb{E}_{q_{-i}} \log p(x, z)\right) \tag{2.39}$$

This results gives us an update rule for the i-th factor based on all others. For many models, the CAVI updates take simple, tractable forms. See algorithm 4.

Algorithm 4 Coordinate Ascent Variational Inference.

Model $p_\theta(x, z)$
while ELBO not converged **do** ▷ Iterations
  for $i = 1 \dots m$ **do** ▷ Iterate over factors
    Update $q_i \propto \exp\left(\mathbb{E}_{q_{-i}} \left[\log p(x, z)\right]\right)$
    Recalculate ELBO and check convergence
  end for
end while

CAVI is conceptually simple, but does not scale well to large datasets, since it requires a full pass through the dataset with each update. Note the similarities to Gibbs sampling.

Stochastic variational inference (SVI) [Hoffman et al., 2013b] instead uses mean-field approximations in conjugate models with natural gradients [Amari, 1998] and stochastic approximation methods [Robbins and Monro, 1951] to scale VI to very large models, such as topic models on millions of documents.

Using CAVI or SVI restricts us to certain classes of models for which we have to derive updates separately. To make variational inference easier to use for practitioners and allow experimentation with such techniques, there has been substantial interest in black-box variational inference algorithms (e.g. Ranganath et al. [2014], Titsias and Lázaro-Gredilla [2014], Tran et al. [2015], Ranganath et al. [2016], Tran et al. [2018]. Such algorithms only require the user to specify the model. No model-specific derivations are necessary. Of course, for specific models, it may often be possible to exploit model structure through more tailored algorithms. Work on variational autoencoders, described below, can also be viewed as black-box variational inference [Kingma and Welling, 2013, Rezende et al., 2014].

Recall the definition of the ELBO in (2.35). The fact that the ELBO objective forms a lower bound on the log marginal probability $\log p(x)$ allows us to perform maximum-likelihood or MAP inference with respect to the model parameters in models with latent variables via the variational expectation maximisation algorithm (EM) [Dempster et al., 1977, Neal and Hinton, 1999].

The (variational) EM algorithm (shown in algorithm 5) performs alternating optimisation with respect to the variational approximation $q(z)$ and the model parameters θ. If the exact posterior $p_\theta(z|x)$ is tractable (e.g. in Gaussian mixture models) the EM algorithm improves the log likelihood in every step and hence will converge to a local maximum of $\log p_\theta(x)$. However, even if this is not the case, we can apply the variational EM algorithm, alternating between updating the variational approximation q and the model parameters. Here, the quality of our variational approximation governs the bias of the learned model parameters relative to the true maximum-likelihood or MAP solution. Generally speaking, the better our variational approximation the smaller the bias will be.

2.3.3 Variational Autoencoders

A particularly successful, recent application of these ideas are variational auto-encoders (VAEs) [Kingma and Welling, 2013, Rezende et al., 2014]. The graphical model for

Algorithm 5 Variational Expectation Maximisation.

Model $p_\theta(x, z)$
for t=1, ... **do**
  E-Step:
  update variational approximation $q^{(t)}(z) = \arg\max_{q \in \mathcal{F}} ELBO(q, \theta^{(t-1)})$
  M-Step:
  update model parameters $\theta^{(t)} = \arg\max_\theta ELBO(q^{(t)}, \theta)$
end for

VAEs is shown in figure 2.1. Variational auto-encoders consist of a model $p_\theta(x|z)$, known as the *decoder* that maps latent variables z to data x. Associated with the latent variables is a prior $p(z)$. To enable learning, the decoder parameters θ via the ELBO uses an *encoder* distribution $q_\phi(z|x)$ that forms the variational approximation. Typically, the decoder and the encoder are both given by neural networks. VAEs make use of two ideas to make training efficient: *amortised inference* and the *reparametrisation trick*.

In a classic mean field approximation with per-datum latent variables, each latent variables z_i has its own variational posterior $q(z_i)$, which is both memory intensive and scales badly to large datasets. In addition, we would expect the variational posterior of similar data points to be similar, but this intuition about posterior smoothness is hard to incorporate in mean-field VI.

Amortised inference [Gershman and Goodman, 2014] instead uses variational distributions whose parameters are given by a neural network, forming the encoder $q_\phi(z|x)$. This shared neural network amortises the cost of learning the variational posterior and also makes it easy to evaluate the model at a new test data point.

The second idea to make training variational autoencoders efficient is the reparametrisation trick. Consider the gradient of the ELBO with respect to the encoder parameters ϕ. One classic way to approximate such gradients is the score-function trick. Since $\nabla \log q_\phi(z|x) = \frac{\nabla_\phi q_\phi(z|x)}{q_\phi(z|x)}$, we can rewrite the gradient of expectations

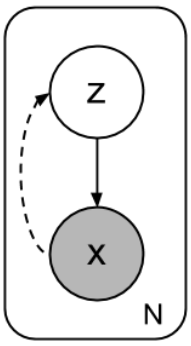


Figure 2.1: Graphical model for variational autoencoders. Here, the solid lines denotes the generative model $p_\theta(x|z)$. The dashed line corresponds to the inference network $q_\phi(z|x)$.

$\mathbb{E}_{q_\phi(z|x)}[f(z)]$ with respect to $q_\phi(z|x)$ as follows:

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi(z|x)}[f(z)] &= \nabla_\phi \int_{\mathcal{Z}} q_\phi(z|x) f(z) dz \\ &= \int_{\mathcal{Z}} \nabla_\phi q_\phi(z|x) f(z) \\ &= \int_{\mathcal{Z}} q_\phi(z|x) \nabla_\phi \log q_\phi(z|x) f(z) \\ &= \mathbb{E}_{q_\phi(z|x)} \left[\nabla_\phi \log q_\phi(z|x) f(z)\right] \end{aligned} \tag{2.40}$$

While this allows us to form unbiased gradient estimates, in practice this estimator suffers from high variance which hinders training. This gradient estimator is also known as the REINFORCE gradient estimator in reinforcement learning after a classic algorithm that uses it to calculate policy gradients [Williams, 1992]. There are a number of papers proposing improved versions of this gradient estimator for dealing with discrete latent variables (e.g. Mnih and Gregor [2014], Mnih and Rezende [2016], Tucker et al. [2017]).

The reparametrisation trick instead leverages reparametrisable distributions, which can be written as deterministic functions of the distribution parameters and some independent noise. For example, for the normal distribution if $x \sim \mathcal{N}(\mu, \sigma^2)$, we can reparametrise the distribution as $x = \mu + \sigma\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$. More generally, assume that we can reparametrise $z = g(\phi, \epsilon)$, where the distribution of ϵ does not depend on the encoder parameters ϕ. This allows us to rewrite the gradient of the ELBO as:

$$\nabla_\phi ELBO = \nabla_\phi \mathbb{E}_{q_\phi(z|x)} \left[\log p(x, z) - \log q_\phi(z|x)\right] \tag{2.41}$$

$$= \nabla_\phi \mathbb{E}_\epsilon \left[\log p(x, g(\phi, \epsilon)) - \log q_\phi(g(\phi, \epsilon)|x)\right] \tag{2.42}$$

$$= \mathbb{E}_\epsilon \left[\nabla_\phi \log p(x, g(\phi, \epsilon)) - \nabla_\phi \log q_\phi(g(\phi, \epsilon)|x)\right], \tag{2.43}$$

which we can easily approximate by sampling from ϵ.

One issue for VAEs, and variational inference more generally, is the flexibility of the variational approximation, since the posterior distribution can be very complicated and a loose variational bound can lead to biased maximum likelihood estimates for the model parameters. Generally, the tighter the bound, the better the model. Many different solutions have been proposed to make variational approximation more flexible, such as auxiliary variables [Salimans et al., 2014, Sønderby et al., 2016] and normalising flows [Rezende and Mohamed, 2015, Kingma et al., 2016]. Chapter 5

describes a new normalising flow that allows for more flexible variational approximations.

Since their invention in 2013, VAEs have become a ubiquitous tool in machine learning. They have been used successfully for generative modelling of image data (e.g. Kingma et al. [2016], Gulrajani et al. [2016], Kingma and Dhariwal [2018]), natural language modelling (e.g. Bowman et al. [2015]), model-based reinforcement learning (e.g. [Ha and Schmidhuber, 2018]), and many other applications. Methods based on VAEs are among the state of the art for semi-supervised learning [Maaløe et al., 2016]. Crucial for many applications is the ability of VAEs to use the learned latent space as a useful representation of the data for down-stream tasks. This form of representation learning also underlies some of the results in chapters 7 and 8, which use ideas from variational inference and VAEs.

2.3.4 Expectation Propagation algorithms

Another class of variational methods are expectation propagation (EP) algorithms [Minka, 2001b, Opper and Winther, 2005]. They can be seen as generalisation of belief propagation and the sum-product algorithm [Pearl, 1988]. One of the most important properties of a multivariate probability distribution is its factorisation structure. Often, leveraging such structure can dramatically reduce the computation cost of common statistical operations such as the computing marginal distributions. A particularly powerful example is the forward-backward algorithm for Hidden Markov Models [Baum, 1972]. EP algorithms make explicit use of the factorisation of the target distribution, forming an exponential family approximation for each factor. These factors are then refined one-by-one in an iterative fashion, taking into account the approximations to all other factors. The assumption underlying EP is that individual factors are easier to approximate than the full distribution.

Exponential family distributions are at the heart of EP algorithm and we review them in section 2.3.4.1. Then, we introduce expectation propagation algorithms in section 2.3.4.2, laying the ground work for chapter 4. In section 2.3.5, we end with a discussion of the advantages and disadvantages of variational inference and EP.

2.3.4.1 Exponential Family Distributions

In this section, we briefly review exponential family distributions (see Brown [1986] for a more in-depth treatment). The notation closely follows Wainwright and Jordan [2008].

A powerful (if controversial) principle in probabilistic modelling is the principle of maximum entropy [Jaynes, 1963]. It states that the distribution that best represents the current state of knowledge is the distribution that maximises entropy given known constraints. Maximising entropy is sometimes described as maximising randomness. Let us assume that we know the value of some statistic $s(x)$ of a data distribution p. The maximum entropy distribution can then be characterised as the solution to the following problem:

$$\begin{aligned} &\max_p \mathbb{E}_p[-\log p(x)] \\ &\text{subject to } \mathbb{E}_p\left[s(X)\right] = \mu. \end{aligned} \tag{2.44}$$

It is easy to show, by introducing a Lagrange multiplier for the constraint and differentiating, that exponential family distributions arise as solutions to this problem. They have rich mathematical structure.

An exponential family with associated d-dimensional sufficient statistic $s : \mathcal{X} \to \mathbb{R}^d$ is the collection of probability distributions indexed by $\theta \in \mathbb{R}^d$ with densities of the form:

$$p_\theta(x) = \exp\left(\theta^T s(x) - A(\theta)\right) \tag{2.45}$$

θ is known as the natural parameter and $A(\theta)$ is known as the log partition function and normalizes the distribution

$$A(\theta) = \log \int \exp\left(\theta^T s(x)\right) dx. \tag{2.46}$$

p_θ is a valid distribution, whenever $A(\theta)$ is bounded (and hence, the density is normalisable). This motivates the definition of the natural parameter space:

$$\Theta := \left\{\theta \in \mathbb{R}^d : A(\theta) < \infty\right\} \tag{2.47}$$

It can be shown that the log partition function is convex [Wainwright and Jordan, 2008] and hence that the natural parameter space is a convex set. An exponential family is *regular* if Θ is an open set and *minimal* if the components of s are linearly independent.

Another important quantity associated with exponential families is the expected value of the sufficient statistic, known as the mean parameter:

$$\mu = \mathbb{E}_{p_\theta}[s(X)] \tag{2.48}$$

Associated with the mean parameter is the mean parameter space:

$$\mathcal{M} = \{\mu : \exists \text{ distribution } p \text{ with } \mu = \mathbb{E}_{p_\theta}[s(X)]\}\,, \tag{2.49}$$

which can be shown to be a closed convex set.

Assuming minimality, it can be shown that the function $\theta \to \nabla A(\theta)$ is bijective onto the interior of the mean parameter space $\mathcal{M}$ and maps a natural parameter to the associated mean parameter: $\mu(\theta) = \nabla A(\theta)$. Furthermore, due to the convexity of A we can define its convex conjugate:

$$A^\star(\mu) = \sup_{\theta\in\Theta} \left\{\theta^T\mu - A(\theta)\right\}. \tag{2.50}$$

Finally, $\nabla A^\star$ maps the mean parameter to the associated natural parameter: $\theta = \nabla A^\star(\mu)$. See figure 2.2.

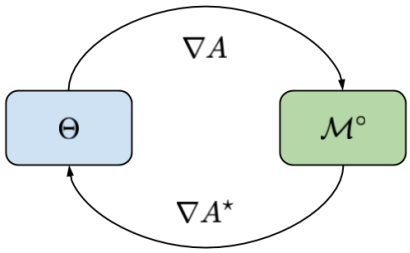


Figure 2.2: Bijection between the natural parameter space Θ and the interior of $\mathcal{M}$.

A crucial operation for EP algorithms is to find a close approximation to a distribution p within an exponential family. In classic EP this is done via minimisation of a Kullback-Leibler divergence and known as a projection. To project a distribution[4] to an exponential family with sufficient statistics s we solve the following optimisation problem:

$$\text{proj}(p) = \arg\min_{\theta\in\Theta} \mathsf{KL}(p\|q_\theta) \tag{2.51}$$

Note that

$$\begin{aligned}
\mathsf{KL}(p\|q_\theta) &= \mathbb{E}_p\left[\log p(X) - \log q_\theta(X)\right] && (2.52)\\
&= \mathbb{E}_p\left[\log p(X) - \left(\theta^T s(X) - A(\theta)\right)\right] && (2.53)\\
&= \mathbb{E}_p\left[\log p(X)\right] - \left(\theta^T\mathbb{E}_p[s(X)] - A(\theta)\right) && (2.54)
\end{aligned}$$

Thus, minimising (2.51) is equivalent to maximising

$$\max_{\theta\in\Theta} \left\{\theta^T\mathbb{E}_p[s(X)] - A(\theta)\right\} \tag{2.55}$$

[4]Note that is possible without knowledge of the normalisation constant.

The first order condition for this maximisation problem is:

$$\mathbb{E}_p[s(X)] = \nabla A(\theta) \tag{2.56}$$

$$\theta = \nabla A^\star(\mathbb{E}_p[s(X)]) \tag{2.57}$$

Thus, projecting to an exponential family is equivalent to choosing the member of the exponential family that matches the expectation of the sufficient statistic under p. This is also known as moment matching.

The slick notation hides two different problems here. Firstly, calculating the expected value of s is generally hard, especially since calculating such expectations is the very reason we consider approximate inference methods in the first place. However, in the following we focus on one factor of a target distribution at a time and assume that each projection step is easy compared to the original problem. Secondly, $\nabla A^\star$, which is only defined as a convex conjugate, is intractable for general exponential families. However, for commonly used exponential families such as a Gaussian with diagonal covariance, $\nabla A^\star$ is available analytically.

2.3.4.2 Assumed Density Filtering, EP and Power EP

Before introducing EP, let us first discuss assumed density filtering. Let us assume an online Bayesian learning setting, in which i.i.d. data points x_t arrive one-by-one. Assuming the exact posterior is intractable, we are interested in efficiently approximating the posterior over parameters of interest $p(\theta|x_{1:t})$. By Bayes' theorem we have:

$$p(\theta|x_{1:(t+1)}) \propto p(x_{t+1}|\theta)p(\theta|x_{1:t}) \tag{2.58}$$

Opper [1998] proposes forming a sequence of exponential family approximations q_t and using q_t as a stand-in for $p(\theta|x_{1:t})$ when incorporating the next likelihood factor $p(x_{t+1}|\theta)$, i.e. posterior approximation q_{t+1} for the next time step is given by

$$q_{t+1}(\theta) = \text{proj}(p(x_{t+1}|\theta)q_t(\theta)). \tag{2.59}$$

q_0 is equal to the prior. More generally, this algorithm, known as assumed density filtering (ADF), can also be applied to a fixed target distribution consisting of a prior term f_0 and K other factors

$$p(\theta) = f_0(\theta)\prod_{k=1}^{K} f_k(\theta), \tag{2.60}$$

Algorithm 6 Assumed Density Filtering.

Factorized target distribution : $p(\theta) = \prod_{k=1}^{M} f_k(\theta)$
$\theta_1 = \text{proj}(f_0 f_1)$
for k=2, ..., K **do**
  $\theta_k = \text{proj}(f_k q_{k-1})$
end for
return θ_K

as shown in algorithm 6.

A potential drawback of ADF is that the algorithm is dependent on the order of the factors and the approximation error can compound significantly for large K. In addition, ADF considers every factor just once – in an online setting this is perhaps sensible, but in general one would hope to be able to do better with an iterative scheme that refines the initial approximations. This is the main idea of expectation propagation [Minka, 2001b, Opper and Winther, 2005].

The expectation propagation algorithm maintains an exponential family approximation q_k with natural parameter θ_k for each factor f_k. The individual approximations imply a global approximation

$$q_{\text{global}} = \prod_{k=1}^{K} q_k, \tag{2.61}$$

which is also in the exponential family and has natural parameter $\theta = \sum_{k=1}^{K} \theta_k$. In each step, it considers one random factor i. Let us define the *cavity distribution* q_{-i}

$$q_{-i} = \prod_{j \neq i} q_j, \tag{2.62}$$

which is the product of the approximations to all other factors. The name cavity distribution refers to the fact that we remove one factor from the global approximation. Similarly, we can define the *tilted distribution* $f_i q_{-i}$. Each step of EP projects the tilted distribution onto the exponential family to obtain a new global approximation θ^{new}. It then adjusts the natural parameter for factor i accordingly $\theta_i^{\text{new}} = \theta^{new} - \sum_{k \neq i} \theta_k$, keeping all other approximation fixed. See algorithm 7.

Note that ADF is a special case of EP: it corresponds to one pass of EP through the data with initial approximations $q_i \equiv 1$.

If EP converges, the global approximation q_θ matches the moments of all tilted distributions

$$\theta = \text{proj}\left(f_i q_{-i}\right) \ \forall i \tag{2.63}$$

$$\mathbb{E}_\theta[s(X)] = \mathbb{E}_{f_i q_{-i}}[s(X)]. \tag{2.64}$$

Algorithm 7 Expectation Propagation.

Factorised target distribution : $p(\theta) = \prod_{k=1}^{M} f_k(\theta)$
Initial approximations q_i with natural parameters θ_i for individual factors f_i
$\theta = \sum_{k=1}^{K} \theta_i$
while not converged **do**
  Pick factor i to refine according to a (random) schedule
  $\theta^{\text{new}} = \text{proj}(f_i q_{-i})$ ▷ Project tilted distribution to obtain new global approx.
  $\theta_i \leftarrow \theta_i + \theta^{\text{new}} - \theta$
  $\theta \leftarrow \theta^{\text{new}}$
end while
return $\theta, \{\theta_k\}_{k=1}^{K}$

This local moment matching can be seen as a relaxation of the global moment matching that would minimise $\mathsf{KL}(p\|q)$ and one view of EP is that it is a fixed-point algorithm that iteratively tries to enforce these moment constraints. While EP normally converges in practice and has been applied very successfully in a variety of different applications, such as the Microsoft's TrueSkill player matching system [Herbrich et al., 2007], it is not guaranteed to converge. While it is possible to derive convergent EP algorithms [Heskes and Zoeter, 2002], these algorithms require a double loop optimisation.

A related class of algorithms is Power EP Minka [2004], which introduces an exponent into the definition of the tilted distribution: $f_i^{\alpha} q / q_i^{\alpha}$. It can be shown to minimise a different divergence function known as an α-divergence between the global approximation and the titled distribution in each step. In chapter 4 we discuss how to formulate EP and power EP in terms of a single variational objective.

2.3.5 Advantages and Disadvantages of Variational Inference and Expectation Propagation

Variational inference and Expectation Propagation are two broad classes of approximate inference algorithm, each with its own advantages and disadvantages. They differ in a number of ways:

Local vs Global Expectation Propagation iteratively refines approximations to individual local factors whereas variational inference forms a global approximation.

Convergence, Speed and Accuracy EP is not guaranteed to converge. VI, on the other hand, is guaranteed to converge to a local optimum. VI often requires

many updates to converge whereas EP often converges within a small number of passes through the dataset. However, the moment matching step within EP can be prohibitively expensive, particularly when MCMC is required to estimate the moments of the tilted distribution. Whether VI or EP lead to more accurate solutions largely depends on the specific problem.

Mode-seeking vs Mass-covering behaviour Variational inference minimises the reverse $\mathsf{KL}(q\|p)$, which only gets penalised for low values of p where q is large. Hence, VI often leads to variational distributions covering a single mode, which partially explains why variational inference is often overly confident and underestimates the variance of the target distribution. [Turner and Sahani, 2011]. On the other hand, EP locally minimises the forward $\mathsf{KL}(p\|q)$ (where p is the tilted distribution) and is thus penalised for missing mass of p. This property is known as mass-covering. For power EP, the story is more complicated and depends on the value of α. See also Minka [2005] for more detail discussion of this phenomenon.

Variational Family EP algorithms are limited to exponential family distributions whereas it is possible to use more flexible variational families in variational inference (e.g. normalising flows [Rezende and Mohamed, 2015]). See chapter 5 for an example. Another crucial difference is the factorisation structure of the approximation. EP leverages the factorisation of the target, while within VI we are free to choose our own approximating family. However, often fully factorised mean-field approximations are used.

Memory requirements EP maintains one approximation per factor, which can be prohibitively expensive for high-dimensional problems. However, factor tying [Li et al., 2015] or using one factor for larger parts of the data rather than a single data item, as in chapter 4, can mitigate this problem.

Overall, both variational inference and expectation propagation perform well in certain cases and dramatically less well in others and no general recommendation can be given.

Chapter 3

Relativistic Monte Carlo

The following chapter is a self-contained paper published as:

> Lu*, X., Perrone*, V., **Hasenclever, L.**, Teh, Y.W. and Vollmer, S., 2017, April. Relativistic Monte Carlo. In Artificial Intelligence and Statistics (pp. 1236-1245).

There is a statement of authorship at the end of this chapter.

Abstract

Hamiltonian Monte Carlo (HMC) is a popular Markov chain Monte Carlo (MCMC) algorithm that generates proposals for a Metropolis-Hastings algorithm by simulating the dynamics of a Hamiltonian system. However, HMC is sensitive to large time discretizations and performs poorly if there is a mismatch between the spatial geometry of the target distribution and the scales of the momentum distribution. In particular the mass matrix of HMC is hard to tune well.

In order to alleviate these problems we propose relativistic Hamiltonian Monte Carlo, a version of HMC based on relativistic dynamics that introduces a maximum velocity on particles. We also derive stochastic gradient versions of the algorithm and show that the resulting algorithms bear interesting relationships to gradient clipping, RMSprop, Adagrad and Adam, popular optimisation methods in deep learning. Based on this, we develop relativistic stochastic gradient descent by taking the zero-temperature limit of relativistic stochastic gradient Hamiltonian Monte Carlo. In experiments we show that the relativistic algorithms perform better than classical Newtonian variants and Adam.

3.1 Introduction

Markov chain Monte Carlo (MCMC) techniques based on continuous-time physical systems allow the efficient simulation of posterior distributions, and are an important mainstay of Bayesian machine learning and statistics. Hamiltonian Monte Carlo (HMC) [Duane et al., 1987a, Neal, 2011, Carpenter et al., 2017, Hoffman and Gelman, 2014, Livingstone et al., 2016] is based on Newtonian dynamics on a frictionless surface, and has been argued to be more efficient than techniques based on diffusions [Xifara et al., 2014]. On the other hand, stochastic gradient MCMC techniques based on diffusive dynamics [Welling and Teh, 2011, Ma et al., 2015, Ding et al., 2014, Chen et al., 2014] have allowed scalable Bayesian learning using mini-batches.

An important consideration when designing such MCMC algorithms is adaptation or tuning to the geometry of the space under consideration [Girolami and Calderhead, 2011, Beskos et al., 2013, Patterson and Teh, 2013]. To give a concrete example, consider HMC. Let $f(\theta)$ be a target density that can be written as $f(\theta) \propto e^{-U(\theta)}$ where $U(\theta)$ is interpreted as the potential energy of a particle in location θ. HMC introduces an auxiliary momentum variable p so that the joint distribution is $f(\theta, p) \propto e^{-H(\theta,p)}$ where the Hamiltonian is $H(\theta, p) = U(\theta) + \frac{1}{2m}p^\top p$. The quantity $\frac{1}{2m}p^\top p$, where m is the mass of the particle, represents the kinetic energy. Denoting by $\dot{\theta}$ and $\dot{p}$ the time derivative of θ and p, the leapfrog discretisation [Neal, 2011] of Hamilton's equations $\dot{\theta} = \frac{\partial H}{\partial p}$ and $\dot{p} = -\frac{\partial H}{\partial \theta}$ gives

$$\begin{aligned} p_{t+1/2} &\leftarrow p_t - \tfrac{1}{2}\epsilon \nabla U(\theta_t), \\ \theta_{t+1} &\leftarrow \theta_t + \epsilon \tfrac{p_{t+1/2}}{m}, \\ p_{t+1} &\leftarrow p_{t+1/2} - \tfrac{1}{2}\epsilon \nabla U(\theta_{t+1}) \end{aligned}$$

where ϵ is the time discretisation and the velocity is $\frac{p_{t+1/2}}{m}$. If m is too small, the particle travels too fast leading to an accumulation of discretisation error. To compensate, ϵ needs to be set small and the computational cost required increases. On the other hand, if m is too large, the particle travels slowly resulting in slow mixing for the resulting Markov chain. While the mass parameter can be tuned, e.g., to optimise acceptance rate according to theory [Beskos et al., 2013], it only incidentally controls the velocity which ultimately affects the discretisation error and algorithm stability.

In this paper, we are interested in making MCMC algorithms based on physical simulations more robust by directly controlling the velocity of the particle. This is achieved by replacing the Newtonian dynamics in HMC with the relativistic dynamics

[Einstein, 1905], whereby particles cannot travel faster than the "speed of light". We also develop relativistic variants of stochastic gradient MCMC algorithms and show that they work better and are more robust than the classical Newtonian variants.

The relativistic MCMC algorithms we develop have interesting relationships with a number of optimisation algorithms popular in deep learning. Firstly, the maximum allowable velocity (speed of light) is reminiscent of gradient clipping [Pascanu et al., 2013]. Our framework gives Bayesian alternatives to gradient clipping, in the sense that our algorithms demonstrably sample from instead of optimising the target distribution (exactly or approximately). Secondly, the resulting formulas (see (3.3)), which include normalisations by L_2 norms, bear strong resemblances to (but are distinct from) RMSprop, Adagrad and Adam [Tieleman and Hinton, 2012, Duchi et al., 2011, Kingma and Ba, 2015]. Motivated by these connections, we develop a relativistic stochastic gradient descent (SGD) algorithm by taking the zero-temperature limit of relativistic SGHMC, and show in experiments on feedforward networks trained on MNIST that it achieves better performance than Adam.

3.2 Relativistic Hamiltonian Dynamics

Our starting point is the Hamiltonian that governs the dynamics in special relativity [Einstein, 1905],

$$H(\theta, p) = U(\theta) + K(p) \tag{3.1}$$

$$K(p) = mc^2 \left(\frac{p^\top p}{m^2 c^2} + 1 \right)^{\frac{1}{2}} \tag{3.2}$$

where the target density is $f(\theta) \propto e^{-U(\theta)}$, for $\theta \in \mathbb{R}^d$ interpreted as the position of the particle, $p \in \mathbb{R}^d$ is a momentum variable, and $K(p)$ is the relativistic kinetic energy. The two tunable hyperparameters are a scalar "rest mass" m and the "speed of light" c which bounds the particle's speed. The joint distribution $f(\theta, p) \propto e^{-H(\theta,p)}$ is separable, with the momentum variable having marginal distribution $\propto e^{-K(p)}$, a multivariate generalisation of the symmetric hyperbolic distribution.

The resulting dynamics is given by Hamilton's equations, which read

$$\begin{aligned} \dot{\theta} &= \frac{\partial H}{\partial p} = M^{-1}(p)p, & M(p) &= m \left(\frac{p^\top p}{m^2 c^2} + 1 \right)^{\frac{1}{2}} \\ \dot{p} &= -\frac{\partial H}{\partial \theta} = -\nabla U(\theta), \end{aligned} \tag{3.3}$$

where $M(p)$ can be interpreted as the relativistic mass and $M^{-1}(p)p$ as the velocity of the particle (c.f. the velocity under Newtonian dynamics is $m^{-1}p$). Note that the relativistic mass is lower bounded by and increases asymptotically to $\|p\|/c$ as the momentum increases, so that the speed $M^{-1}(p)\|p\|$ is upper bounded by and asymptototes to c. On the other hand, the larger the rest mass m, the smaller the typical "cruising" speed of the particle. Conversely, as $m \to 0$ the particle will travel at the speed of light at all times, i.e. it behaves like a photon. This gives an intuition for tuning both hyperparameters c and m based on knowledge about the length scale of the target density: we choose c as an upper bound on the speed at which the parameter of interest θ changes at each iteration, while we choose m to control the typical sensible speed at which the parameter changes. We will demonstrate this intuition in the experimental Section 3.5.

A crucial property of the relativistic dynamics is that the mass is a function of the momentum. We will show in the next sections that this represents a simple way to adapt to the geometry of the target distribution that, as opposed to differential geometric approaches [Girolami and Calderhead, 2011], does not require high order geometric information and more complex integrators.

In very high dimensional problems (e.g. those in deep learning, collaborative filtering or probabilistic modelling), the maximum overall speed imposed on the system might need to be very large so that reasonably large changes in each coordinate are possible at each step of the algorithm. This means that each coordinate could in principle achieve a much higher speed than desirable. An alternative approach is to upper bound the speed at which each coordinate changes by choosing the following relativistic kinetic energy

$$K(p) = \sum_{j=1}^{d} m_j c_j^2 \left(\frac{p_j^2}{m_j^2 c_j^2} + 1 \right)^{\frac{1}{2}}, \tag{3.4}$$

where j indexes the coordinates of the d-dimensional system, and each coordinate can have its own mass m_j and speed of light c_j. This leads to the same Hamiltonian dynamics (3.3), but with all variables interpreted as vectors, and all arithmetic operations interpreted as element-wise operations. Experimental results will be based on this independent-momenta variant, which showed consistently better performance. For simplicity, in the theoretical sections we will describe only the version (3.2).

3.2.1 Relativistic Hamiltonian Monte Carlo

As a demonstration of the relativistic Monte Carlo framework, we derive a relativistic variant of the Hamiltonian Monte Carlo (HMC) algorithm [Neal, 2011, Duane et al., 1987a]. In the following, we will refer to all classical variants as Newtonian, since they follow Newtonian dynamics (e.g. *Newtonian HMC (NHMC)* vs *relativistic HMC (RHMC)*).

Each iteration of HMC involves first sampling the momentum variable, followed by a series of L leapfrog steps, followed by a Metropolis-Hastings accept/reject step. The momentum can be simulated by first simulating the speed $\|p\|$ followed by simulating p uniformly on the sphere with radius $\|p\|$. The speed $\|p\|$ has marginal distribution given by a symmetric hyperbolic distribution, for which specialised random variate generators exist. Alternatively, the density is log-concave, and we used adaptive rejection sampling to simulate it. The leapfrog steps [Sanz-Serna and Calvo, 1994] with stepsize ϵ follows (3.3) directly: set θ_0, p_0 to the current location and momentum and for $t = 1, \ldots, L$,

$$\begin{aligned} p_{t+1/2} &\leftarrow p_t - \tfrac{1}{2}\epsilon \nabla U(\theta_t) \\ \theta_{t+1} &\leftarrow \theta_t + \epsilon M^{-1}(p_{t+1/2}) p_{t+1/2} \\ p_{t+1} &\leftarrow p_{t+1/2} - \tfrac{1}{2}\epsilon \nabla U(\theta_{t+1}) \end{aligned}$$

The leapfrog steps leave the Hamiltonian H approximately invariant and are volume-preserving [Leimkuhler and Shang, 2016], so that the MH acceptance probability is simply $\min(1, \exp(-H(\theta_L, p_L) + H(\theta_0, p_0)))$.

Observe that the momentum p is unbounded and may become very large in the presence of large gradients in the potential energy. However, the size of the θ update is bounded by ϵc and therefore the stability of the proposed sampler can be controlled. This behaviour is essential for good algorithmic performance on complex models such as neural networks, where the scales of gradients can vary significantly across different parameters and may not be indicative of the optimal scales of parameter changes. This is consistent with past experiences optimising neural networks, where it is important to adapt the learning rates individually for each parameter so that typical parameter changes stay in a sensible range [Pascanu et al., 2013, Tieleman and Hinton, 2012, Duchi et al., 2011, Kingma and Ba, 2015, Şimşekli et al., 2016]. Such adaptation techniques have also been explored for stochastic gradient MCMC techniques [Li et al., 2016a, Hasenclever et al., 2017a], but we will argue in Sections 3.3.2 and 3.5 that they introduce another form of instability that is not present in the relativistic approach.

3.3 Relativistic Stochastic Gradient Markov Chain Monte Carlo

In recent years stochastic gradient MCMC (SGMCMC) algorithms have been very well explored as methods to scale up Bayesian learning by using mini-batches of data [Welling and Teh, 2011, Chen et al., 2014, Ding et al., 2014, Ma et al., 2015, Shang et al., 2015]. In this section we develop relativistic variants of SGHMC [Chen et al., 2014] and SGNHT [Ding et al., 2014, Shang et al., 2015]. These algorithms include momenta, which serve as reservoirs of previous gradient computations and thus can integrate and smooth out gradient signals from previous mini-batches of data. As noted earlier, because the momentum can be large, particularly as the stochastic gradients can have large variance, the resulting updates to θ can be overly large, and small values of the step size are required for stability, leading potentially to slower convergence. This motivates our development of relativistic variants.

We make use of the framework of [Ma et al., 2015] for deriving SGMCMC algorithms. Let z be a collection of variables with target distribution $f(z) \propto e^{-H(z)}$. Let $D(z)$ be a symmetric positive-definite diffusion matrix and $Q(z)$ be a skew-symmetric matrix describing an energy-conserving dynamics. Ma et al. [2015] showed that a diffusion process has $f(z)$ as its equilibrium distribution if and only if it has the following form:

$$\begin{aligned} dz &= -[D(z) + Q(z)]\nabla H(z)dt + \Gamma(z)dt + \sqrt{2D(z)}dW \\ \Gamma_i(z) &= \sum_{j=1}^{d} \frac{\partial[D_{ij}(z)+Q_{ij}(z)]}{\partial z_j}, \end{aligned} \tag{3.5}$$

where W is the d-dimensional Wiener process (Brownian motion) and $\Gamma(z)$ is a correction factor which can be computed from D and Q. To derive a correct SGMCMC algorithm, one just has to choose D and Q, compute Γ, discretise the SDE, and substitute a stochastic estimate $\nabla\tilde{U}(z)$ for $\nabla U(z)$. The stochastic gradient has asymptotically negligible variance compared to the noise injected by W for small step sizes. In the following, we will derive a range of relativistic algorithms using different choices of the state space and D and Q.

As a historical aside, the work of Ma et al. [2015] brings together beautifully a number of disparate partial results in the applied mathematics and physics literatures which we would like to highlight. In the case of constant diffusion matrix, $D(z) = D_0$, the result goes back to Hwang et al. [1993]. Moreover, the characterization result of Ma et al. [2015] follows from two observations. The first is to characterise SDEs

with a prescribed invariant distribution using implicit divergence free vector fields, which appeared in, e.g. Pavliotis [2014]. The second consists of an explicit expression of these vector fields in terms of skew-symmetric "steam" matrices Q. This is well known in the stochastic fluid dynamic literature, see e.g. Komorowski et al. [2012].

3.3.1 Relativistic Stochastic Gradient HMC

Suppose our noisy gradient estimate $\nabla\tilde{U}(\theta)$ of $\nabla U(\theta)$ is based on a minibatch of data. Then, appealing to the central limit theorem, we can assume that $\nabla\tilde{U}(\theta) \approx \nabla U(\theta) + \mathcal{N}(0, B(\theta))$. Let $z = (\theta, p)$ and $H(z)$ be the relativistic Hamiltonian in (3.2). Choosing

$$D(z) = \begin{pmatrix} 0 & 0 \\ 0 & D \end{pmatrix}, \quad Q(z) = \begin{pmatrix} 0 & -I \\ I & 0 \end{pmatrix}, \tag{3.6}$$

and thus $\Gamma(z) = \mathbf{0}$, where D is a fixed symmetric diffusion matrix results in the following relativistic SGHMC dynamics:

$$\begin{pmatrix} d\theta \\ dp \end{pmatrix} = \begin{pmatrix} M^{-1}(p)p \\ -\nabla U(\theta) - DM^{-1}(p)p \end{pmatrix} dt + \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{2D} \end{pmatrix} dW_t.$$

Using a simple Euler-Maruyama discretisation, the relativistic SGHMC algorithm is,

$$\begin{aligned} p_{t+1} \leftarrow\ & p_t - \epsilon_t \nabla\tilde{U}(\theta_t) - \epsilon_t D M^{-1}(p_t) p_t \\ & + \mathcal{N}(0, \epsilon_t(2D - \epsilon_t \hat{B}_t)) \\ \theta_{t+1} \leftarrow\ & \theta_t + \epsilon_t M^{-1}(p_{t+1}) p_{t+1} \end{aligned} \tag{3.7}$$

where $\hat{B}$ is an estimate of the noise coming from the stochastic gradient $B(\theta)$. The term $DM^{-1}(p)p$ can be interpreted as friction, which prevents the kinetic energy from building up and corrects for the noise coming from the stochastic gradient.

It is useful to compare RSGHMC with preconditioned SGLD [Li et al., 2016a, Hasenclever et al., 2017a], which attempts to adapt the SGLD algorithm to the geometry of the space, using adaptations similar to RMSProp, Adagrad or Adam. The relevant term above is the update $M^{-1}(p_{t+1})p_{t+1}$ to θ_{t+1}:

$$M^{-1}(p_{t+1})p_{t+1} = \frac{p_{t+1}}{\sqrt{\frac{p_{t+1}^\top p_{t+1}}{c^2} + m^2}}. \tag{3.8}$$

Note the surprising similarity to RMSProp, Adagrad and Adam, with the main difference being that the relativistic mass adaptation uses the current momentum instead of being separately estimated using the square of the gradient. This has the advantage

that the relativistic SGHMC enforces a maximum speed of change. In contrast, as we also observe in Section 3.5, preconditioned SGLD has the following failure mode: when the gradient is small, the adaptation scales up the gradient so that the gradient update has a reasonable size. However it also scales up the injected noise, which can end up being significantly larger than the gradient update, and making the algorithm unstable.

3.3.2 Relativistic Stochastic Gradient Descent (with Momentum)

Motivated by the relationship to RMSprop, Adagrad and Adam, we develop a relativistic stochastic gradient descent (RSGD) algorithm with momentum by taking the zero-temperature limit of the RSGHMC dynamics. This idea connects to Santa [Chen et al., 2016], a recently developed algorithm where an annealing scheme on the system temperature makes it possible to obtain a stochastic optimization algorithm starting from a Bayesian one.

From thermodynamics [Laurendeau, 2005], the canonical (Gibbs Boltzmann) density is proportional to $e^{-\beta U(z)}$, where β is the inverse temperature. Previously we have been using $\beta = 1$, which corresponds to the posterior distribution. For general β,

$$\begin{pmatrix} d\theta \\ dp \end{pmatrix} = \begin{pmatrix} \beta M^{-1}(p)p \\ \beta\left(-\nabla U(\theta) - DM^{-1}(p)p\right) \end{pmatrix} dt + \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{2D} \end{pmatrix} dW.$$

By taking $\beta \to \infty$ the target distribution becomes more peaked around the MAP estimator. Simulated annealing [Geman and Hwang, 1986, Andrieu and Doucet, 2000, Chen et al., 2016], which increases $\beta \to \infty$ over time, forces the sampler to converge to a MAP estimator. We can derive RSGD by rescaling time as well, guaranteeing a non-degenerate limit process. Letting $\hat{\theta}(t) = \theta(\beta t)$, $\hat{p}(t) = p(\beta t)$, so that

$$\begin{pmatrix} d\hat{\theta} \\ d\hat{p} \end{pmatrix} = \begin{pmatrix} M^{-1}(\hat{p})\hat{p} \\ -\nabla U(\hat{\theta}) - DM^{-1}(\hat{p})\hat{p} \end{pmatrix} dt + \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{\frac{2D}{\beta}} \end{pmatrix} dW$$

and letting $\beta \to \infty$, we obtain the following ODE:

$$\begin{pmatrix} d\theta \\ dp \end{pmatrix} = \begin{pmatrix} M^{-1}(p)p \\ -\nabla U(\theta) - DM^{-1}(p)p \end{pmatrix} dt$$

Discretising the above then gives RSGD. Notice that if the above converges, i.e. $\dot{\theta} = \dot{p} = 0$, it does so at a critical point of U. Similar to other adaptation schemes, RSGD adaptively rescales the learning rates for different parameters, which enables effective learning especially in high dimensional settings. Moreover, the update in each iteration is upper bounded by the speed of light. Our algorithm differs from others through the use of a momentum, and adapting based on the momentum instead of the average of squared gradients.

3.4 A Stochastic Gradient Nosé-Hoover Thermostat for Relativistic Hamiltonian Monte Carlo

Borrowing a second concept from physics, SGHMC can be improved by introducing a dynamic variable ξ that adaptively increases or decreases the momenta. The new variable ξ can be thought of as a *thermostat* in a statistical physics setting and its dynamics expressed as

$$d\xi = \frac{1}{d}\left(p^T p - d\right) dt. \tag{3.9}$$

The idea is that the system adaptively changes the friction for the momentum, 'heating' or 'cooling down' the system. The dynamics of this new variable, known as Nosé-Hoover [Leimkuhler and Reich, 2009] thermostat due to its links to statistical physics, has been shown to be able to remove the additional bias due to the stochastic gradient provided that the noise is isotropic Gaussian and spatially constant [Ding et al., 2014, Leimkuhler and Shang, 2016]. In general, the noise is neither Gaussian, spatially constant or isotropic. Nevertheless, there is numerical evidence that the thermostat increases stability and mixing. Heuristically, the dynamics for ξ can be motivated by the fact that in equilibrium we have

$$\begin{aligned}
\mathbb{E}\left[\frac{\partial^2 K}{\partial p_i^2}\right] &= \int \frac{\partial^2 K}{\partial p_i^2} e^{-K(p)} dp \\
&= -\int \frac{\partial K}{\partial p_i}\left(-\frac{\partial K}{\partial p_i} e^{-K(p)}\right) dp \\
&= \mathbb{E}\left[\left(\frac{\partial K}{\partial p_i}\right)^2\right].
\end{aligned}$$

Since $\frac{\partial K}{\partial p_i} = p_i$ and $\frac{\partial^2 K}{\partial p_i^2} = 1$ this implies

$$\mathbb{E}\left[\frac{d\xi}{dt}\right] = \mathbb{E}\left[\frac{1}{d}\sum_i \left(\left(\frac{\partial K}{\partial p_i}\right)^2 - \frac{\partial^2 K}{\partial p_i^2}\right)\right] = 0$$

The additional dynamics pushes the system towards $\frac{d\xi}{dt} = 0$, suggesting that the distribution will be moved closer to the equilibrium. This gives a recipe for a stochastic gradient Nosé-Hoover thermostat with a general kinetic energy $K(p)$.

We first augment the Hamiltonian with ξ:

$$H(q, p, \xi) = U(q) + K(p) + \frac{d}{2}(\xi - D)^2.$$

We are now in the position to derive the SDE preserving the probability density $\propto \exp(-H)$ by adopting the framework of Ma et al. [2015] and defining:

$$\begin{aligned}
H(\theta, p, \xi) &= U(\theta) + K(p) + \frac{d}{2}(\xi - D)^2 \\
D(\theta, p, \xi) &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & D \cdot I & 0 \\ 0 & 0 & 0 \end{pmatrix} \\
Q(\theta, p, \xi) &= \begin{pmatrix} 0 & -I & 0 \\ I & 0 & \nabla K(p)/d \\ 0 & -\nabla K(p)^T/d & 0 \end{pmatrix}.
\end{aligned}$$

From (3.5) it follows that $\Gamma = (0\ 0\ -\Delta K(p)/d)^T$ and the dynamics becomes

$$\begin{aligned}
\begin{pmatrix} d\theta \\ dp \\ d\xi \end{pmatrix} = &\begin{pmatrix} \nabla K(p) \\ -\nabla \tilde{U} dt - \xi \nabla K(p) \\ \frac{1}{d}\left(\|\nabla K(p)\|^2 - \Delta K(p)\right) \end{pmatrix} dt \\
&+ \begin{pmatrix} 0 & 0 & 0 \\ 0 & \sqrt{2D} & 0 \\ 0 & 0 & 0 \end{pmatrix} dW_t
\end{aligned}$$

where Δ is the Laplace operator defined as $\Delta K(p) = \sum_i \frac{\partial^2 K(p)}{\partial p_i^2}$. For the relativistic kinetic energy $K(p)$, we have that $\nabla_p K(p) = M^{-1}(p)p$ with $M(p) := m\left(\frac{p^T p}{m^2 c^2} + 1\right)^{\frac{1}{2}}$ a scalar and that $\Delta K(p) = tr\left(\frac{d}{dp}\left(\frac{1}{d} M^{-1}(p)p\right)\right)$. The Stochastic Gradient Nosé-Hoover Thermostat for relativistic HMC follows:

$$\begin{aligned}
\begin{pmatrix} d\theta \\ dp \\ d\xi \end{pmatrix} = &\begin{pmatrix} M^{-1}(p)p \\ -\nabla \tilde{U} - \xi M^{-1}(p)p \\ \frac{p^T p}{d}\left(M^{-2}(p) + c^{-2} M^{-3}(p)\right) - M^{-1}(p) \end{pmatrix} dt \\
&+ \begin{pmatrix} 0 & 0 & 0 \\ 0 & \sqrt{2D} & 0 \\ 0 & 0 & 0 \end{pmatrix} dW_t.
\end{aligned}$$

3.5 Experiments

3.5.1 Synthetic data

All the experimental results in this section are based on the independent-momenta versions (3.4) as they give superior results. We first explore the performances of the algorithms on a set of small examples including a two dimensional banana function (Banana) Haario et al. [1999] with density $p(\mathbf{x}) \propto \exp\{-0.5(0.01x_1^2 + (x_2 + 0.1x_1^2 - 10)^2)\}$, and Gaussian mixture models (GMM1, GMM2, GMM3) obtained by combining the three following Gaussian random variables with equal mixing proportions: $\mathcal{N}(-5, 1/\sigma^2)$, $\mathcal{N}(0, \sigma^2)$, $\mathcal{N}(5, 1/\sigma^2)$, where $\sigma^2 = 1, 0.5, 0.3$. When $\sigma^2 = 1$ the three Gaussians have the same variance, while a lower σ^2 means larger discrepancies between their variances and thus a wider range of length scales and log density gradients. The density sgmcmc/plots of the examples can be found in the top row of Figure 3.3.

We start with an exploration of the behaviour of RHMC as the tuning parameters m, c and ϵ are varied. First we considered the effective sample sizes (ESS) of the algorithm on the Banana and GMM1 datasets. We varied both ϵ and $\epsilon \times c$ over a grid, and computed the average ESS, over 20 chains, each of length 10^4 for Banana, and over 100 chains of length 10^5 for GMM1. The ESS contour sgmcmc/plots can be found in Figure 3.1, which suggests that ϵc and ϵ can be independently tuned. While ϵ controls the time discretisation of the continuous-time dynamics, ϵc controls the maximum change in the parameters at each leapfrog step. Next we varied the mass parameter m for GMM1, showing sgmcmc/plots in Figure 3.2. As expected the ESS is optimised at an intermediate value of m, and the average "cruising speed" $\bar{v}$ decreases with m. In order to understand how to tune m, on the third panel we overlaid two contour sgmcmc/plots: one for ESS and the other for $\bar{v}$. We see that the cruising speed $\bar{v}$ correlates much better with the ESS than m does, which suggests that m should be tuned via $\bar{v}$, e.g. by the user specifying a desired value for $\bar{v}$ and m being adapted to achieve the speed (noting that m and $\bar{v}$ have a monotonic relationship, which makes for easy adaptation).

We next compare the performances of NHMC and RHMC for a wide range of stepsizes, via the ESS (higher better), the *mean absolute error (MAE)* between the true probabilities and the histograms of the sample frequencies (lower better), and the log Stein discrepancy [Gorham and Mackey, 2015] (lower better). The Stein discrepancy is a more accurate measure of sample quality, the reason being that it can be used to bound the Wasserstein distance, thus accounting for bias and insufficient exploration of the target. The results can be found in rows 2-4 of Figure 3.3. It can

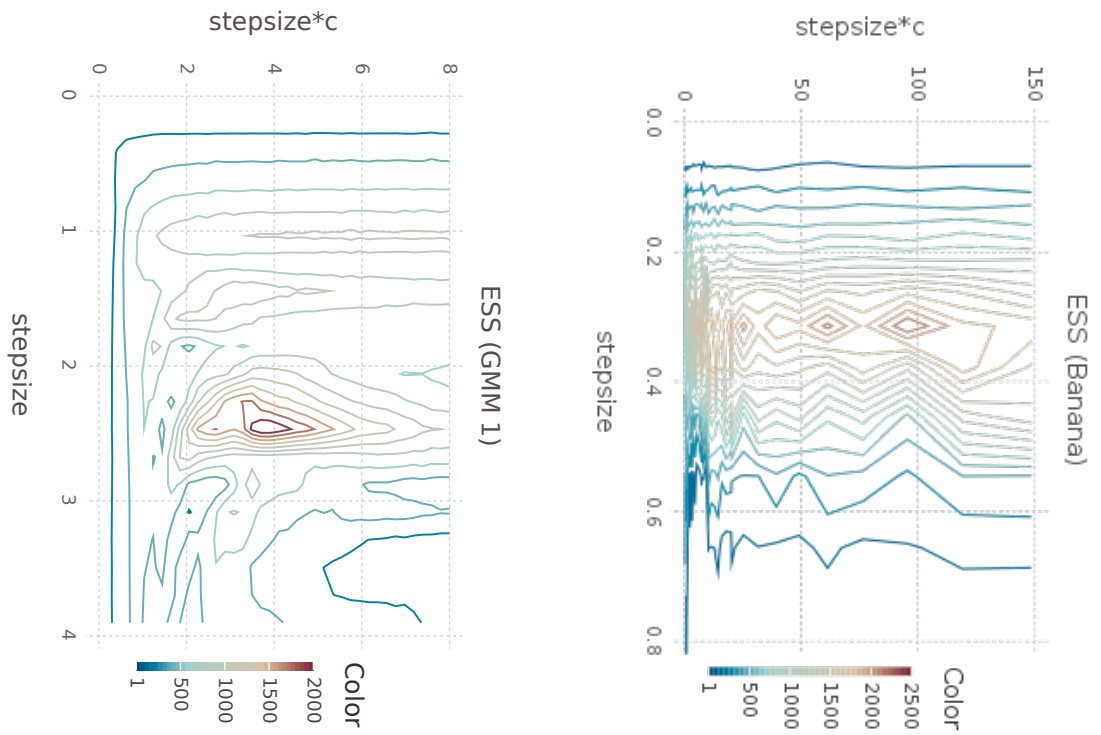


Figure 3.1: ESS contour sgmcmc/plots of $\epsilon \times c$ versus ϵ for Banana and GMM1 models.

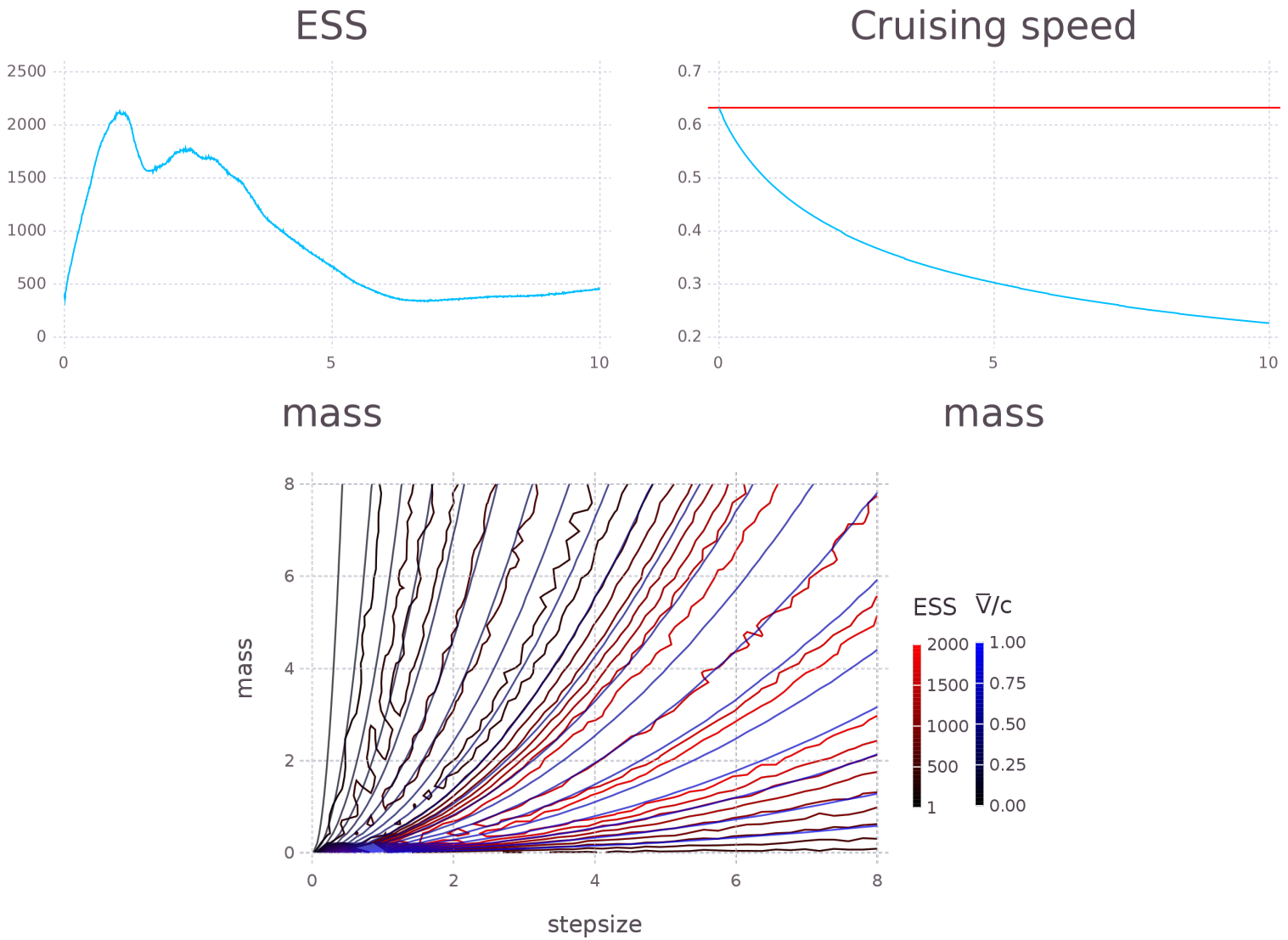


Figure 3.2: Varying m for GMM1. From left to right: ESS, cruising speed (the red horizontal line is c), and ESS and relative cruising speed $\bar{v}/c$ contour plots versus m and ϵ.

Figure 3.3: Left to right: Banana, GMM1, GMM2, GMM3 datasets. Top to bottom: density plot, ESS versus step size ϵ, MAE versus ϵ, log Stein discrepancy versus ϵ.

be seen that RHMC achieves better performance and is strikingly more robust to the step size ϵ than NHMC. As expected, this behaviour is particularly pronounced when the step size is large. Moreover, when the gradients of the target model span a large range of values (GMM2, GMM3), the improvements yielded by the relativistic variants are more pronounced. These results confirm that, since the speed of particles is bounded by c, RHMC is less sensitive to the presence of large gradients in the target density and more stable with respect to the choice of ϵ, allowing for a more efficient exploration of the target density.

Next we compare both the Newtonian and relativistic variants of HMC and SGMCMC algorithms on a simulated 3-dimensional logistic regression example with 500 observations. For the stochastic versions of the algorithms, we use mini-batches of size 100. After a burn-in period of 1000 iterations, we calculated the Stein discrepancy for different ϵ while keeping the product $\epsilon \times c$ fixed. To make a fair comparison in terms of computational time, we used 200 samples for NHMC and RHMC and 1000 samples for the SGMCMC algorithms. From Figure 3.4, we see that the relativistic variants are significantly more robust than the Newtonian variants. The NHT algorithms were able to correct for stochastic gradient noise and performed better than SGHMC algorithms. Particularly, RSGNHT had lower Stein discrepancies than the other algorithms for most values of ϵ.

3.5.2 Neural Networks

Turning to more complex models, we first considered a neural network with 50 hidden units and initialised its weights by the widely used Xavier initialisation. We used the Pima Indians dataset for binary classification (552 observations and 8 covariates) to compare the relativistic and the preconditioning approach. Indeed, these methods represent two different ways to normalise gradients so that the update sizes are reasonable for the local lengthscale of the target distribution. In particular we consider SGLD Adam, namely a preconditioned SGLD algorithm with an additional Adam-style debiasing of the preconditioner. Figure 3.5 compares the test-set accuracy of SGLD Adam with RSGD and RHMC, showing that the first is significantly outperformed by the relativistic algorithms. Due to Xavier initialization all of the weights are small, which causes small gradients; therefore, the injected noise becomes very large due to the rescaling by the inverse of squared root of the average gradients, which makes SGLD Adam unstable. The histograms reveal that at the first iteration the preconditioning causes the weights to become extremely large and this strongly compromises the performance of SGLD Adam, which takes a long time to recover.

The relativistic framework represents therefore a much better approach to perform adaptation of the learning rates specific to each parameter.

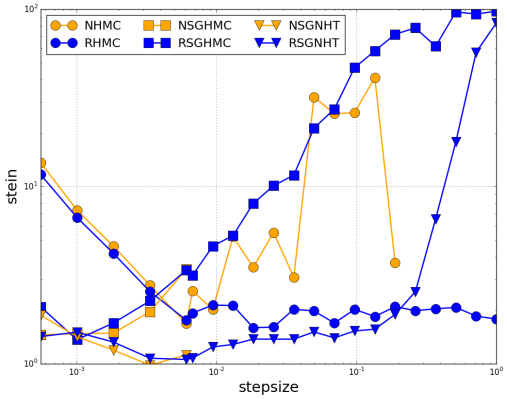


Figure 3.4: Stein discrepancy versus stepsize ϵ for logistic regression. NSGHMC and NSGNHT were unstable for $\epsilon > 6 \times 10^{-3}$ and thus their Stein discrepancies were not plotted.

We then apply our algorithms to the standard MNIST dataset, which consists of 28×28 handwritten digital images from 10 classes with a training set of size $60,000$ and a test set of size $10,000$. We tested our optimization algorithm on a single layer with 100 hidden units and two multi-layer neural networks with $500 * 300$ and $400 * 400$ hidden units. In Figure 3.6 a comparison with Adam and Santa [Chen et al., 2016] is displayed (their relation is discussed in more detail in Section 3.3.2). As the goal of these experiments is to compare stochastic optimizers, we consider Santa SGD, namely Santa in its zero-temperature limit. In addition, we adopt an Euler integration scheme for all algorithms, since the improvements yielded by higher-order integrators are orthogonal to the specific algorithm. It can be observed that our algorithm is competitive with Adam and is able to achieve a lower error rate, particularly with the 100 hidden units architecture. Moreover, RSGD performs significantly better than Santa SGD on all the considered architectures.

3.6 Conclusion

Our numerical experiments demonstrate that the relativistic algorithms developed in this paper are much more stable and robust to the choice of parameters and noise in stochastic gradients compared to the Newtonian counterparts. Moreover, while the standard Newtonian algorithms also require parameter tuning, in the relativistic setting we have a clearer understanding of how each parameter affects performance:

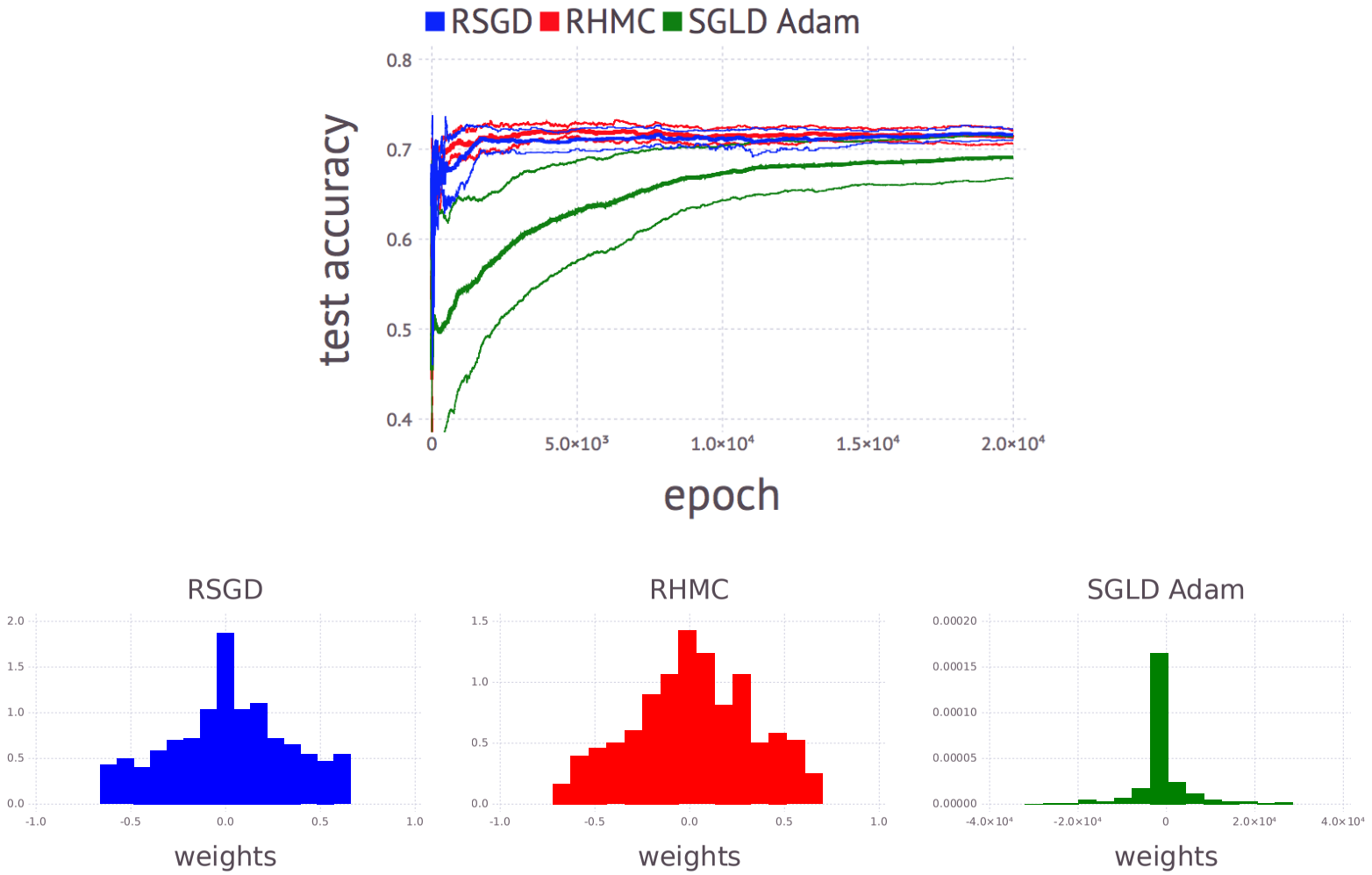


Figure 3.5: Comparison between RSGD, RHMC and SGLD Adam on the Pima Indians dataset using 50 hidden units. The histograms show the neural network weights at the first iteration.

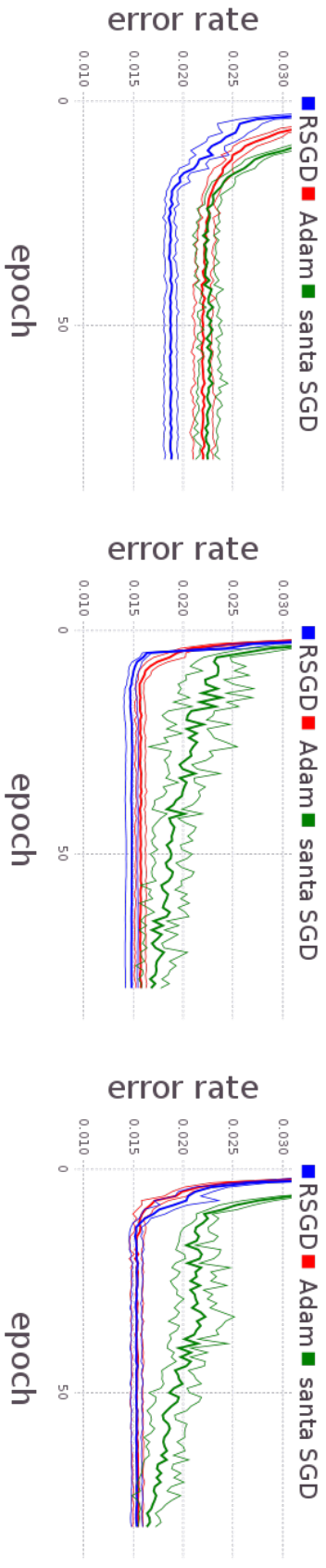


Figure 3.6: Comparison of the test-set error rate on the MNIST dataset. From left to right: 100 hidden units; $500 * 300$ hidden units; $400 * 400$ hidden units.

ϵc and m respectively control the maximal parameter change and the average speed. This direct interpretation simplifies tuning and is used consistently across experiments to select parameters, achieving improved performance for a wider range of parameters.

The connection of our algorithms with popular stochastic optimizers such as Adam and RMSProp is novel and gives an interesting perspective to understand them. Moreover, the relativistic stochastic gradient descent showed to be very competitive with state of the art stochastic gradient methods for fitting neural nets. We anticipate a variety of algorithms with different kinetic energies to be developed following our work.

Each of the proposed methodologies has scope for further research. The HMC version of the algorithm could be improved by some more advanced HMC methodology such as the NUTS version [Hoffman and Gelman, 2014] and partial moment refreshment instead of Adaptive Rejection Sampling [Neal, 2011]. Additionally, better numerical integration schemes could be employed. Finally, we leave an in-depth ergodic theory as an interesting avenue for future work.

Acknowledgements XL thanks the PAG scholarship and New College for support. VP and LH are funded by the EPSRC CDT OxWaSP through EP/L016710/1. SJV thanks EPSRC for funding through EPSRC grants EP/N000188/1 and EP/K009850/1. YWT acknowledges the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 617071 and EPSRC for funding through EP/K009362/1.

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	Relativistic Monte Carlo
Publication Status	x Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work
Publication Details	X Lu*, V Perrone*, L Hasenclever, Y W Teh, S J Vollmer, Relativistic Monte Carlo, in AISTATS, 2017.

Student Confirmation

Student Name:	Leonard Hasenclever		
Contribution to the Paper	I was responsible for the initial implementation and the work on the thermostat extension. I contributed to the analysis and writing the manuscript. Most of the experiments in the paper were performed by Xiaoyu Lu and Valerio Perrone.		
Signature		Date	

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: Professor Yee Whye Teh		
Supervisor comments		
Signature	Date	

Chapter 4

Stochastic Natural Gradient Expectation Propagation

The following chapter is a self-contained paper published as:

> **Hasenclever, L.**, Webb, S., Lienart, T., Vollmer, S., Lakshminarayanan, B., Blundell, C. and Teh, Y.W., 2017. Distributed Bayesian learning with stochastic natural gradient expectation propagation and the posterior server. The Journal of Machine Learning Research, 18(1), pp.3744-3780.

There is a statement of authorship at the end of this chapter.

Abstract

This paper makes two contributions to Bayesian machine learning algorithms. Firstly, we propose stochastic natural gradient expectation propagation (SNEP), a novel alternative to expectation propagation (EP), a popular variational inference algorithm. SNEP is a black box variational algorithm, in that it does not require any simplifying assumptions on the distribution of interest, beyond the existence of some Monte Carlo sampler for estimating the moments of the EP tilted distributions. Further, as opposed to EP which has no guarantee of convergence, SNEP can be shown to be convergent under certain albeit impractical conditions, provided the objective is bounded. Secondly, we propose a novel architecture for distributed Bayesian learning which we call the posterior server. The posterior server allows scalable and robust Bayesian learning in cases where a data set is stored in a distributed manner across a cluster, with each compute node containing a disjoint subset of data. An independent Monte Carlo sampler is run on each compute node, with direct access only to the local data subset, but which targets an approximation to the global posterior

distribution given all data across the whole cluster. This is achieved by using a distributed asynchronous implementation of SNEP to pass messages across the cluster. We demonstrate SNEP and the posterior server on distributed Bayesian learning of logistic regression and neural networks.

4.1 Introduction

Algorithms and systems for enabling machine learning from large scale data sets are becoming increasingly important in the era of Big Data. This has driven many developments, including various forms of stochastic gradient descent, parallel and distributed learning systems, use of GPUs, sketching, random Fourier features, divide-and-conquer methods, as well as various approximation schemes. These large scale machine learning systems have in turn driven significant advances across many data-oriented sciences and technologies, ranging from the biological sciences, neuroscience, social sciences, signal processing, speech processing, natural language processing, computer vision etc.

In this paper we will consider methods for large scale *Bayesian* machine learning. As opposed to the more common empirical risk minimisation or maximum likelihood approaches, where learning is phrased as finding a set of parameters optimal with respect to a data set and to a loss or likelihood function, Bayesian machine learning rests upon probabilistic models which capture the dependencies among all observed and unobserved variables, and where learning is phrased as computing the posterior distribution over unobserved variables (including both latent variables and model parameters) given the observed data.

The Bayesian framework can more fully capture the uncertainty in learnt parameters and prevent overfitting. In principle this allows the use of more complex and larger scale models. However, Bayesian approaches are generally more computationally intensive than optimisation-based ones, and have to date not led to methods which are as scalable.

For complex models, exact computation of the posterior distribution is intractable and approximate schemes such as variational inference (VI) (Wainwright and Jordan, 2008), Markov chain Monte Carlo (MCMC) (Gilks et al., 1996) and sequential Monte Carlo (Doucet et al., 2001) are needed. Scalable methods in both traditions include: stochastic variational inference (Hoffman et al., 2013a, Mnih and Gregor, 2014, Rezende et al., 2014) which apply minibatch stochastic gradient descent (Robbins and Monro, 1951) to optimise the variational objective function, stochastic gra-

dient MCMC (Welling and Teh, 2011, Patterson and Teh, 2013, Ding et al., 2014, Teh et al., 2015, Leimkuhler and Shang, 2016, Ma et al., 2015, Li et al., 2016a) which uses minibatch stochastic gradients within MCMC, austerity MCMC (Korattikara et al., 2014, Bardenet et al., 2014) which uses data subsampling to reduce computational cost of Metropolis-Hastings acceptance steps, and embarrassingly parallel MCMC (Huang and Gelman, 2005, Scott et al., 2013, Wang and Dunson, 2013, Neiswanger et al., 2014) which distribute data across a cluster, runs independent MCMC samplers on each worker and combines samples across the cluster only at the end to reduce network communication costs. In addition, standard learning schemes have also been successfully deployed in large scale settings. A successful example that is particularly relevant to our work is expectation propagation (EP), introduced by (Minka, 2001a) and (Opper and Winther, 2000) which iteratively constructs approximations to a factorized distribution. EP is at the heart of the TrueSkill XBox player rating and matching system (Herbrich et al., 2007).

Our work builds upon prior work on using EP for performing distributed Bayesian learning (Xu et al., 2014, Gelman et al., 2014). In this framework, a data set is partitioned into disjoint subsets with each subset stored on a worker node in a cluster. Learning is performed at each worker based on the data subset there using MCMC sampling. As opposed to embarrassingly parallel MCMC methods which only communicate the samples to the master at the end of learning, EP is used to communicate messages (infrequently) across the cluster. These messages coordinate the samplers such that the target distributions (which coincidentally are the tilted distributions in EP) of all samplers on all workers share certain moments, e.g. means and variances, hence the name sampling via moment sharing (SMS) coined by (Xu et al., 2014). At convergence, it can be shown that the target distributions of the samplers also share moments with the EP approximation to the global posterior distribution given all data, hence the target distributions on the workers can themselves be treated as approximations to the global posterior.

While SMS works well on simpler models like Bayesian logistic regression, we have found that it did not work for more complex, high-dimensional, and non-convex models like Bayesian deep neural networks. This is due to the non-convergence of EP, particularly as the moments of the tilted distributions needed by EP are estimated using MCMC sampling, with estimation noise that further compounds the well-known lack of convergence guarantees for EP, and the fact that extremely long MCMC runs are needed for the samplers to equilibrate due to the complex posterior distribution in these models.

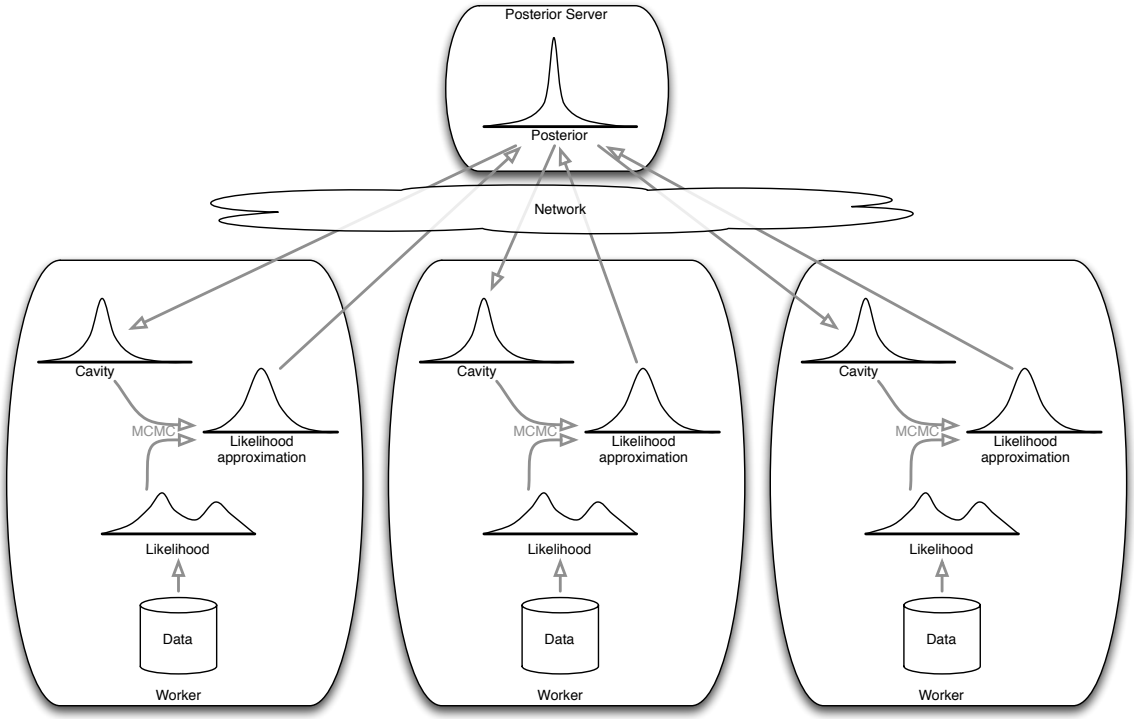


Figure 4.1: The posterior server.

Our first contribution is thus the development of stochastic natural-gradient EP (SNEP), an alternative algorithm to power EP (a generalisation of EP) (Minka, 2004) which optimises the same variational objective function. SNEP is a double-loop algorithm: the inner loop is a stochastic natural-gradient descent algorithm which tolerates estimation noise. SNEP can be shown to be convergent under certain conditions. Our derivation of SNEP improves upon the derivation of the related convergent EP algorithm of Heskes and Zoeter (2002) in that ours works for a more general class of models, we make explicit the underlying variational objective function that is being optimised, and ours uses a natural-gradient descent algorithm (Amari and Nagaoka, 2001) more tolerant of Monte Carlo noise.

Building upon the development of SNEP, our second contribution is a distributed Bayesian learning architecture which we call the posterior server. In analogy to the parameter server (Ahmed et al., 2012) which maintains and serves the parameter vector to a cluster of workers, the posterior server maintains and serves (an approximation to) the posterior distribution. Figure 4.1 gives a schematic for the steps involved. Each worker has a subset of data, from which we get a likelihood function. It also maintains a tractable approximation of the likelihood and a cavity distribution which is effectively a conditional distribution over the parameters given all data

on other workers. An MCMC sampler targets the normalised product of the cavity distribution and the (true) likelihood, and forms a stochastic estimate of the required moments, which is in turn used to update the likelihood approximation using stochastic natural-gradient descent. Each worker communicates with the posterior server asynchronously and in a non-blocking manner, sending the current likelihood approximation and receiving the new cavity distribution. This communication protocol makes more efficient use of computational resources on workers than SMS, which requires either synchronous or blocking asynchronous protocols.

Note that our set-up of the distributed learning problem is that each worker has access to a subset of data, and no worker has access to all data. This situation might occur in situations other than large scale learning. For example, when working with sensitive patient data which cannot be shared directly, we might still want to be able to make use of all available data across multiple sites to improve inference. Typically known as divide-and-conquer or consensus inference (Zhang et al., 2015b, Zhao et al., 2016, Kleiner et al., 2014, Battey et al., 2015), it is also well-known that this situation is harder than typical distributed learning settings where it is assumed that all data is accessible on all workers, which are settings assumed in DistBelief networks (Dean et al., 2012) and elastic-averaging SGD (Zhang et al., 2015a), two state-of-the-art distributed learning algorithms for neural networks.

In the following, Section 4.2 describes our set up of distributed Bayesian learning and reviews the necessary background on exponential families and convex duality. Section 4.3 formulates EP and power EP within the framework of variational inference. Our contributions are contained in Section 4.4 deriving SNEP and Section 4.5 describing the posterior server architecture. Section 4.8.2 describes additional techniques we used to make the method work on more complex problems like neural networks. We demonstrate the approach Bayesian logistic regression and Bayesian neural networks in Section 4.6. Section 4.7 concludes with a summary and discussion of future work.

4.2 Problem Set-up and Background

In this section we set-up the problem of distributed Bayesian learning, using the framework of variational inference in exponential families. For an excellent introduction to exponential families and variational inference we refer the interested reader to (Wainwright and Jordan, 2008). Section 4.2.1 reviews exponential families and con-

vex duality while introducing notation used throughout the paper. Readers familiar with these concepts can skim ahead to section 4.2.2.

4.2.1 Exponential Families and Convex Duality

.

Consider an exponential family described by a d-dimensional sufficient statistics function $s(x)$. A member p_θ of this exponential family is parameterized by a natural parameter $\theta \in \mathbb{R}^d$, and has density (with respect to some base measure, say Lebesgue),

$$p_\theta(x) = \exp\left(\theta^\top s(x) - A(\theta)\right),$$
$$A(\theta) = \log \int \exp\left(\theta^\top s(x)\right) dx.$$

The log partition function $A(\theta)$ is convex and finite on the natural domain of the exponential family,

$$\Theta := \{\theta : A(\theta) < \infty\} \subset \mathbb{R}^d,$$

which is a convex subset of $\mathbb{R}^d$.

Associated with any distribution p and the d-dimensional sufficient statistics function $s(x)$ is a mean parameter,

$$\mu := \mathbb{E}_p[s(x)],$$

where $\mathbb{E}_p$ denotes the expectation operator with respect to p. The set of valid mean parameters $\mathcal{M}$ is a closed convex set, which we refer to as the mean domain,

$$\mathcal{M} = \{\mu : \exists \text{ distribution } p \text{ with } \mu = \mathbb{E}_p[s(x)]\} \subset \mathbb{R}^d$$

Given a natural parameter $\theta \in \Theta$, the exponential family member p_θ is also associated with a mean parameter $\mu = \mathbb{E}_\theta[s(x)]$ (where $\mathbb{E}_\theta$ denotes the expectation with respect to p_θ), which we can write as a function of the natural parameters, $\mu(\theta)$. If the exponential family is minimal[1], then the mapping $\theta \mapsto \nabla A(\theta)$ is one-to-one and onto the interior of $\mathcal{M}$, and maps θ to the mean parameter, $\mu(\theta) = \nabla A(\theta)$.

We will assume that our exponential family of interest is minimal.

The convex conjugate of $A(\theta)$ is,

$$A^*(\mu) := \sup_{\theta \in \Theta} \theta^\top \mu - A(\theta).$$

[1]The exponential family is minimal if the d 1-dimensional functions making up the sufficient statistics function $s(x)$ are linearly independent, i.e. $\theta^\top s(x) = 0$ for all x implies $\theta = 0$.

Evaluated at the mean parameter $\mu(\theta)$, the conjugate is the negative entropy of p_θ,

$$A^*(\mu(\theta)) = \mathbb{E}_\theta[\log p_\theta(x)].$$

Conversely, we have

$$A(\theta) = \sup_{\mu \in \mathcal{M}} \theta^\top \mu - A^*(\mu)$$

and that the natural parameter associated with μ is $\theta = \theta(\mu) = \nabla A^*(\mu)$. In the following we will make extensive use of the duality between A and A^* and between θ and μ described above.

It is useful to write down formulae for the KL divergence between two exponential family distributions, parameterized by natural and mean parameter pairs θ, μ and θ', μ' respectively:

$$\begin{aligned}
\mathrm{KL}(p_\theta \| p_{\theta'}) &= \mathbb{E}_\theta[\log p_\theta(x) - \log p_{\theta'}(x)] \\
&= \mathbb{E}_\theta[\theta^\top s(x) - A(\theta) - (\theta')^\top s(x) + A(\theta')] \\
&= \mu^\top(\theta - \theta') - A(\theta) + A(\theta') \\
&= A^*(\mu) + A(\theta') - \mu^\top \theta' \\
&= A^*(\mu) - A^*(\mu') + (\mu' - \mu)^\top \theta'. \qquad (4.1)
\end{aligned}$$

We will write $\mathrm{KL}(\theta \| \theta'), \mathrm{KL}(\mu \| \theta')$ etc. to refer to the same KL divergence between the same two distributions.

As an example, for a diagonal covariance Gaussian of dimension $d/2$, we have

$$\begin{aligned}
p(x) &= \prod_{j=1}^{d/2} \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{1}{2\sigma_j^2}(x_j - u_j)^2\right) \\
&= \exp\left(\sum_{j=1}^{d/2} (u_j \sigma_j^{-2})(x_j) + (-\sigma_j^{-2})(\tfrac{1}{2}x_j^2) - \tfrac{1}{2}(u_j^2 \sigma_j^{-2} + \log(2\pi\sigma_j^2))\right)
\end{aligned}$$

So the sufficient statistics are x_j and $\frac{1}{2}x_j^2$, mean parameters are $\mu_{j1} = u_j$ and $\mu_{j2} = \frac{1}{2}(u_j^2 + \sigma_j^2)$, natural parameters are $\theta_{j1} = u_j \sigma_j^{-2}$ and $\theta_{j2} = -\sigma_j^{-2}$, and

$$A(\theta) = \sum_{j=1}^{d/2} \frac{1}{2}(u_j^2 \sigma_j^{-2} + \log(2\pi\sigma_j^2))$$

$$A^*(\mu) = \sum_{j=1}^{d/2} \frac{-1}{2}(1 + \log(2\pi\sigma_j^2))$$

The conversions between natural and mean parameters are:

$$u_j = \mu_{j1} = -\theta_{j1}\theta_{j2}^{-1} \qquad \theta_{j1} = \mu_{j1}(2\mu_{j2} - \mu_{j1}^2)^{-1} \qquad \mu_{j1} = -\theta_{j1}\theta_{j2}^{-1}$$
$$\sigma_j^2 = 2(\mu_{j2} - \mu_{j1}^2) = -\theta_{j2}^{-1} \qquad \theta_{j2} = -(2\mu_{j2} - \mu_{j1}^2)^{-1} \qquad \mu_{j2} = \frac{1}{2}(\theta_{j1}^2\theta_{j2}^{-2} - \theta_{j2}^{-1})$$

We will use a diagonal covariance Gaussian as our exponential family in most of our experiments, due to the high-dimensionality of the models used.

4.2.2 Distributed Bayesian Learning

We assume that our model is parameterized by a high-dimensional parameter vector x. Let the prior distribution $p_0(x)$ be a member of a tractable and minimal exponential family distribution, with natural parameter $\theta_0 \in \Theta \subset \mathbb{R}^d$, sufficient statistics function $s(x)$ and log partition function $A(\theta_0)$. Specifically, we will take $p_0(x)$ to be a diagonal covariance Gaussian. We refer to this exponential family as the *base exponential family*.

We assume that our training data set is spread across a cluster of n compute nodes or workers, with log likelihood $\ell_i(x)$ on compute node $i = 1, \ldots, n$. For example, if we let $\{S_i\}$ be a partition of the data indices, each compute node i could store the corresponding subset of the data $D_i = \{y_c\}_{c \in S_i}$, so that the log likelihood $\ell_i(x)$ is a sum over terms, each corresponding to the log density of one data point stored on node i,

$$\ell_i(x) = \sum_{c \in S_i} \log p(y_c \mid x).$$

Covariates and input vectors can be easily incorporated in the above. The target posterior distribution is then,

$$\tilde{p}(x) := p(x \mid \{D_i\}_{i=1}^n) \propto p_0(x) \exp\left(\sum_{i=1}^n \ell_i(x)\right). \tag{4.2}$$

Using neural networks as an example, x corresponds to all learnable weights and biases in a network, $\log p(y_c \mid x)$ gives the probability of the class of data item c given the corresponding input vector, and the Gaussian prior corresponds to weight decay.

The learning task is then to compute the posterior distribution. For example, we may want to estimate the posterior mean or variance of the model parameters x, or we may want to draw samples distributed according to $\tilde{p}(x)$, using these to predict on test data by averaging the predictive densities over the samples as a Monte Carlo estimate of the marginal predictive density. In the rest of the paper we will aim to obtain these efficiently but approximately.

4.3 Variational Inference in an Extended Exponential Family

In general, the likelihood functions are intractable and approximations are necessary. In this paper we will formulate the learning task as variational inference in an extended exponential family. In particular, we will consider a class of variational methods known as *power expectation propagation* (power EP) (Minka, 2004).

4.3.1 Extended Exponential Family

To start with, we may trivially formulate the target posterior distribution as an *extended exponential family distribution* with sufficient statistics $\tilde{s}(x) := [s(x), \ell_1(x), \ldots, \ell_n(x)]$ and natural parameters $\tilde{\theta} := [\theta_0, \mathbf{1}_n]$ where $\mathbf{1}_n$ is a vector of 1's of length n:

$$\tilde{p}(x) = \exp\left(\tilde{\theta}^\top \tilde{s}(x) - \tilde{A}(\tilde{\theta})\right).$$

The extended log partition function $\tilde{A}(\tilde{\theta})$ is (up to a constant) the log marginal probability of the data,

$$\begin{aligned}\tilde{A}(\tilde{\theta}) = \log \int \exp\left(\tilde{\theta}^\top \tilde{s}(x)\right) dx &= \log \int \exp\left(\theta_0^\top s(x) + \sum_{i=1}^{n} \ell_i(x)\right) dx \\ &= \log \mathbb{E}_{\theta_0}\left[\exp\left(\sum_{i=1}^{n} \ell_i(x)\right)\right] + A(\theta_0) \\ &= \log p(\{D_i\}_{i=1}^n) + A(\theta_0).\end{aligned}$$

Denoting the convex conjugate by $\tilde{A}^*(\tilde{\mu})$ and the extended mean domain by $\tilde{\mathcal{M}} \subset \mathbb{R}^{d+n}$, the problem of posterior computation can be expressed as the following concave variational maximization problem:

$$\max_{\tilde{\mu} \in \tilde{\mathcal{M}}} \tilde{\theta}^\top \tilde{\mu} - \tilde{A}^*(\tilde{\mu}). \tag{4.3}$$

For example, if the prior exponential family is a diagonal covariance Gaussian, then the optimal mean parameter is $\tilde{\mu}^* := [\mu^*, \nu_1^*, \ldots \nu_n^*]$, where $\mu^* := \mathbb{E}_{\tilde{\theta}}[s(x)] \in \mathbb{R}^d$ corresponds to the posterior means and variances of the model parameters x, while $\nu_i^* := \mathbb{E}_{\tilde{\theta}}[\ell_i(x)]$ is the posterior expectation of the ith log likelihood $\ell_i(x)$. Hence the extended mean parameters capture the important aspects of the posterior distribution.

4.3.2 Power Expectation Propagation

In this section we derive another class of variational approximations corresponding to a generalization of expectation propagation (EP) (Minka, 2001a) called power EP (Minka, 2004). The derivation is a straightforward generalisation of the variational formulation of Wainwright and Jordan (2008) from EP to power EP.

For each worker node i let $\beta_i > 0$ be a given positive real number. Typically, we take $\beta_i = 1$ which corresponds to standard EP, while $\beta_i \to \infty$ corresponds to variational Bayes (Wiegerinck and Heskes, 2003, Minka, 2004). In the formulation of Wainwright and Jordan (2008), EP involves two approximations; both associated with simpler exponential families, which we refer to as *locally extended exponential families* (or sometimes *local exponential families*). For each i, let the ith locally extended exponential family be associated with the sufficient statistics function $s_i(x) := [s(x), \ell_i(x)]$. Let Θ_i, $\mathcal{M}_i$, A_i, A_i^* be the associated (local) natural domain, mean domain, log partition function and negative entropy respectively. A distribution in this locally extended exponential family with natural parameter $[\theta_i, \eta_i] \in \Theta_i$ has the form

$$p_{\theta_i,\eta_i}(x) = \exp\left(\theta_i^\top s(x) + \eta_i \ell_i(x) - A_i(\theta, \eta_i)\right), \tag{4.4}$$

which is a distribution obtained by tilting a tractable distribution with density proportional to $\exp(\theta_i^\top s(x))$ by a single intractable likelihood $\exp(\ell_i(x))$ raised to the power of η_i. This family can be thought of as treating the ith likelihood term exactly, while approximating all other likelihood terms by projecting them onto the tractable base exponential family, the hope being that this family is still tractable while being closer to the true posterior distribution $\tilde{p}(x)$.

For the first approximation, the extended negative entropy $\tilde{A}^*$ is approximated using a tree-like approximation constructed using only the locally extended negative entropies,

$$\tilde{A}^*([\mu, \nu_1, \ldots, \nu_n]) \approx A^*(\mu) + \sum_{i=1}^{n} \beta_i (A_i^*(\mu, \nu_i) - A^*(\mu)).$$

This approximation is related to the Bethe entropy of loopy belief propagation (Yedidia et al., 2001) and fractional belief propagation (Wiegerinck and Heskes, 2003). Secondly, the extended mean domain $\tilde{\mathcal{M}}$ is approximated by a local mean domain,

$$\mathcal{L} := \{[\mu, \nu_1, \ldots, \nu_n] : [\mu, \nu_i] \in \mathcal{M}_i \text{ for all } i = 1, \ldots, n\}. \tag{4.5}$$

The local mean domain is an outer bound, $\mathcal{L} \supset \tilde{\mathcal{M}}$. Intuitively, the constraints described by $\tilde{\mathcal{M}}$ are replaced by the weaker constraints described by the local mean

domains. We assume that working with the local exponential families in these ways will be more tractable than working with the full extended exponential family.

Let $\mu_0 \in \mathcal{M}$ be a mean parameter in the base exponential family. For the ith local exponential family, we denote a mean parameter by $[\mu_i, \nu_i] \in \mathcal{M}_i$, and require the marginalization constraint $\mu_i = \mu_0$. The power EP variational problem, which is not in general concave, is,

$$\begin{aligned} \max_{\mu_0, [\mu_i,\nu_i]_{i=1}^n} \quad & \theta_0^\top \mu_0 + \sum_{i=1}^n 1 \cdot \nu_i - A^*(\mu_0) - \sum_{i=1}^n \beta_i (A_i^*(\mu_i, \nu_i) - A^*(\mu_i)) \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i, \nu_i] \in \mathcal{M}_i \quad \text{for } i = 1, \ldots, n \\ & \mu_0 = \mu_i \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{4.6}$$

Note in particular that the entropy and mean domain in the original variational problem (4.3) have been replaced by their respective approximations.

4.3.3 Deriving EP and Power EP Updates

In this section, for completeness' sake, we derive the updates for EP and power EP as fixed-point equations that solve the variational problem. Readers familiar with this derivation can skip ahead to section 4.3.4.

First we introduce Lagrange multipliers λ_i for the equality constraints $\mu_0 = \mu_i$, so that the above is equivalent to,

$$\begin{aligned} \max_{\mu_0, [\mu_i,\nu_i]_{i=1}^n} \min_{[\lambda_i]_{i=1}^n} \quad & \underbrace{\theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^n \left(\nu_i - \lambda_i^\top (\mu_i - \mu_0) - \beta_i (A_i^*(\mu_i, \nu_i) - A^*(\mu_i)) \right)}_{=: L(\mu_0, [\mu_i, \nu_i, \lambda_i]_{i=1}^n)} \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i, \nu_i] \in \mathcal{M}_i \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{4.7}$$

where the domain of λ_i is $\mathbb{R}^d$. Let the Lagrangian above be denoted by $L(\mu_0, [\mu_i, \nu_i, \lambda_i]_{i=1}^n)$. The Karush-Kuhn-Tucker (KKT) conditions of the above variational problem have to be satisfied at an optimum, and simply involve setting the derivatives with respect

to $\mu_0, \mu_i, \nu_i, \lambda_i$ to zero:

$$\frac{dL}{d\lambda_i} = 0 \; : \qquad \mu_i = \mu_0 \tag{4.8a}$$

$$\frac{dL}{d\mu_0} = 0 \; : \qquad \theta_0 - \nabla A^*(\mu_0) + \sum_{j=1}^{n} \lambda_j = 0$$

$$\theta_0 + \sum_{j=1}^{n} \lambda_j = \nabla A^*(\mu_0) \tag{4.8b}$$

$$\frac{dL}{d\mu_i} = 0 \; : \qquad -\lambda_i - \beta_i \nabla_{\mu_i} A_i^*(\mu_i, \nu_i) + \beta_i \nabla A^*(\mu_i) = 0$$

$$\theta_0 + \sum_{j=1}^{n} \lambda_j - \beta_i^{-1}\lambda_i = \nabla_{\mu_i} A_i^*(\mu_i, \nu_i) \tag{4.8c}$$

$$\frac{dL}{d\nu_i} = 0 \; : \qquad \beta_i^{-1} = \nabla_{\nu_i} A_i^*(\mu_i, \nu_i) \tag{4.8d}$$

Equation (4.8b) shows that an optimal μ_0 has to be the mean parameter corresponding to a (base) exponential family distribution with natural parameter $\theta_0 + \sum_{j=1}^{n} \lambda_j$. Specifically, the posterior distribution can be approximated as,

$$\tilde{p}(x) \propto p_0(x) \exp\left(\sum_{j=1}^{n} \ell_j(x)\right) \approx \exp\left(\left(\theta_0 + \sum_{j=1}^{n} \lambda_j\right)^\top s(x)\right). \tag{4.9}$$

In other words, we can interpret λ_j as the natural parameter of an exponential family approximation to the likelihood factor $\exp(\ell_j(x))$.

Further, from (4.8c) and (4.8d) above, we see that the optimal $[\mu_i, \nu_i]$ is the mean parameter associated with the local posterior distribution,

$$p_i(x) \propto \exp\left(\left(\theta_0 + \sum_{j=1}^{n} \lambda_j - \beta_i^{-1}\lambda_i\right)^\top s(x) + \beta_i^{-1}\ell_i(x)\right). \tag{4.10}$$

For standard EP, where $\beta_i = 1$, we get,

$$p_i(x) \propto \exp\left(\left(\theta_0 + \sum_{j \neq i} \lambda_j\right)^\top s(x) + \ell_i(x)\right).$$

The above local posterior distribution is known as the tilted distribution in EP, with the term $(\theta_0 + \sum_{j=1}^{n} \lambda_j - \beta_i^{-1}\lambda_i)^\top s(x)$ corresponding to the cavity distribution with (the β_i^{-1}th power of) the exponential family approximation to the ith likelihood removed, and replaced by (the β_i^{-1}th power of) the likelihood factor itself. Each step of EP

involves first computing $[\mu_i, \nu_i]$ as the mean parameter of the tilted distribution, then updating λ_i, using (4.8a) and (4.8b):

$$\lambda_i^{\text{new}} = \nabla A^*(\mu_i) - \theta_0 - \sum_{j \neq i} \lambda_j, \tag{4.11}$$

where we recall that $\nabla A^*(\mu_i)$ computes the natural parameter corresponding to the mean parameter μ_i in the base exponential family. The EP update ensures that the expectation of the sufficient statistics function under the tilted distribution $p_i(x)$ and under its exponential family approximation (with natural parameters $\theta_0 + \sum_{j=1}^n \lambda_j - \beta_i^{-1}\lambda_i + \beta_i^{-1}\lambda_i^{\text{new}}$) match. At convergence, this ensures that the expectations under all the tilted distributions and under the approximated posterior distribution (4.9) match.

Note that the (power) EP updates are derived as fixed point equations and have no guarantees of convergence. In practice, damped updates can be used to avoid oscillations without affecting the fixed points:

$$\lambda_i^{\text{new}} = \alpha\lambda_i + (1-\alpha)\left(\nabla A^*(\mu_i) - \theta_0 - \sum_{j \neq i} \lambda_j\right). \tag{4.12}$$

where $\alpha \in [0, 1)$ is a damping factor. Another potential issue is that λ_i^{new} as computed may not lie in the natural domain Θ, in which case higher damping factors ensuring that it does can be used, or the update can be discarded.

4.3.4 Computing Mean Parameters

Assuming that the base exponential family is tractable, the above steps of EP involve additions, subtractions, and conversions between mean and natural parameters so are easy to compute. The only difficulty is in the computation of the mean parameters (expectations of the sufficient statistics function) of the tilted distributions, which involve the log likelihoods $\ell_i(x)$. In typical applications of EP to graphical models, these are either obtained analytically or using numerical quadrature.

In more complex and general scenarios, both analytic or quadrature-based methods are ruled out. A successful and common class of methods for calculating expectations under otherwise intractable distributions is Monte Carlo. A number of papers have proposed such an approach, including Barthelmé and Chopin (2011), Heess et al. (2013), Gelman et al. (2014) using importance sampling and Xu et al. (2014), Gelman et al. (2014) using Markov chain Monte Carlo (MCMC). In addition,

Heess et al. (2013), Eslami et al. (2014), Jitkrittum et al. (2015) use learning techniques to speed-up the process by directly predicting the natural parameters given properties of the tilted distribution.

We have explored the sampling via moment sharing (SMS) algorithm (Xu et al., 2014) in the context of distributed Bayesian learning of deep neural networks. Unfortunately, the SMS algorithm did not work even for relatively small neural networks and data sets, partly because of the high dimensionality and partly because the Monte Carlo estimation is inherently stochastic, both of which we found affected the convergence of the EP fixed point equations.

4.4 Stochastic Natural Gradient Expectation Propagation

In this section we will derive a novel stochastic approximation based alternative to EP and power EP, which optimises the same variational objective but is significantly more tolerant of Monte Carlo noise. Our algorithm takes a double loop form and can be shown to be convergent under certain conditions. Our algorithm is derived using a modified variational objective with additional auxiliary variables but with the same optima as the original problem, and solving the dual problem using a stochastic approximation algorithm (Robbins and Monro, 1951). In the following we will use "EP" to refer to both EP and power EP for simplicity.

4.4.1 Auxiliary Variational Problem

For each i, we introduce an auxiliary natural parameter vector $\theta'_i \in \Theta$, and introduce a term $-\sum_{i=1}^{n} \mathrm{KL}(p_{\mu_i} \| p_{\theta'_i})$ into the variational objective (4.7), where p_{μ_i} and $p_{\theta'_i}$ are the base exponential family distribution given by mean parameter μ_i and natural parameter θ'_i, respectively. This results in a lower bound on the original objective and is reminiscent of the EM algorithm (Dempster et al., 1977, Neal and Hinton, 1999) and of typical variational Bayes approximations (Beal, 2003, Wainwright and Jordan, 2008). Adding these new terms to the previous formulation in (4.7), we have

the resulting variational problem

$$\begin{aligned} \max_{\mu_0,[\mu_i,\nu_i,\theta_i']_{i=1}^n} \quad & \theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^n \left(\nu_i - \beta_i \left(A_i^*(\mu_i,\nu_i) - \mu_i^\top \theta_i' + A(\theta_i')\right)\right) \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i,\nu_i] \in \mathcal{M}_i \quad \text{for } i = 1,\ldots,n \\ & \theta_i' \in \Theta \quad \text{for } i = 1,\ldots,n \\ & \mu_0 = \mu_i \quad \text{for } i = 1,\ldots,n \end{aligned} \tag{4.13}$$

Maximizing over θ_i' while keeping the other variables fixed will simply set $\theta_i' = \nabla A^*(\mu_i)$, so that the KL terms vanish and resulting in the original problem (4.6). On the other hand, the constrained maximization over the original parameters $\mu_0, [\mu_i,\nu_i]_{i=1}^n$ requires introducing Lagrange multipliers,

$$\begin{aligned} \max_{[\theta_i']_{i=1}^n} \max_{\mu_0,[\mu_i,\nu_i]_{i=1}^n} \min_{[\lambda_i]_{i=1}^n} & \theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^n \big(\nu_i - \lambda_i^\top(\mu_i - \mu_0) - \beta_i\big(A_i^*(\mu_i,\nu_i) - \mu_i^\top\theta_i' + A(\theta_i')\big)\big) \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i,\nu_i] \in \mathcal{M}_i \quad \text{for } i = 1,\ldots,n \\ & \theta_i' \in \Theta \quad \text{for } i = 1,\ldots,n \end{aligned} \tag{4.14}$$

We can solve this inner constrained optimization by transforming to the convex dual. Noticing that the Lagrangian is concave in $\mu_0, [\mu_i,\nu_i]_{i=1}^n$ and that Slater's condition holds, the duality gap is zero and we have,

$$\begin{aligned} \max_{[\theta_i']_{i=1}^n} \min_{[\lambda_i]_{i=1}^n} \max_{\mu_0,[\mu_i,\nu_i]_{i=1}^n} & \theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^n \big(\nu_i - \lambda_i^\top(\mu_i - \mu_0) - \beta_i\big(A_i^*(\mu_i,\nu_i) - \mu_i^\top\theta_i' + A(\theta_i')\big)\big) \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i,\nu_i] \in \mathcal{M}_i \quad \text{for } i = 1,\ldots,n \\ & \theta_i' \in \Theta \quad \text{for } i = 1,\ldots,n \end{aligned} \tag{4.15}$$

Now maximizing over $\mu_0, [\mu_i,\nu_i]_{i=1}^n$, we have the equivalent dual problem, whose objective function we will denote with $L([\theta_j', \lambda_j]_{j=1}^n)$,

$$\begin{aligned} \max_{[\theta_i']_{i=1}^n} \min_{[\lambda_i]_{i=1}^n} & \underbrace{A\left(\theta_0 + \sum_{i=1}^n \lambda_i\right) + \sum_{i=1}^n \beta_i \left(A_i\left(\theta_i' - \beta_i^{-1}\lambda_i, \beta_i^{-1}\right) - A(\theta_i')\right)}_{=:L([\theta_j',\lambda_j]_{j=1}^n)} \\ \text{subject to} \quad & \theta_i' \in \Theta \quad \text{for } i = 1,\ldots,n \end{aligned} \tag{4.16}$$

In summary, our algorithm is coordinate maximization of (4.13), where we alternate between maximizing $[\theta_i']_{i=1}^n$ given $\mu_0, [\mu_i,\nu_i]_{i=1}^n$, and maximizing $\mu_0, [\mu_i,\nu_i]_{i=1}^n$

given $[\theta'_i]_{i=1}^n$. To solve the second maximization, we transform to the dual and minimize (4.16) with respect to the Lagrange multipliers $[\lambda_i]_{i=1}^n$. This requires an inner iteration loop involving stochastic natural gradient descent, which is described next. The structure of our derived algorithm is the same as the double loop EP algorithm of (Heskes and Zoeter, 2002). Our derivation is interesting and useful in that a single global variational problem is described, rather than a sequence of variational problems each obtained by minorizing around the current parameters and then maximizing, as in the minorization-maximization (MM) algorithm (Hunter and Lange, 2004) and the CCCP algorithm (Yuille, 2002). Our algorithm is also different in the choice of stochastic natural gradient descent inner loop.

4.4.2 Stochastic Natural Gradient Descent

The gradient of the dual objective $L([\theta'_j, \lambda_j]_{j=1}^n)$ is,

$$\frac{dL}{d\lambda_i} = \nabla A\left(\theta_0 + \sum_{j=1}^{n} \lambda_j\right) - \nabla_{\theta_i} A_i\left(\theta'_i - \beta_i^{-1}\lambda_i, \beta_i^{-1}\right) \tag{4.17}$$

where we used the notation $\nabla_{\theta_i} A_i$ for the partial derivative of $A_i(\cdot,\cdot)$ with respect to its first (d-dimensional) argument. While this gradient is the direction of steepest descent in a Euclidean geometry it does not take into account the geometry of the base exponential family. In other words, the gradient is not covariant[2] (MacKay, 1999) and the corresponding gradient descent algorithm is not expected to perform well.

A better approach is to use natural gradient descent (Amari et al., 1996, Amari and Nagaoka, 2001) or, equivalently, mirror descent (Beck and Teboulle, 2003, Raskutti and Mukherjee, 2015) instead. These methods treat the parameter space as a Riemannian manifold where the metric tensor is given by the Fisher information. The direction of steepest descent in this geometry is given by the gradient preconditioned by the inverse metric. Natural gradient descent has appealing theoretical properties as it incorporates second order information to be more covariant. As we will see below we can take advantage of natural gradient descent at no additional computational cost by reparametrising. As noted in the previous section, the Lagrange multiplier λ_i

[2] A covariant gradient descent algorithm roughly means that the size of the gradient in a direction positively covaries with the length scale in that direction hence with the desired step size. An example of a non-covariant gradient descent algorithm is steepest descent with standard Euclidean geometry, say for a 2D quadratic loss function that is long in the x-axis and short in the y-axis. The gradient will be close to pointing vertically up or down, but the best descent direction points horizontally instead.

can be interpreted as the natural parameters of the base exponential family approximation to the likelihood $\exp(\ell_i(x))$. Reparameterising using the corresponding mean parameter γ_i instead, with $\lambda_i = \nabla A^*(\gamma_i)$, the gradient is,

$$\begin{aligned}\frac{dL}{d\gamma_i} &= \frac{d\lambda_i}{d\gamma_i}\left(\nabla A\left(\theta_0 + \sum_{j=1}^{n}\lambda_j\right) - \nabla_{\theta_i} A_i\left(\theta_i' - \beta_i^{-1}\lambda_i, \beta_i^{-1}\right)\right) \\ &= \nabla^2 A^*(\gamma_i)\left(\nabla A\left(\theta_0 + \sum_{j=1}^{n}\lambda_j\right) - \nabla_{\theta_i} A_i\left(\theta_i' - \beta_i^{-1}\lambda_i, \beta_i^{-1}\right)\right) \end{aligned} \tag{4.18}$$

The appropriate metric in the mean parameter space is simply $\nabla^2 A^*(\gamma_i)$, so that its inverse cancels the $\nabla^2 A^*(\gamma_i)$ term (Raskutti and Mukherjee, 2015), and the natural gradient update is simply,

$$\gamma_i^{(t+1)} = \gamma_i^{(t)} + \epsilon_t\left(\nabla_{\theta_i} A_i\left(\theta_i' - \beta_i^{-1}\lambda_i^{(t)}, \beta_i^{-1}\right) - \nabla A\left(\theta_0 + \sum_{j=1}^{n}\lambda_j^{(t)}\right)\right) \tag{4.19}$$

where the corresponding natural parameter is given as a function of the mean parameter, $\lambda_i^{(t)} := \nabla A^*(\gamma_i^{(t)})$, and ϵ_t is the step size at iteration t. These updates can be performed in series or parallel fashion. In our distributed Bayesian learning setting they are performed in an asynchronous distributed fashion (Section 4.5).

Recall that derivatives of the log partition function compute the mean parameters from the natural parameters. Hence the first term of the natural gradient step is the mean parameter of the current local tilted distribution,

$$p_i^{(t)}(x) = \exp\left((\theta_i' - \beta_i^{-1}\lambda_i^{(t)})^\top s(x) + \beta_i^{-1}\ell_i(x) - A_i(\theta_i' - \beta_i^{-1}\lambda_i^{(t)}, \beta_i^{-1})\right), \tag{4.20}$$

while the second term is the mean parameter of the current exponential family approximation to the global posterior. Their difference gives the update for the mean parameter of the likelihood approximation. The gradient is zero when both terms are equal, precisely the condition from which the EP fixed point equation is derived.

In general, the first term cannot be obtained in closed form, and we instead use a Markov chain Monte Carlo (MCMC) estimate, leading to a stochastic natural-gradient descent algorithm. Specifically, let $\mathcal{K}_i(\cdot \mid x; \theta_i' - \beta_i^{-1}\lambda_i^{(t)}, \beta_i)$ be a Markov chain kernel with previous state x whose invariant distribution is the local tilted distribution (4.20). Let $x_i^{(t)}$ be the state of the Markov chain at iteration t. The next state $x_i^{(t+1)}$ is obtained by simulating from the Markov chain, using the current values of the parameters,

$$x_i^{(t+1)} \sim \mathcal{K}_i\left(\cdot \mid x_i^{(t)}; \theta_i' - \beta_i^{-1}\lambda_i^{(t)}, \beta_i\right). \tag{4.21}$$

In summary, the stochastic natural gradient update is,

$$\gamma_i^{(t+1)} = \gamma_i^{(t)} + \epsilon_t \left(s\left(x_i^{(t+1)}\right) - \nabla A\left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)}\right)\right) \tag{4.22}$$

We refer to the resulting algorithm as "Stochastic Natural-gradient EP", or SNEP for short.

Technically, the stochastic natural-gradient descent requires unbiased estimates of gradients, and the mean parameter estimates obtained using MCMC updates are only unbiased if the Markov chain equilibrates in between gradient updates. In practice, we find that using MCMC samples works well regardless, an observation consistent with past experiences with stochastic approximation algorithms in machine learning (Neal and Hinton, 1999, Tieleman, 2008).

The stochastic natural gradient descent algorithm above serves as the inner loop of our double loop algorithm, and solves for the constrained maximization of the mean parameters $\mu_0, [\mu_i, \nu_i]_{i=1}^n$ given auxiliary parameters $[\theta_i']_{i=1}^n$. Returning now to the outer loop update, maximizing θ_i' involves simply setting it to be $\nabla A^*(\mu_i)$, the natural parameter corresponding to the mean parameter μ_i of the local tilted distribution (see the discussion after Equation 4.13). Assuming that the inner loop has converged, this and the mean parameters of all other tilted distributions would be equal to μ_0, so that the t'th outer loop update is,

$$(\theta_i')^{(t'+1)} = \nabla A^*(\mu_i) = \nabla A^*\left(\mu_0\right) = \theta_0 + \sum_{j=1}^{n} \lambda_j^{(\infty)} \tag{4.23}$$

where $\lambda_j^{(\infty)}$ is the converged value of the dual parameters in the inner loop. In practice, we simply perform the outer loop update infrequently (and before the inner loop has fully converged). In our experiments we do not see instabilities resulting from this, an observation also consistent with past experiences with double loop algorithms (Teh and Welling, 2002, Yuille, 2002, Heskes and Zoeter, 2002).

In the extreme case where the outer loop update is performed after every inner loop update, we can roll both updates into the following update,

$$\gamma_i^{(t+1)} = \gamma_i^{(t)} + \epsilon_t \left(\nabla_{\theta_i} A_i \left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)} - \beta_i^{-1} \lambda_i^{(t)}, \beta_i^{-1} \right) - \nabla A \left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)} \right) \right) \tag{4.24}$$

It is interesting to contrast the above with a damped EP update, which in our notation is given by,

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + \epsilon_t \left(\nabla A^* \left(\nabla_{\theta_i} A_i \left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)} - \beta_i^{-1} \lambda_i^{(t)}, \beta_i^{-1} \right) \right) - \left(\theta_0 + \sum_{j=1}^{n} \lambda_j^{(t)} \right) \right) \tag{4.25}$$

Note that $\nabla A^*(\cdot)$ converts from mean parameters to natural parameters, so that the first term in parentheses can be read as first computing the moments (mean parameters) of the tilted distribution then converting into the corresponding natural parameter. Hence each term in the damped EP update is obtained by converting the corresponding term in (4.24) from mean to natural parameters, i.e. applying $\nabla A^*(\cdot)$. In other words the update (4.24) can be thought of as a mean parameter space version of a damped EP update. When mean parameters are estimated with Monte Carlo noise, updating using mean parameters rather than natural parameters leads to more stable behaviour; see Section 4.6.1 for empirical results supporting this claim.

4.4.3 Convergence of Stochastic Natural Gradient Expectation Propagation and Related Algorithms

Both SNEP and the double loop EP algorithm of (Heskes and Zoeter, 2002) are part of a family of double loop algorithms solving the same optimization problem given by (4.14):

$$\begin{aligned} \max_{\mu_0, [\mu_i, \nu_i, \theta_i']_{i=1}^n} \quad & \underbrace{\theta_0^\top \mu_0 - A^*(\mu_0) + \sum_{i=1}^{n} \left(\nu_i - \beta_i \left(A_i^*(\mu_i, \nu_i) - \mu_i^\top \theta_i' + A(\theta_i') \right) \right)}_{\mathcal{F}(\mu_0, [\mu_i, \nu_i]_{i=1}^n, [\theta_i']_{i=1}^n)} \\ \text{subject to} \quad & \mu_0 \in \mathcal{M} \\ & [\mu_i, \nu_i] \in \mathcal{M}_i \quad \text{for } i = 1, \ldots, n \\ & \theta_i' \in \Theta \quad \text{for } i = 1, \ldots, n \\ & \mu_0 = \mu_i \quad \text{for } i = 1, \ldots, n, \end{aligned} \tag{4.26}$$

which is obtained by bounding the Power EP objective with auxiliary KL terms, introducing auxiliary parameters θ_i'. Due to the auxiliary KL terms the inner maximization problem is concave in $\mu_0, [\mu_i, \nu_i]_{i=1}^n$ and thus has a unique maximum $\mu_0^*, [\mu_i^*, \nu_i^*]_{i=1}^n$ given auxiliary variables θ_i'. This optimization problem can be solved by a variety of algorithms and SNEP and the algorithm of (Heskes and Zoeter, 2002) are simply particular choices. As discussed above, in the outer loop we can simply set

$$(\theta_i')^{new} = \nabla A^*(\mu_i) = \nabla A^*(\mu_0) = \theta_0 + \sum_{j=1}^{n} \lambda_j^{(\infty)} \tag{4.27}$$

where $\lambda_j^{(\infty)}$ is the converged value of the dual parameters in the inner loop. This is equivalent to setting the additional KL terms to zero and improves the objective. Thus, after running the inner loop to convergence and performing an outer update we have:

$$\begin{aligned} \mathcal{F}(\mu_0, [\mu_i, \nu_i]_{i=1}^n, [\theta_i']_{i=1}^n) &\leq \mathcal{F}(\mu_0^*, [\mu_i^*, \nu_i^*]_{i=1}^n, [\theta_i']_{i=1}^n) \\ &\leq \mathcal{F}(\mu_0^*, [\mu_i^*, \nu_i^*]_{i=1}^n, [(\theta_i')^{new}]_{i=1}^n), \end{aligned} \tag{4.28}$$

that is, the objective is monotonically increases in each outer iteration. Provided that the objective is bounded above, this implies convergence. This convergence guarantee only applies if the inner loop is run to convergence in each outer iteration, which limits its practical usefulness. In practice we only perform a small number of inner loop updates per outer loop update. In addition, this convergence result only holds if updates are performed synchronously, whereas in our experiments we consider a distributed, asynchronous architecture.

4.4.4 Discussion and Related Works

While we developed SNEP in the context of distributed Bayesian learning, it is clear that it is generally applicable, since the distribution (4.2) targeted is simply a product of factors, each of which is approximated by a factor in the base exponential family, precisely the setting of EP. SNEP can therefore be used in place of EP in situations where Monte Carlo moment estimates are used, including graphical models (Heess et al., 2013, Eslami et al., 2014, Jitkrittum et al., 2015, Lienart et al., 2015), hierarchical Bayesian models (Gelman et al., 2014), and approximate Bayesian computation (Barthelmé and Chopin, 2011).

Since Minka (2001a), there have been a substantial number of extensions and alternatives to EP proposed. Stochastic EP (Li et al., 2015) and averaged EP (Dehaene

and Barthelmé, 2015) assume that all factors can be well approximated by the *same* exponential family factor via parameter tying. This saves memory storage and was shown to work well. This is an orthogonal idea to our work, and it is possible to apply it to SNEP as well in the future, although in the distributed learning setting considered in this paper each worker must keep a copy of the parameters anyway, which means it would not reduce memory requirements. Convergent EP (Heskes and Zoeter, 2002) is also a double loop coordinate descent algorithm with convergence guarantees but SNEP differs in that the inner loop is a stochastic natural-gradient descent which is more robust to the use of Monte Carlo estimated moments. We note here that stochastic natural-gradient descent has also been used in Stochastic Variational Inference (Hoffman et al., 2013a).

SNEP can be considered a black-box variational inference algorithm, as the only requirement on the model is the existence of MCMC samplers targeting the tilted distributions. Black-box methods have recently been developed for variational Bayes (Ranganath et al., 2014, Black-box Variational Inference or BBVI) and for power EP (Hernandez-Lobato et al., 2016, Black-box Alpha or BB-α). BB-α and SNEP are both methods for Bayesian learning based on power EP and both operate in the mean parameter space. BB-α uses the parameter tying idea of stochastic and averaged EP to construct a further approximated objective function which is then optimised using a convergent single-loop algorithm. As a result the fixed points of the algorithm are different from power EP. Single loop algorithms are generally preferrable as it is well known that double loop algorithms can be slow if the inner loop is run until convergence. In our experiments, we do not run the inner loop of SNEP until convergence, and have found empirically that even a single iteration of the inner loop per outer loop update does not affect convergence or stability. In fact, on a two-layer feedforward neural network, SNEP converges in less than one tenth of the number of epochs and a fraction of the running times of those quoted by Hernandez-Lobato et al. (2016). In both BBVI and BB-α, naive Monte Carlo estimators are used, with samples drawn from the approximating distribution. The resulting estimators can have high variance, requiring techniques for variance reduction. In contrast, SNEP uses MCMC samplers targeting the tilted distribution. It is generally accepted that, in high-dimensional settings, MCMC samplers often have lower variance than naive Monte Carlo and as a result work better, with the tradeoff being that MCMC samplers need to equilibrate and produce correlated samples.

Our application of SNEP to distributed Bayesian learning applies one exponential family approximation per subset of data on each worker node. This contrasts with

typical applications of EP and variational inference in general, which applies one approximation per data item. This is made possible due to the black-box flexibility of our approach, since the likelihood associated with a data subset is more complex than for a single data item. In cases where the subset is itself quite large, and the Bernstein-von Mises theorem holds, the likelihood will be close to a Gaussian, so that if we use a (full-covariance) Gaussian as the base exponential family, the approximation will introduce negligible biases. For a recent study of EP in the large data limit, see (Dehaene and Barthelmé, 2015). We can think of our approach as a hybrid which interpolates between a pure variational approach (when $n = N$) and a pure MCMC approach (when $n = 1$), with smaller n corresponding to less bias introduced by approximations but higher variance/computational costs.

In our experiments, we apply SNEP to deep neural networks. Deep learning has been extremely successful in a large number of different domains. However, one drawback of neural networks trained by stochastic optimisation techniques is that they do not provide uncertainty estimates. Both black-box variational methods and stochastic gradient MCMC methods can be applied to neural networks yielding uncertainty estimates. Probabilistic backpropagation (Hernandez-Lobato and Adams, 2015, PBP) relies on assumed density filtering (ADF), a sequential form of EP. Other approaches are based on mean field variational inference, which optimises a lower bound of the marginal likelihood. Graves (2011) proposed an algorithm using a biased Monte Carlo estimate of this lower bound to optimise it. More recently, this approach has been extended by Blundell et al. (2015, Bayes by Backprop) who obtain unbiased Monte Carlo estimates of the same objective using the reparametrization trick (Kingma and Welling, 2013).

4.5 The Posterior Server

Our development of SNEP is motivated by the problem of distributed Bayesian learning outlined in Section 4.2.2, where each log likelihood term $\ell_i(x)$ corresponds to the log probability/density of the data subset on worker node i. Using SNEP, each worker node iteratively learns an exponential family approximation of $\ell_i(x)$, with a master node coordinating the learning across workers. We call the master node the *posterior server*, as it maintains and serves the exponential family approximation of the posterior distribution, obtained by combining the prior with the likelihood approximations at the workers.

In more detail, the posterior server maintains the natural parameter $\theta_{\text{posterior}} := \theta_0 + \sum_{j=1}^{n} \lambda_j$ of the global posterior approximation, while each worker node i maintains the mean and natural parameters γ_i, λ_i of the likelihood approximation and the state of the MCMC sampler x_i. It also maintains the auxiliary parameter θ_i' used in the outer loop. Learning at the worker proceeds by alternating between the MCMC (4.21) and inner loop updates (4.22), with periodic outer loop updates (4.23). These updates require access to the data subset at the node, as well as the cavity distribution, with natural parameters $\theta_{-i} := \theta_0 + \sum_{j \neq i} \lambda_j$, which is obtained by getting $\theta_{\text{posterior}}$ from the posterior server and subtracting the local λ_i.

Communication between the worker node and the posterior server involves the worker first sending the posterior server the difference $\Delta_i := \lambda_i^{\text{new}} - \lambda_i^{\text{old}}$ between the current natural parameters λ_i^{new} and the one during the previous communication with the server, λ_i^{old}. The posterior server updates its global posterior approximation via $\theta_{\text{posterior}}^{\text{new}} = \theta_{\text{posterior}}^{\text{old}} + \Delta_i$, and sends the new value $\theta_{\text{posterior}}^{\text{new}}$ back to the worker. The worker in turn uses this to update the cavity, $\theta_{-i}^{\text{new}} = \theta_{\text{posterior}}^{\text{new}} - \lambda_i^{\text{new}}$.

The pseudocode for the overall algorithm is given in Algorithm 8. Note that all communications are performed asynchronously and in a non-blocking fashion. In particular, Steps 11-17 are performed in a separate coroutine from the main loop (Steps 7-18), and Step 15 can happen several iterations of the main loop after Step 14. This is so that compute nodes can spend most of their time learning (Steps 8-10) and do not have to wait for network communications to complete. However, this means that the previous copy of the natural parameter λ_i used when the last message was sent to the master (Step 13) needs to be stored, in addition to the most recent one. We also note that faster compute nodes need not wait for slower ones since they each learn their own separate likelihood approximation parameters. It would be interesting for future research to explore adaptive methods to allow faster compute nodes to increase the data subsets that they learn from, and slower ones to decrease, to balance the learning progress across compute nodes more evenly.

4.5.1 Discussion and Related Works

Our naming of the posterior server contrasts with that of the parameter server (Ahmed et al., 2012) which is typically used for maximum likelihood (or minimum empirical risk) estimation of model parameters. Note however that our algorithmic contribution is effectively orthogonal to (Ahmed et al., 2012), who proposed a generic and robust computational architecture for distributed machine learning. We believe it is possible to implement our algorithm using the parameter server software framework.

Algorithm 8 Posterior Server: Distributed Bayesian Learning via SNEP

for each compute node $i = 1, \ldots, n$ **asynchronously do**
  let $\gamma_i^{(1)}$ be the initial mean parameter of local likelihood approximation.
  let $\lambda_i^{\text{old}} := \lambda_i^{(1)} := \nabla A^*(\gamma_i^{(1)})$ be the initial natural parameter of local likelihood approximation.
  let $\theta_{-i} := \theta_0 + \sum_{j \neq i} \lambda_j^{(1)}$ be the initial natural parameter of cavity distribution
  let $\theta_i' := \theta_{-i} + \lambda_i^{(1)}$ be the initial auxiliary parameter.
  let $x_i^{(1)} \sim p_{\theta_{-i} + \lambda_i^{(1)}}$ be the initial state of MCMC sampler.
  for $t = 1, 2, \ldots$ until convergence **do**
    update local state via MCMC:

$$x_i^{(t+1)} \sim \mathcal{K}_i \left(\cdot \mid x_i^{(t)}; \theta_i' - \beta_i^{-1} \lambda_i^{(t)}, \beta_i^{-1} \right)$$

    update local likelihood approximation:

$$\gamma_i^{(t+1)} := \gamma_i^{(t)} + \epsilon_t \left(s(x_i^{(t+1)}) - \nabla A \left(\theta_{-i} + \lambda_i^{(t)} \right) \right)$$
$$\lambda_i^{(t+1)} := \nabla A^*(\gamma_i^{(t+1)})$$

    every N_{outer} iterations **do**: update auxiliary parameter:

$$\theta_i' := \theta_{-i} + \lambda_i^{(t)}$$

    every N_{sync} iterations **asynchronously do:** communicate with posterior server:
      let $\Delta_i := \lambda_i^{(t)} - \lambda_i^{\text{old}}$.
      update $\lambda_i^{\text{old}} := \lambda_i^{(t)}$.
      send Δ_i to posterior server.
      receive $\theta_{\text{posterior}}$ from posterior server.
      update $\theta_{-i} := \theta_{\text{posterior}} - \lambda_i^{\text{old}}$.
  end for
end for
for the posterior server **do**
  let $\theta_{\text{posterior}} := \theta_0 + \sum_{j=1}^{n} \lambda_i^{(1)}$ be the initial natural parameter of the posterior approximation.
  maintain a queue of messages from workers.
  for each message Δ_i received from some worker i **do**
    update $\theta_{\text{posterior}} := \theta_{\text{posterior}} + \Delta_i$.
    send $\theta_{\text{posterior}}$ to worker i.
  end for
end for

One of the difficulties of the parameter server architecture is that learning happens at the level of parameters, with a single set of parameters being maintained across the cluster. Since the data subsets on workers are disjoint, the learning on each worker tends to make the local copy of the parameters diverge from those on the parameter server and on other workers. As a result, frequent synchronisation with the parameter server is necessary to prevent stale parameters and gradients. As an example, in the DistBelief method (Dean et al., 2012), experiments were conducted where the communication with the master was performed after every iteration, which can significantly slow down the learning process. On the other hand, one of the interesting aspects of the posterior server is that it lifts learning from the level of parameters to the level of distributions over parameters. As a result each worker can maintain a distinct parameter set in the MCMC sampler and a distinct likelihood approximation (since the likelihoods on different workers are indeed different as they have different data subsets) without requiring frequent communication with the posterior server. The only role of communication here is for the cavity distributions, which can be thought of as a way for the system to focus the learning happening on the workers on the relevant regions of the parameter space (Xu et al., 2014). Empirically, the precise parameterisation of the cavity distribution is not very important. For an extreme example, suppose both the prior and likelihood terms are close to being Gaussians and the tractable family is also Gaussian. Then the likelihood approximation will not in fact depend on the cavity distribution at all, and will converge to the true likelihood on each worker independently. See also Zhang et al. (2015a) for elastic-averaging SGD, a similar idea of allowing each worker a separate parameter vector, using an ADMM-like methodology.

4.6 Experiments

In this section we present our experiments on SNEP and the posterior server. Our implementation, which is available at `http://bigbayes.github.com/PosteriorServer`, is in the Julia[3] technical computing language. In our setup, the master and workers are run on separate system processes (that is, in separate memory spaces) on a many-core computing cluster, making use of the high-level parallel programming abstractions provided by Julia. Other architectures are possible; for example, multiple threads sharing an address space, multiple GPUs, or nodes communicating over a network using MPI. For the MCMC sampler on workers, we chose an adaptive version

[3] `http://julialang.org`.

of stochastic gradient Langevin dynamics (SGLD) (Welling and Teh, 2011) related to Adam (Kingma and Ba, 2015), which is more computationally scalable to larger neural network models and data sets than standard MCMC (see Appendix 4.8.2 for details). Our neural network models are implemented in the Mocha[4] deep learning Julia package.

4.6.1 Comparison to SMS on Bayesian Logistic Regression

In this section we compare SNEP against *Sampling via Moment Sharing* (SMS) (Xu et al., 2014), a related algorithm for distributed Bayesian learning whereby each worker has a separate MCMC sampler and coordination across workers is achieved using plain EP. SMS was originally proposed to scale up MCMC methods and as such it assumes that MCMC chains can be run for many iterations to convergence in between communications with the master. SMS uses the MCMC iterates to estimate the moments required for EP updates and so, as discussed in the introduction, requires a large number of iterations to produce the low Monte Carlo noise for EP updates to work properly.

We illustrate below the differing dynamics of SMS and SNEP when applied to a Bayesian logistic regression model with simulated data. We took a similar setup as in the SMS paper (Xu et al., 2014), generating a dataset $\mathcal{D} = \{(z_c, y_c)\}_{c=1}^{N}$ where covariates $z_c \in \mathbb{R}^d$ and response $y_c \in \{0, 1\}$. The conditional distribution of y_c given z_c and weights x is

$$p(y_c = 1 \mid z_c, x) = \sigma(z_c^\top x) \tag{4.29}$$

where σ denotes the logistic function. We used a Gaussian prior $p_0(x) = \mathcal{N}(x; 0_d, 10I_d)$ on x and the aim is to construct an approximation to the posterior $p(x \mid \mathcal{D})$. We generated $N = 50000$ data points with $d = 50$ using iid draws for the covariates, $z_c \sim \mathcal{N}(\mu, \Sigma)$ where $\mu \in [0, 1]^d$ and $\Sigma = PP^\top$ with $P \in [-1, 1]^{d \times d}$, with entries drawn uniformly at random for both P and μ. The generating weight vector x^* is drawn from the prior $\mathcal{N}(x; 0_d, 10I_d)$. The labels y_c are then sampled according to the model.

Both algorithms were run with three workers each with one third of the data. SNEP is run with 1 inner loop iteration per synchronisation with the master (for the purpose of comparing against SMS). Varying numbers of MCMC samples per inner loop iteration were used for both algorithms, to investigate the effect of Monte Carlo noise on the performances of the algorithms (low number of samples meaning high

[4] `https://github.com/pluskid/Mocha.jl`.

noise and both lower performance and lower computational cost). The damping for SMS and the learning rate for SNEP were tuned for best performances. As the base exponential family, we used a full-covariance Gaussian. We compared the predictive RMSE $\sqrt{\sum_c |\hat{p}_c - y_c|^2/N}$ obtained with both methods over time where $\hat{p}_c = \sigma(z_c^T \overline{x})$ where $\overline{x}$ is the currently estimated posterior mean. We also compared the relative difference between the estimated posterior mean and that estimated from a long run of the No-U-Turn sampler (Hoffman and Gelman, 2014) as implemented in Stan (Carpenter et al., 2017) with 4 chains and 50000 iterations.

Figure 4.2 illustrates the performances of both algorithms and how they are influenced by the number of MCMC samples used per iteration. It can be observed that SNEP is more robust and better performing than SMS in the presence of noise. In general, we found that the number of samples needed for SMS to perform well grows with the dimensionality of the problem. In models with very high dimensionality and multi-modality (e.g., neural networks), the number of samples needed per step is very large leading to iterations that are computationally impractical, whereas SNEP can afford to use many fewer samples.

4.6.2 Bayesian Neural Networks

In this section we report experimental results applying SNEP and the posterior server to distributed Bayesian learning of neural networks. We have found that the SMS algorithm exhibited significant instabilities and was not suitable for these larger scale problems. Instead our aim here is to explore the behaviour of SNEP when varying various key hyperparameters. We used diagonal covariance Gaussians as the approximating exponential family due to computational cost considerations. While a diagonal Gaussian approximation can be quite poor for the full Bayesian posterior, past works have shown that they can produce good predictive performances (Hernandez-Lobato and Adams, 2015, Blundell et al., 2015, Kirkpatrick et al., 2017). Hence our aim is to evaluate the predictive performance of SNEP, treated as a distributed learning algorithm for neural networks, and comparing it to Adam (Kingma and Ba, 2015), a state-of-the-art stochastic gradient descent (SGD) algorithm with access to the whole data set on a single computer, as well as several state-of-the-art distributed SGD algorithms: asynchronous SGD (A-SGD) (Dean et al., 2012) and elastic averaging SGD (EASGD) (Zhang et al., 2015a).

In our experiments we used stochastic gradient Langevin dynamics with an preconditioning scheme reminiscent of Adam as the MCMC sampler. The preconditioning scheme is the same as the one proposed by (Li et al., 2016a) except for the addition

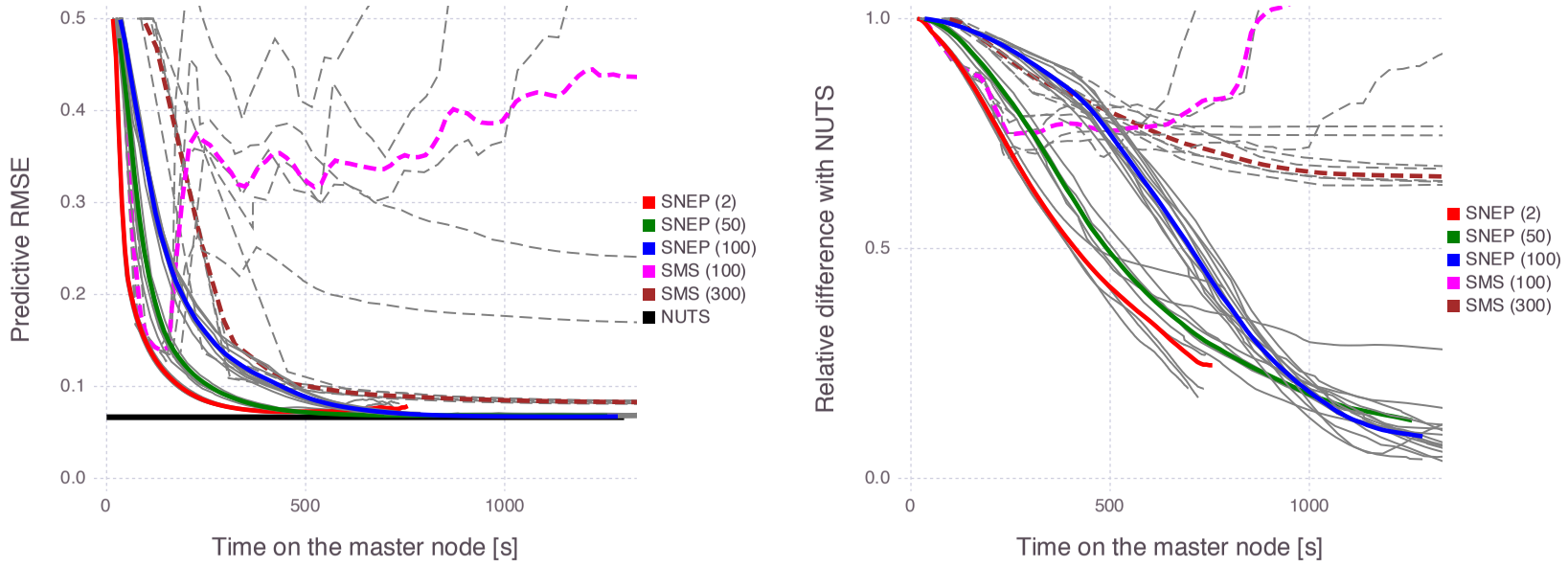


Figure 4.2: (left) Comparison of the predictive RMSE obtained with SMS (dashed-lines) and SNEP (full lines) versus the time on the master node when varying the number of MCMC samples per iteration (numbers reported in the legend). When too few samples are used for the moment estimates, SMS becomes unstable whilst SNEP does not suffer from this. SNEP is also consistently faster and more accurate than SMS. The accuracy obtained with the posterior mean estimated with a long run of Stan/NUTS is also displayed as comparison (horizontal line). (right) Relative difference between the posterior means estimated using Stan/NUTS and using SMS (dashed lines) or SNEP (full lines). Each coloured line corresponds to an average over several runs (represented in light grey).

of a debiasing reminiscent of Adam. We found that sometimes this adaptive scheme lead to large injected noise, especially at the start of training which was detrimental to learning. To mitigate this effect we limited the standard deviation of the injected noise to be at most ϵ in our experiments. For further details see Appendix 4.8.2.

4.6.2.1 MNIST, Two Layer Fully Connected Network

In this section we look at training a fully connected neural network on the MNIST data set[5], which consists of 60000 training images of handwritten digits of size 28×28 and 10000 test images, the task being to classify each image into one of ten classes. The network has two hidden layers with 500 and 300 rectified linear units (ReLUs) respectively and softmax output units.

In the first set of experiments, we varied a number of hyperparameters of the learning regime while keeping the rest kept at default values, to investigate the sensitivity of the learning to these hyperparameters. The default values were chosen by hand in a rough initial round of experimentation as follows: the minibatch size is 100, the learning rate for SNEP was 0.02 and step size for the adaptive SGLD sampler was 0.001, the number of inner loop iterations per communication with master is $N_{sync} = 10$, the number of inner loop iterations per outer loop update is $N_{outer} = 10$, and the parameters were initialised with the popular Xavier initialisation (Glorot and Bengio, 2010). We set a minimum variance of 0.01 for the likelihood approximations, as we found that the estimated variances are otherwise too small due to the diagonal Gaussian approximation. Unless otherwise stated, we used four workers to explore the distributed behaviour. Test curves were produced by evaluating networks based on the approximate posterior means at the master. Each reported curve in the figures is an average over ten repetitions. In our experiments we used one CPU core per worker process. See Figure 4.3 for the results of this set of experiments.

First, we examined the behaviour of SNEP as N_{worker} is varied. We see that two workers perform significantly better than one worker or Adam. However, performance deteriorates as the number of workers is increased further. This is possibly due to smaller number of data items per worker which makes the variational approximation more severe. Another factor we found is that because we imposed a minimum variance in for the likelihood approximations, using more workers gives tighter cavity distributions which can affect the scale of the natural-gradients and slow convergence significantly.

[5] `http://yann.lecun.com/exdb/mnist/`.

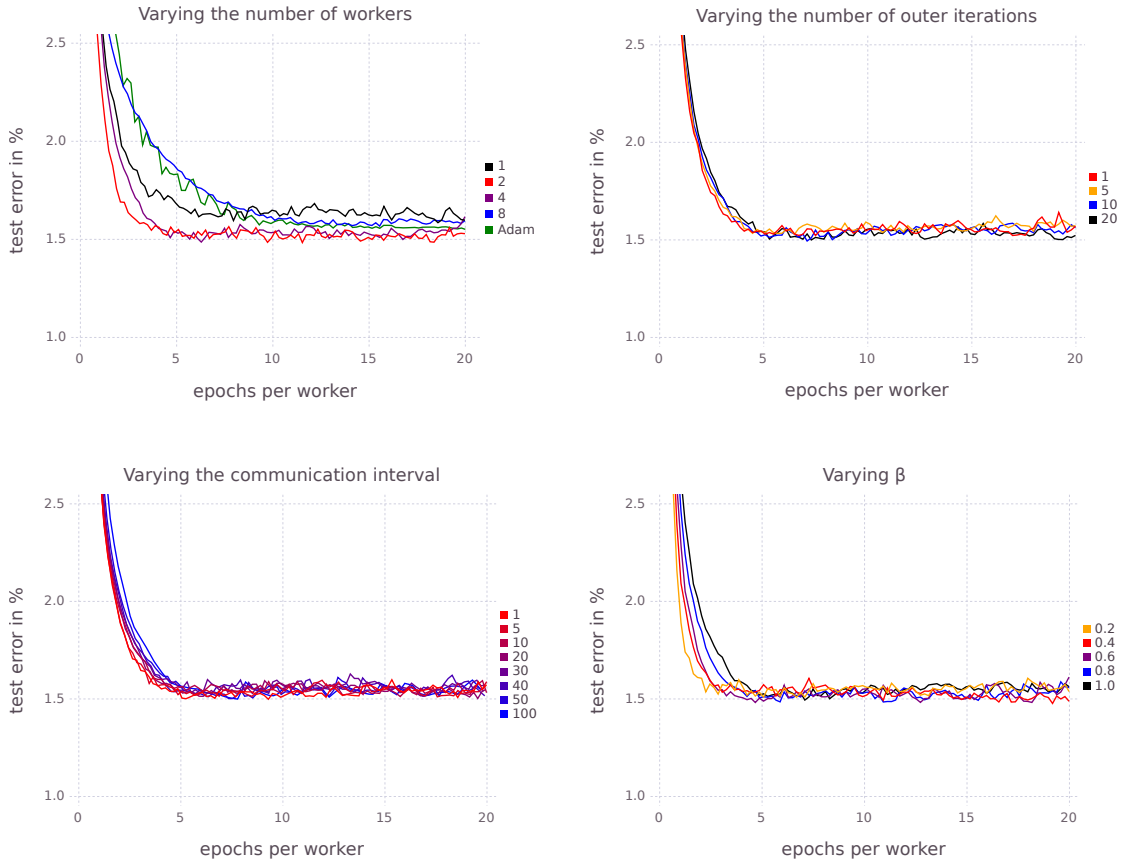


Figure 4.3: Results on a two-layer densely connected model on MNIST varying the parameters of SNEP and comparing against Adam as a baseline. Four workers were used in the distributed methods in the experiments that held the number of workers fixed. This demonstrates that SNEP is insensitive to the full convergence of the outer loop, is robust to large communication intervals, that convergence is faster for a lower β (on this data set), and that SNEP produces competitive results quickly. Detailed analyses of sub-figures (left to right, top to bottom) are reported in the main text.

Second, we investigated the effect of varying N_{sync} and N_{outer}. Figure 4.3 demonstrates that SNEP is insensitive to these hyperparameters over reasonably large range of values. Note in particular that infrequent communications with the posterior server (up to once every 100 iterations in this experiment) did not significantly deteriorate the learning process at all. In fact, the related SMS algorithm (Xu et al., 2014) effectively involves communications with the master once every thousands of iterations.

Third, we investigated the effect of varying the β parameter. Interestingly, lower values of β sped up convergence significantly. The literature on power EP links the β parameter with locally minimising $\alpha-$divergences (where $\beta = 1/\alpha$). Lower values of β correspond to local approximations that are likely to capture more of the full posterior rather than matching one mode (Minka, 2005). Another factor is that given the minimum variance setting, lower values of β increase the variance of the cavity distribution allowing for more local exploration. Generally, In our experiments we found that a heuristic of setting $\beta = 1/N_{workers}$ worked particularly well for MNIST. We call the resulting algorithm power SNEP (p-SNEP). With this parameter setting we see a clear advantage for more workers (see figure 4.4). More workers give better results in terms of the test error and convergence is fastest for eight workers. Note that eight workers reach a test error of 1.5% after about two epochs, which is much faster than our experiments with $\beta = 1.0$.

Finally, we compared SNEP with two distributed SGD algorithms, asynchronous SGD (A-SGD, also known as Downpour) (Dean et al., 2012) and elastic averaging SGD (EASGD) (Zhang et al., 2015a). Our implementations differ slightly from those given in the papers. For A-SGD, both the workers and master performed Adam updates with independently chosen step size constants. The authors of EASGD do not address the case of distributing the data among the workers. Nonetheless, we found this posed no problem to the algorithm. All hyperparameters were tuned to optimise performances. We used the same length of communication intervals $N_{sync} = 10$ and four workers for all algorithms. As shown in figure 4.5, SNEP (with $\beta = 1$) and p-SNEP (with $\beta = 1/N_{workers}$) achieve performance comparable to these state-of-the-art methods, both in terms of speed and final test accuracies. SNEP is comparable to Adam and better than A-SGD. p-SNEP outperforms A-SGD, Adam and SNEP, and is slightly faster initially than EASGD but achieves slightly higher test errors. It is important to note that we are tackling a harder problem with SNEP/p-SNEP, that of Bayesian learning rather than optimisation.

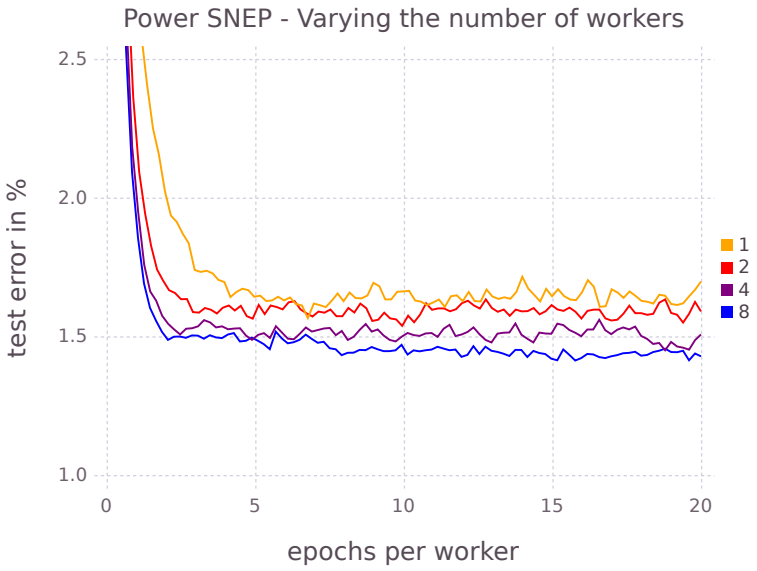


Figure 4.4: Results on a two-layer densely connected model on MNIST for power SNEP ($\beta = 1/N_{workers}$) varying the number of workers. There is a clear improvement in terms of convergence speed and final accuracy for more workers.

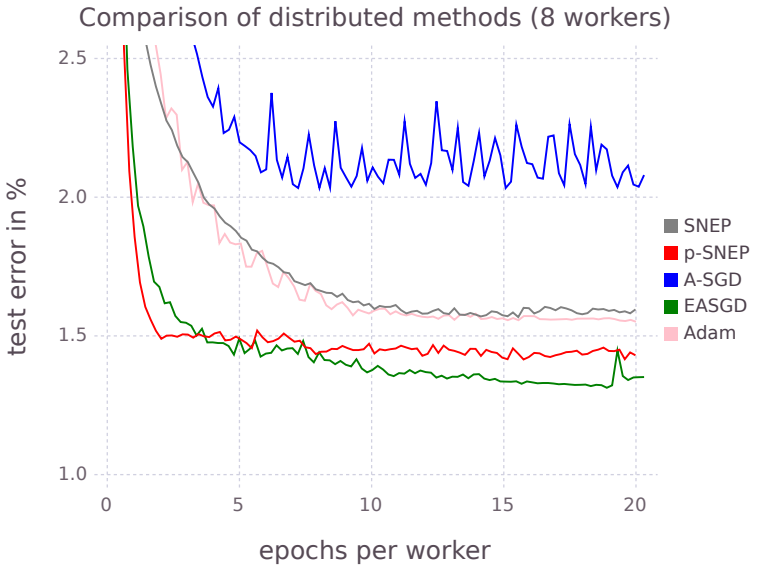


Figure 4.5: Results on a two-layer densely connected model on MNIST comparing SNEP and p-SNEP to asychronous SGD and elastic averaging SGD.

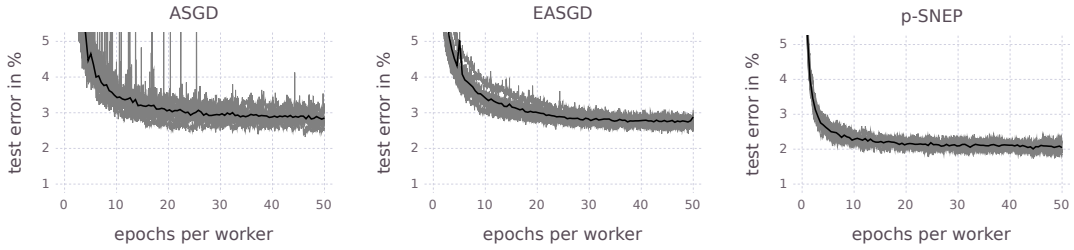
4.6.2.2 MNIST, Very Deep Fully Connected Network

Deeper models pose a much harder learning problem—as the depth increases, the number of saddle points increases exponentially (Dauphin et al., 2014). Moreover, learning often gets stuck not at critical points but in large plateaus of the error surface (Lipton, 2016). It is, therefore, interesting to evaluate (p-)SNEP against standard methods on a deeper fully-connected network, and we chose the architecture from Neelakantan et al. (2016), which has 20 hidden layers of 50 units each.

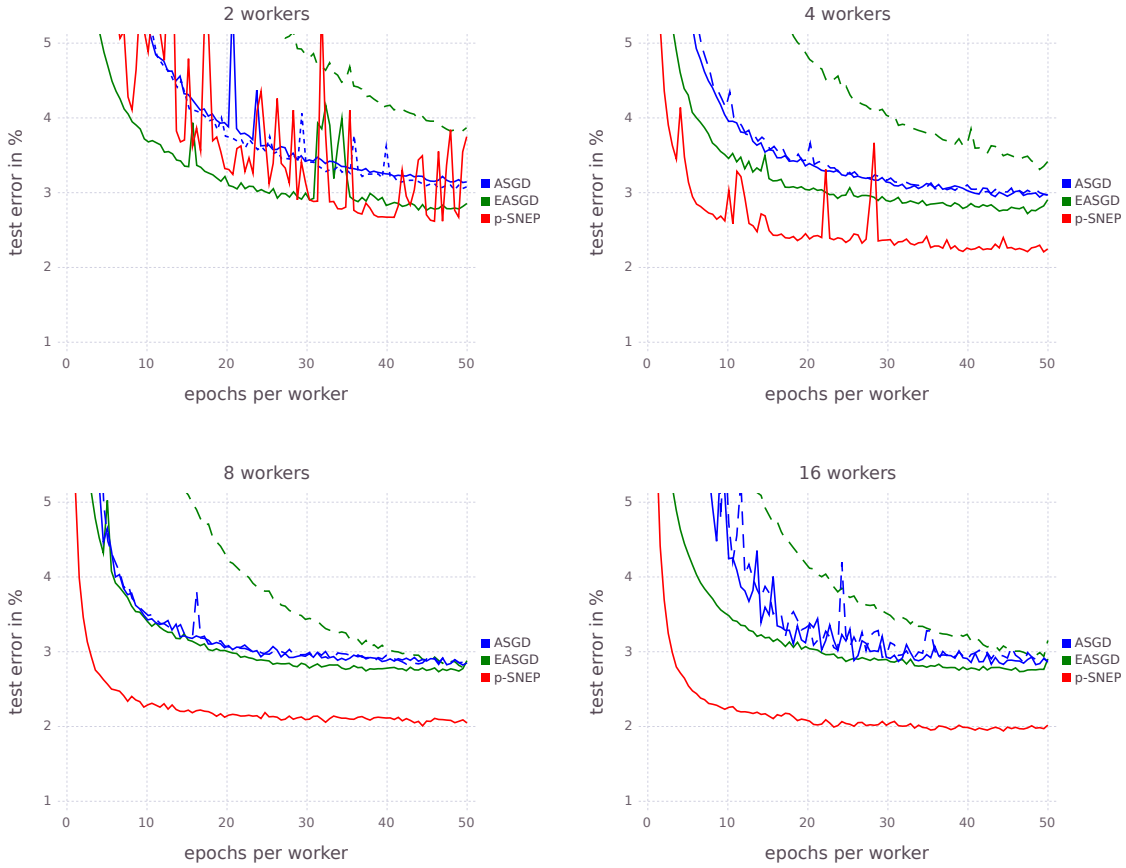
The experimental setup was similar to the previous section. The distributed methods used a communication intervals of $N_{sync} = 10$ and a varying number of workers. For p-SNEP, we optimized the prior variance, scaling of the injected noise, learning rate, step size, and beta hyperparameters by a course grid search over five orders of magnitude. Similarly, for A-SGD we optimized the Adam step size constants, separately on the workers and master, for EASGD the moving rate and Adam step size constants, and for SGD the Adam step size constants and scaling of injected noise, from no noise to three orders of magnitude. All methods were initialized by the Xavier method, and run with and without a non-distributed pre-learning phase of a third of an epoch. We found that pre-learning was essential for EASGD, while it was unnecessary for A-SGD and p-SNEP. In fact, for SNEP, too long a pre-learning phase is actually harmful to the final solution reached, which we suggest is due to the reduced exploration between workers. We only report results for p-SNEP without pre-learning (as per the previous section).

The same Adam step size constant was found to be optimal across all methods, 0.00065. For p-SNEP, the prior variance was set to 0.05, β to 1/8, and the learning rate to 0.04. For EASGD, the moving rate was set to 0.25. For A-SGD, the worker step sizes were set to 0.00065 and the master step size to a third of this; it must be slower to account for the staleness of the parameters. For SGD, we discovered that a small amount of noise, the same constant as is optimal for p-SNEP, reduces the variance of the runs around their mean by helping some runs avoid getting stuck in bad solutions. However, SGD did not converge to within the plot limits within the time set for experiments and so is omitted in the results shown in Figure 4.6a.

p-SNEP was found to significantly outperform A-SGD and EASGD, obtaining an extra percent of accuracy relative to other methods after fifty epochs for 8 and 16 workers. From the perspective of traditional neural network learning, our algorithm has two advantages for learning deep models: the addition of noise, and a principled method of regularizing the parameters. As in Neelakantan et al. (2016), we found



(a)



(b)

Figure 4.6: Results on deep narrow model for MNIST. (a) Comparing distributed methods for 8 workers. p-SNEP attains an extra percent of accuracy with less variability in the runs. The solid line is the average over 10 runs. (b) Varying the number of workers, averaged over 10 runs. The dashed line indicates result with no prelearning phase for A-SGD and EASGD, whereas for those methods the solid line indicates an Adam pre-learning phase of a third of an epoch.

it was important to add noise to the SGD methods. Although with a good initialization, added noise only improved the worst runs and not the best, smoothing out convergence. We conjecture that the added noise helps learning escape the difficult saddle points and plateaus. For this model, a strong prior variance was found to be important for p-SNEP to extract the full capacity of the model. We conjecture that this allows the gradients to flow back through the network more easily by forcing the parameters of the top layers to be small. Surprisingly, we found L2 regularization did not help with A-SGD and EASGD. p-SNEP also clearly benefits from additional workers in this setting, in contrast to A-SGD and EASGD (see Figure 4.6b).

It is interesting to note that this model has about one sixth the parameters of the previous two layer model. Yet, due to the deeper architecture and ability to successfully navigate the difficult learning landscape, p-SNEP is able to obtain similar test accuracies.

4.6.2.3 CIFAR-10, Convolutional Network

We also experimented with distributed Bayesian learning of convolutional neural networks, applying these to the CIFAR-10 data set (Krizhevsky, 2009) which consists of 50000 training instances and 10000 test instances from 10 classes, each instance being a 32x32 colour natural image. The network used is the one described in Alex Krizhesky's CIFAR tutorial[6] and consists of 8 layers: a first convolutional layer followed by max-pooling and local response normalization, a second convolutional layer also followed by max-pooling and local response normalization, and a third convolutional layer followed by a fully connected layer.

For SNEP, we used the following settings: $N_{sync} = 10$, $N_{outer} = 10$, mini-batch size 100, weights initialised with the Xavier initialisation, and SNEP learning rate and adaptive SGLD step sizes of 0.02. We did not investigate improving performance by building in invariances using data perturbations or having multiple learning phases with different learning rates. Learning curves averaged over 3 runs are shown in Figure 4.7. All distributed algorithms are shown with hand-tuned optimal hyperparameter settings. On this data set SNEP outperforms other distributed algorithms. It converges as quickly as EASGD and p-SNEP but reaches a better final test accuracy. p-SNEP converges very fast but to a worse local optimum. It is not clear why p-SNEP performed better on previous experiments but not here; understanding the optimal choice of β is an interesting area for future research as we have observed that this parameter has a strong influence on the performance of the algorithm.

[6] `https://code.google.com/p/cuda-convnet/wiki/Methodology`

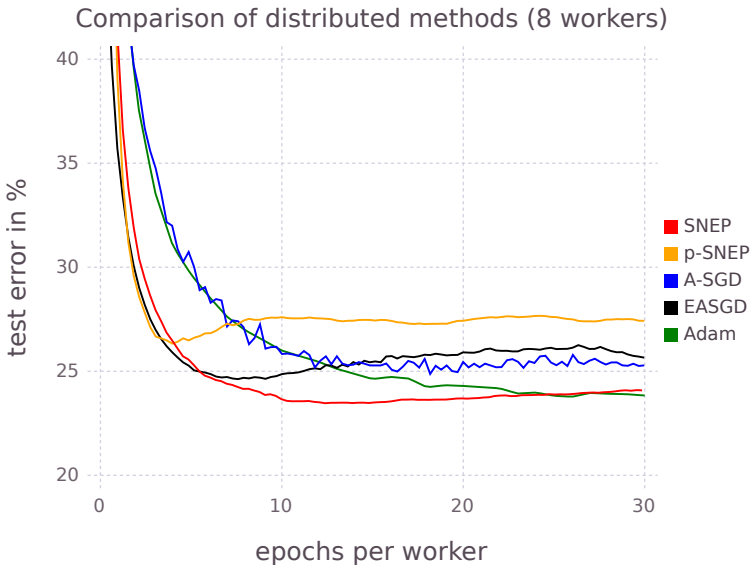


Figure 4.7: Learning curves varying the number of workers (averaged over three runs). The green line shows Adam with default parameters for comparison. Best viewed in colour.

4.7 Discussion

In this paper, we have proposed a novel alternative to expectation propagation called stochastic natural-gradient expectation propagation (SNEP). SNEP is tolerant to Monte Carlo estimation of the moments/mean parameters of tilted distributions and can be shown to be convergent under certain albeit impractical conditions, provided the objective is bounded. Experimentally, we find that SNEP converges more efficiently and more stably than other methods considered, particularly when Monte Carlo noise is high. Using SNEP, we have proposed the posterior server architecture for distributed Bayesian learning using an asynchronous non-blocking message-passing protocol. The architecture uses a separate MCMC sampler on each worker, and SNEP to coordinate the samplers across the cluster so that the target distributions agree on the moments which characterise the base exponential family. In contrast with typical maximum likelihood parameter server architectures, the posterior server allows each worker to learn separate variational parameters, and as a result requires less frequent synchronisation across the cluster. We believe that this insight can allow for significant advances to distributed learning, although more work is still needed to make this reality. Finally, we applied SNEP and the posterior server to distributed

Bayesian learning of both fully-connected and convolutional neural networks, where we showed performances on par with or better than state-of-the-art distributed and non-distributed optimisation algorithms. In conclusion, we have demonstrated that Bayesian learning of large complex models can be achieved efficiently and effectively in a distributing computing setting, and can achieve state-of-the-art performances on learning neural networks.

Our work leaves open a number of interesting avenues of future research, which we have discussed throughout the paper. It would be interesting to develop other novel convergent alternatives to EP with potentially faster convergence (e.g. using a single loop instead of double loop algorithm), and to apply such methods to other settings where Monte Carlo estimates are used within EP. It would also be interesting to further investigate combining SNEP with ideas from other projects such as the parameter tying idea of stochastic and averaged EP. We also believe that further explorations of learning regimes and hyperparameters, as well as of larger data sets, is called for, and will demonstrate further improvements to the method. These explorations can include using samples obtained at workers to form predictive probabilities, evaluating the algorithms using test log probabilities and on their abilities to quantify uncertainties, understanding the role of the β parameter, and combining SNEP with simulated annealing for optimisation. Another interesting area for application is on-device machine learning which falls naturally into our distributed data set-up. It would be interesting to compare SNEP to existing approaches such as federated learning (McMahan et al., 2016). We have also noted that our application of SNEP to neural network learning necessitates a diagonal covariance Gaussian approximation for computational reasons. Further work can consider richer posterior approximations, such as block-diagonal covariances or the use of matrix variate Gaussian distributions (Louizos and Welling, 2016). Ultimately, we believe these explorations will demonstrate the utility of a distributed Bayesian approach to learning.

Acknowledgements LH is funded by the EPSRC OxWaSP CDT through grant EP/L016710/1. SW gratefully acknowledges support from the EPSRC AIMS CDT through grant EP/L015987/2. TL gratefully acknowledges funding from the Scatcherd European scholarship and EPSRC Grant EP/L505031/1. SJV thanks EPSRC for funding through EPSRC Grants EP/N000188/1 and EP/K009850/1. YWT gratefully acknowledges EPSRC for research funding through grant EP/K009362/1, and the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 617071. The authors thank

Daniel Hernández-Lobato, Yingzhen Li, Thang Bui and Rich Turner for valuable feedback on the preprint and suggestions about tied EP factors.

4.8 Supplemenatary Material

4.8.1 Relationship of Ideal Variational Problem in the Extended Exponential Family to Variational Inference

As expected, the ideal variational problem in the extended exponential family in (4.3) is intractable and approximations are needed for tractability, with different approximations leading to different variational approaches. For readers unfamiliar with this framework, it might be illuminating to link the above framework to standard mean field variational inference, which corresponds to approximating the mean domain $\tilde{\mathcal{M}}$ by the set of moments achievable by some family of factorised distributions q over x. The standard evidence lower bound on the log marginal probability (also known as the variational free energy) is,

$$\begin{aligned} p(\{D_i\}_{i=1}^n) &\geq \mathbb{E}_q[\log p(x, \{D_i\}_{i=1}^n) - \log q(x)] \\ &= \mathbb{E}_q\left[\theta_0^\top s(x) - A(\theta_0) + \sum_{i=1}^n \ell_i(x)\right] - \mathbb{E}_q\left[\log q(x)\right] \\ &= \tilde{\theta}^\top \tilde{\mu} - A(\theta_0) - \mathbb{E}_q\left[\log q(x)\right] \end{aligned}$$

where we have introduced $\mu = \mathbb{E}_q[s(x)]$ and $\nu_i = \mathbb{E}_q[\ell_i(x)]$ as the moments of q, and we have used the definition of the extended natural and mean parameters $\tilde{\theta}$, $\tilde{\mu}$. The second term is a constant not dependent on q, while the third term above is the negative entropy, so that the above is equivalent to (4.3) except that the optimisation is only over a family of factorised distributions q.

4.8.2 Additional Techniques for Bayesian Neural Networks

In our experiments we investigated the use of SNEP and the posterior server for distributed Bayesian learning of neural networks. To get the method working well on the notoriously complicated posterior distributions for neural networks, a number of additional techniques are needed, which we describe here.

4.8.2.1 Adaptive Stochastic Gradient Langevin Dynamics

Most of the computational costs associated with the algorithm involve the MCMC updates to the state x_i. When the number of data points stored on each compute node

is large, standard MCMC updates are infeasible as each update requires computations involving every data point. In our experiments we used the stochastic gradient Langevin dynamics (SGLD) algorithm proposed by Welling and Teh (2011) which scales well to large data sets.

SGLD uses mini-batches of data to provide unbiased estimates of gradients which are used in a time discretized Langevin dynamics simulation whose stationary distribution is the desired tilted distribution (4.20). SGLD injects noise in every step. As the discretization stepsize decreases the noise due to the stochastic gradients is eventually dominated by the injected noise and hence neglibible. Likewise the discretization introduces errors which tend to zero as the discretization step sizes decreases to zero; see (Teh et al., 2015, Vollmer et al., 2016). Recall that the data points on compute node i is $D_i = \{y_c\}_{c \in S_i}$, and the log likelihood is

$$\ell_i(x_i) = \sum_{c \in S_i} \log p(y_c \mid x_i).$$

Let $B^{(t)} \subset S_i$ be a mini-batch of data, chosen uniformly at random with fixed size. Each SGLD update is,

$$x_i^{(t+1)} = x_i^{(t)} + \kappa_t (M^{(t)})^{-1} \left(\nabla s(x_i^{(t)})^\top (\theta_i' - \beta_i^{-1} \lambda_i^{(t)}) + \beta_i^{-1} \frac{|S_i|}{|B^{(t)}|} \sum_{c \in B^{(t)}} \nabla \log p(y_c | x_i^{(t)}) \right) + \eta_t,$$
$$\eta_t \sim \mathcal{N}(0, 2\kappa_t (M^{(t)})^{-1}). \tag{4.30}$$

The term inside the parentheses is an unbiased estimate of the gradient of the log density of the tilted distribution (4.20), κ_t is the discretization step size, $M^{(t)}$ is (an adaptive) diagonal mass matrix, while η_t is an injected normally-distributed noise, which prevents SGLD from converging to a mode of the distribution and distinguishes it from stochastic gradient descent. See Welling and Teh (2011) for details.

In (4.30), $M^{(t)}$ is a mass matrix which effectively controls the length scale of updates to each dimension of x_i. It is well known that in neural networks the length scales of gradients differ significantly across different parameters and adaptation of learning rates specific to each parameter is crucial for successful deployment of stochastic gradient descent learning. We have found that this is the case for SGLD as well, and used an adaptation scheme for the mass matrix reminiscent of Adam (Kingma and Ba, 2015). The adaptation scheme is the same as the one proposed by (Li et al., 2016a) except for the addition of a debiasing reminiscent of Adam. At iteration t let g_t be the stochastic gradient estimate in (4.30). We use a diagonal mass

matrix M_t that is updated according to the following update equations:

$$v_t = \beta v_{t-1} + (1-\beta) g_t \odot g_t$$
$$\text{diag}(M_t) = \frac{v_t}{1-\beta^2},$$

where $\odot$ denotes the elementwise product. We used $\beta = 0.999$ in our experiments. The actual adaptive stepsize is then given by

$$\kappa_t = \frac{\epsilon}{\sqrt{M_t} + \delta},$$

where $\epsilon = 10^{-3}$ and $\delta = 10^{-8}$ is added for numerical stability. We found that sometimes this adaptive scheme lead to large injected noise, specially at the start of training which was detrimental to learning. To mitigate this effect we limited the standard deviation of injected noise to be at most ϵ in our experiments. An alternative solution to this problem could be the approach proposed by Lu et al. (2017).

4.8.2.2 Shifting MCMC States After Communication with Posterior Server

After each communication with the posterior server, the target distribution of the MCMC sampler on the worker, say i, is changed, because of θ_{-i} being updated in Step 16 of Algorithm 8. Assuming that the MCMC sampler had previously converged, it will now not be so anymore because of this shift in the target distribution, and a number of burn-in iterations may be needed before the mean parameter estimates can be used for SNEP updates again.

For Gaussian base exponential families, we can shift the MCMC state along with the target distribution when θ_{-i} is updated in the following way. Suppose μ_i^{old}, Σ_i^{old}, μ_i^{new}, Σ_i^{new} are the means and covariances of the approximate Gaussian posterior before and after the update to θ_{-i} (with natural parameters $\lambda_i + \theta_{-i}^{\text{old}}$, $\lambda_i + \theta_{-i}^{\text{new}}$ respectively where λ_i is the current natural parameter of the Gaussian likelihood approximation). Suppose x_i^{old} is the MCMC state before the update. Then we shift the MCMC state as follows:

$$x_i^{\text{new}} = \mu_i^{\text{new}} + (\Sigma_i^{\text{new}})^{\frac{1}{2}} (\Sigma_i^{\text{old}})^{-\frac{1}{2}} (x_i^{\text{old}} - \mu_i^{\text{old}}). \tag{4.31}$$

The idea is that x_i^{new} should be at the same location relative to the new Gaussian approximation to the posterior as x_i^{old} is relative to the old Gaussian approximation. We have found that no burn-in is needed with this shift in the MCMC state after each communication.

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	Distributed Bayesian Learning with Stochastic Natural Gradient Expectation Propagation
Publication Status	x Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work
Publication Details	Leonard Hasenclever, Stefan Webb, Thibaut Lienart, Sebastian Vollmer, Balaji Lakshminarayanan, Charles Blundell, Yee Whye Teh, Distributed Bayesian Learning with Stochastic Natural Gradient Expectation Propagation and the Posterior Server, *Journal of Machine Learning Research*, vol. 18, no. 106, pp. 1–37, 2017.

Student Confirmation

Student Name:	Leonard Hasenclever	
Contribution to the Paper	I contributed heavily to implementation of this algorithm. I ran a large part of the experiments with deep learning models and contributed to the experimental analysis as well as writing the paper. The idea for the algorithm is due to Yee Whye Teh.	
Signature	Date	

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: Professor Yee Whye Teh		
Supervisor comments		
Signature	Date	

Chapter 5

Sylvester Normalizing Flows

The following chapter is a self-contained paper published as:

> van den Berg*, R., **Hasenclever*, L.**, Tomczak, J.M. and Welling, M., 2018. Sylvester normalizing flows for variational inference. Conference on Uncertainty in Artificial Intelligence, Oral presentation.

There is a statement of authorship at the end of this chapter.

Abstract

Variational inference relies on flexible approximate posterior distributions. Normalizing flows provide a general recipe to construct flexible variational posteriors. We introduce Sylvester normalizing flows, which can be seen as a generalization of planar flows. Sylvester normalizing flows remove the well-known single-unit bottleneck from planar flows, making a single transformation much more flexible. We compare the performance of Sylvester normalizing flows against planar flows and inverse autoregressive flows and demonstrate that they compare favorably on several datasets.

5.1 Introduction

Stochastic variational inference (Hoffman et al., 2013b) allows for posterior inference in increasingly large and complex problems using stochastic gradient ascent. In continuous latent variable models, variational inference can be made particularly efficient through the amortized inference, in which inference networks amortize the cost of calculating the variational posterior for a data point. A particularly successful class of models is the variational autoencoder (VAE) in which both the generative model and the inference network are given by neural networks, and sampling from the

variational posterior is efficient through the non-centered parameterization (Kingma and Welling, 2014), also known as the reparameterization trick (Kingma and Welling, 2013, Rezende et al., 2014).

Despite its success, variational inference has drawbacks compared to other inference methods such as MCMC. Variational inference searches for the best posterior approximation within a parametric family of distributions. Hence, the true posterior distribution can only be recovered exactly if it happens to be in the chosen family. In particular, with widely used simple variational families such as diagonal covariance Gaussian distributions, the variational approximation is likely to be insufficient. More complex variational families enable better posterior approximations, resulting in improved model performance. Therefore, designing tractable and more expressive variational families is an important problem in variational inference (Nalisnick et al., 2016, Salimans et al., 2014, Tran et al., 2015).

Rezende and Mohamed (2015) introduced a general framework for constructing more flexible variational distributions, called normalizing flows. Normalizing flows transform a base density through a number of invertible parametric transformations with tractable Jacobians into more complicated distributions. They proposed two classes of normalizing flows: planar flows and radial flows. While effective for small problems, these can be hard to train and often many transformations are required to get good performance. For planar flows, Kingma et al. (2016) argue that this is due to the fact that the transformation used acts as a bottleneck, warping one direction at a time. Having a large number of flows makes the inference network very deep and harder to train, empirically resulting in suboptimal performance. Kingma et al. (2016) proposed inverse auto-regressive flows (IAF), achieving state of the art results on dynamically binarized MNIST at the time of publication. While very successful, IAFs require a very large number of parameters. Due to the large number of parameters IAFs cannot amortize all flow parameters. Instead amortization is achieved through an additional context vector that is fed into each flow step.

Paper contribution In this paper, we use Sylvester's determinant identity to introduce Sylvester normalizing flows (SNFs). This family of flows is a generalization of planar flows, removing the bottleneck. We compare a number of different variants of SNFs and show that they compare favorably against planar flows and IAFs. We show that one specific variant of SNFs is related to IAFs, while requiring many fewer parameters due to direct amortization of all flow parameters.

5.2 Variational Inference

Consider a probabilistic model with observations $\mathbf{x}$ and continuous latent variables $\mathbf{z}$ and model parameters θ. In generative modeling we are often interested in performing maximum (marginal) likelihood learning of the parameters θ of the latent-variable model $p_\theta(\mathbf{x}, \mathbf{z})$. This requires marginalization over the unobserved latent variables $\mathbf{z}$. Unfortunately, this integration is generally intractable. Variational inference (Jordan et al., 1999) instead introduces a variational approximation $q_\phi(\mathbf{z}|\mathbf{x})$ to the posterior with learnable parameters ϕ, to construct a lower bound on the log marginal likelihood:

$$\log p_\theta(\mathbf{x}) \geq \log p_\theta(\mathbf{x}) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z}|\mathbf{x})) \tag{5.1}$$

$$= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z})) \tag{5.2}$$

$$=: -\mathcal{F}(\theta, \phi) \tag{5.3}$$

This bound is known as the evidence lower bound (ELBO) and $\mathcal{F}$ is referred to as the variational free energy. In equation (5.2), the first term represents the reconstruction error, and the second term is the Kullback-Leibler (KL) divergence from the approximate posterior to the prior distribution, which acts as a regularizer. In this paper we consider variational autoencoders (VAEs), where both $p_\theta(\mathbf{x}|\mathbf{z})$ and $q_\phi(\mathbf{z}|\mathbf{x})$ are distributions parametrized by neural networks. The parameters θ and ϕ of the generative model and inference model, respectively, are trained jointly through stochastic minimisation of $\mathcal{F}$ which can be made efficient through the reparameterization trick (Kingma and Welling, 2013, Rezende et al., 2014).

From equation (5.1) we see that the better the variational approximation to the posterior the tighter the ELBO. The simplest, but probably most widely used choice of variation distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is diagonal-covariance Gaussians of the form $\mathcal{N}(\boldsymbol{\mu}(\mathbf{x}),\ \boldsymbol{\sigma}^2(\mathbf{x}))$ However, with such simple variational distributions the ELBO will be fairly loose, resulting in biased maximum likelihood estimates of the model parameters θ (see Fig. 5.1) and harming generative performance. Thus, for variational inference to work well, more flexible approximate posterior distributions are needed.

5.2.1 Normalizing Flows

Rezende and Mohamed (2015) propose a way to construct more flexible posteriors by transforming a simple base distribution with a series of invertible transformations

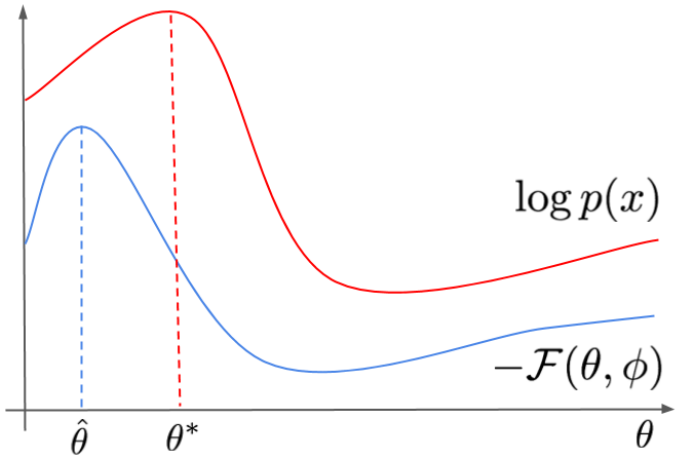


Figure 5.1: Since the ELBO is only a lower bound on the log marginal likelihood, they do not share the same local maxima. The looser the ELBO is the more this can bias maximum likelihood estimates of the model parameters.

(known as normalizing flows) with easily computable Jacobians. The resulting transformed density after one such transformation f is as follows (Tabak and Turner, 2013, Tabak and Vanden-Eijnden, 2010):

$$p_1(\mathbf{z}') = p_0(\mathbf{z}) \left| \det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1}, \tag{5.4}$$

where $\mathbf{z}' = f(\mathbf{z})$, $\mathbf{z}, \mathbf{z}' \in \mathbb{R}^D$ and $f : \mathbb{R}^D \mapsto \mathbb{R}^D$ is an invertible function. In general the cost of computing the Jacobian will be $\mathcal{O}(D^3)$. However, it is possible to design transformations with more efficiently computable Jacobians.

This strategy is used in variational inference as follows: first, a stochastic variable is drawn from a simple base posterior distribution such as a diagonal Gaussian $\mathcal{N}(\mathbf{z}_0|\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}^2(\mathbf{x}))$. The sample is then transformed with a number of flows. After applying K flows, the final latent stochastic variables are given by $\mathbf{z}_K = f_K \circ \ldots f_2 \circ f_1(\mathbf{z}_0)$. The corresponding log-density is then given by:

$$\begin{aligned} \log q_K(\mathbf{z}_K|\mathbf{x}) = & \log q_0(\mathbf{z}_0|\mathbf{x}) \\ & - \sum_{k=1}^{K} \log \left| \det \left(\frac{\partial f_k(\mathbf{z}_{k-1}; \lambda_k(\mathbf{x}))}{\partial \mathbf{z}_{k-1}} \right) \right|, \end{aligned} \tag{5.5}$$

where λ_k are the parameters of the k-th transformation. Given variational posterior $q_\phi(\mathbf{z}|\mathbf{x}) = q_K(\mathbf{z}|\mathbf{x})$ parametrized by a normalizing flow of length K, the variational

objective can be rewritten as:

$$\begin{aligned}\mathcal{F}(\theta, \phi) &= \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p_\theta(\mathbf{x}, \mathbf{z})] \qquad (5.6)\\ &= \mathbb{E}_{q_0}[\log q_0(\mathbf{z}_0|\mathbf{x}) - \log p_\theta(\mathbf{x}, \mathbf{z})] \\ &\quad - \mathbb{E}_{q_0}\left[\sum_{k=1}^{K} \log \left|\det\left(\frac{\partial f_k(\mathbf{z}_{k-1}; \lambda_k(\mathbf{x}))}{\partial \mathbf{z}_{k-1}}\right)\right|\right]. \qquad (5.7)\end{aligned}$$

Normalizing flows are normally used with amortized variational inference. Instead of learning variational parameters for each data point, both $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, as well as all the flow parameters are outputs of a deep neural network conditioned on $\mathbf{x}$. This is referred to as the inference network.

Rezende and Mohamed (2015) introduced a normalizing flow, called planar flow, for which the Jacobian determinant could be computed efficiently. A single transformation of the planar flow is given by:

$$\mathbf{z}' = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b). \qquad (5.8)$$

Here, $\mathbf{u}, \mathbf{w} \in \mathbb{R}^D$, $b \in \mathbb{R}$ and h is a suitable smooth activation function. Rezende and Mohamed (2015) show that for $h = \tanh$, transformations of this kind are invertible as long as $\mathbf{u}^T\mathbf{w} \geq -1$.

By the *Matrix determinant lemma* the Jacobian of this transformation is given by:

$$\begin{aligned}\det\left(\frac{\partial \mathbf{z}'}{\partial \mathbf{z}}\right) &= \det\left(\mathbf{I} + \mathbf{u}h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{w}^T\right) \\ &= 1 + \mathbf{u}^T h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{w}, \qquad (5.9)\end{aligned}$$

where h' denotes the derivative of h and which can be computed in $O(D)$ time.

In practice, many planar flow transformations are required to transform a simple base distribution into a flexible distribution, especially for high dimensional latent spaces. Kingma et al. (2016) argue that this is related to the term $\mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$ in Eq. (5.8), which effectively acts as a single-neuron MLP. In the next section we will derive a generalization of planar flows, which does not have a single-neuron bottleneck, while still maintaining the property of an efficiently computable Jacobian determinant.

5.3 Sylvester Normalizing Flows

Consider the following more general transformation similar to a single layer MLP with M hidden units and a residual connection:

$$\mathbf{z}' = \mathbf{z} + \mathbf{A}h(\mathbf{B}\mathbf{z} + \mathbf{b}), \tag{5.10}$$

with $\mathbf{A} \in \mathbb{R}^{D\times M}, \mathbf{B} \in \mathbb{R}^{M\times D}$, $\mathbf{b} \in \mathbb{R}^M$, and $M \leq D$. The Jacobian determinant of this transformation can be obtained using Sylvester's determinant identity, which is a generalization of the matrix determinant lemma.

Theorem 1 (Sylvester's determinant identity)**.** *For all* $\mathbf{A} \in \mathbb{R}^{D\times M}, \mathbf{B} \in \mathbb{R}^{M\times D}$,

$$\det\left(\mathbf{I}_D + \mathbf{A}\mathbf{B}\right) = \det\left(\mathbf{I}_M + \mathbf{B}\mathbf{A}\right), \tag{5.11}$$

where $\mathbf{I}_M$ *and* $\mathbf{I}_D$ *are* M *and* D*-dimensional identity matrices, respectively.*

When $M < D$, the computation of the determinant of a $D \times D$ matrix is thus reduced to the computation of the determinant of an $M \times M$ matrix.

Using Sylvester's determinant identity, the Jacobian determinant of the transformation in Eq. (5.10) is given by:

$$\det\left(\frac{\partial \mathbf{z}'}{\partial \mathbf{z}}\right) = \det\left(\mathbf{I}_M + \operatorname{diag}\left(h'(\mathbf{B}\mathbf{z} + \mathbf{b})\right)\mathbf{B}\mathbf{A}\right). \tag{5.12}$$

Since Sylvester's determinant identity plays a crucial role in the proposed family of normalizing flows, we will refer to them as *Sylvester normalizing flows*.

5.3.1 Parametrization of A and B

In general, the transformation in (5.10) will not be invertible. Therefore, we propose the following special case of the above transformation:

$$\mathbf{z}' = \mathbf{z} + \mathbf{Q}\mathbf{R}h(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{z} + \mathbf{b}) = \phi(\mathbf{z}), \tag{5.13}$$

where $\mathbf{R}$ and $\tilde{\mathbf{R}}$ are upper triangular $M \times M$ matrices, and

$$\mathbf{Q} = (\mathbf{q}_1 \dots \mathbf{q}_M)$$

with the columns $\mathbf{q}_m \in \mathbb{R}^D$ forming an orthonormal set of vectors. By theorem 1, the determinant of the Jacobian $\mathbf{J}$ of this transformation reduces to:

$$\begin{aligned}\det \mathbf{J} &= \det\left(\mathbf{I}_M + \operatorname{diag}\left(h'(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{z} + \mathbf{b})\right)\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{Q}\mathbf{R}\right) \\ &= \det\left(\mathbf{I}_M + \operatorname{diag}\left(h'(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{z} + \mathbf{b})\right)\tilde{\mathbf{R}}\mathbf{R}\right),\end{aligned} \tag{5.14}$$

which can be computed in $O(M)$, since $\tilde{\mathbf{R}}\mathbf{R}$ is also upper triangular. The following theorem gives a sufficient condition for this transformation to be invertible.

Theorem 2. *Let* $\mathbf{R}$ *and* $\tilde{\mathbf{R}}$ *be upper triangular matrices. Let* $h : \mathbb{R} \longrightarrow \mathbb{R}$ *be a smooth function with bounded, positive derivative. Then, if the diagonal entries of* $\mathbf{R}$ *and* $\tilde{\mathbf{R}}$ *satisfy* $r_{ii}\tilde{r}_{ii} > -1/\|h'\|_\infty$ *and* $\tilde{\mathbf{R}}$ *is invertible, the transformation given by* (5.13) *is invertible.*

Proof. **Case 1: R and $\tilde{\mathbf{R}}$ diagonal**

Recall that one-dimensional real functions with strictly positive derivatives are invertible. The columns of $\mathbf{Q}$ are orthonormal and span a subspace $\mathcal{W} = \text{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_M\}$ of $\mathbb{R}^D$. Let $\mathcal{W}^\perp$ denote its orthogonal complement. We can decompose $\mathbf{z} = \mathbf{z}_\parallel + \mathbf{z}_\perp$, where $\mathbf{z}_\parallel \in \mathcal{W}$ and $\mathbf{z}_\perp \in \mathcal{W}^\perp$. Similarly we can decompose $\mathbf{z}' = \mathbf{z}'_\parallel + \mathbf{z}'_\perp$. Clearly, $\mathbf{Q}\mathbf{R}h(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{z} + \mathbf{b}) \in \mathcal{W}$. Hence ϕ only acts on $\mathbf{z}_\parallel$ and $\mathbf{z}_\perp = \phi(\mathbf{z})_\perp = \mathbf{z}'_\perp$. Thus, it suffices to consider the effect of ϕ on $\mathbf{z}_\parallel$. Multiplying (5.13) by $\mathbf{Q}^T$ from the left gives:

$$\begin{aligned}\underbrace{\mathbf{Q}^T\mathbf{z}'}_{\mathbf{v}'} &= \underbrace{\mathbf{Q}^T\mathbf{z}}_{\mathbf{v}} + \mathbf{R}h(\tilde{\mathbf{R}}\underbrace{\mathbf{Q}^T\mathbf{z}}_{\mathbf{v}} + \mathbf{b}) \\ &= (f_1(v_1), \ldots, f_M(v_M))^T, \end{aligned} \tag{5.15}$$

where the vectors $\mathbf{v}$ and $\mathbf{v}'$ are the respective coordinates of $\mathbf{z}_\parallel$ and $\mathbf{z}'_\parallel$ w.r.t. $\mathbf{q}_1, \ldots, \mathbf{q}_M$. The dimensions in (5.15) are completely independent and each dimension is transformed by a real function $f_i(v) = v + r_{ii}h(\tilde{r}_{ii}v + b_i)$. Consider a single dimension i of (5.15). Since $\|h'\|_\infty r_{ii}\tilde{r}_{ii} > -1$, we have $f'_i(v) > 0$ and thus f_i is invertible. Since all dimensions are independent and the transformation is invertible in each dimension we can find $f^{-1} : \mathcal{W} \to \mathcal{W}$ such that $\mathbf{z}_\parallel = f^{-1}(\mathbf{z}'_\parallel)$. Hence we can write the inverse of ϕ as:

$$\phi^{-1}(\mathbf{z}') = \underbrace{\mathbf{z}'_\perp}_{\mathbf{z}_\perp} + \underbrace{f^{-1}(\mathbf{z}'_\parallel)}_{\mathbf{z}_\parallel} = \mathbf{z}, \tag{5.16}$$

Case 2: R triangular, $\tilde{\mathbf{R}}$ diagonal

Let us now consider the case when $\mathbf{R}$ is an upper triangular matrix. By the argument for the diagonal case above, it suffices to consider the effect of the transformation in $\mathcal{W}$. Multiplying (5.13) by $\mathbf{Q}^T$ from the left gives:

$$\underbrace{\mathbf{Q}^T\mathbf{z}'}_{\mathbf{v}'} = \underbrace{\mathbf{Q}^T\mathbf{z}}_{\mathbf{v}} + \mathbf{R}h(\tilde{\mathbf{R}}\underbrace{\mathbf{Q}^T\mathbf{z}}_{\mathbf{v}} + \mathbf{b}) \tag{5.17}$$

where the vectors $\mathbf{v}$ and $\mathbf{v}'$ contain the respective coordinates of $\mathbf{z}_\parallel$ and $\mathbf{z}'_\parallel$ w.r.t. $\mathbf{q}_1, \ldots, \mathbf{q}_M$. As in the diagonal case consider the functions $f_i(v) = v + r_{ii}h(\tilde{r}_{ii}v + b_i)$.

Since $\|h'\|_\infty r_{ii}\tilde{r}_{ii} > -1$, we have $f_i'(v) > 0$ and thus f_i is invertible. Let us rewrite (5.17) in terms of f_i:

$$v_1' = f_1(v_1) + \sum_{j=2}^{M} r_{1j} h(\tilde{r}_{jj} v_j + b_j) \tag{5.18}$$

$$\dots$$

$$v_k' = f_k(v_k) + \sum_{j=k+1}^{M} r_{kj} h(\tilde{r}_{jj} v_j + b_j) \tag{5.19}$$

$$\dots$$

$$v_M' = f_M(v_M) \tag{5.20}$$

Since f_M is invertible we can write $v_M = f_M^{-1}(v_M')$. Now suppose we have expressed $\{v_j, \forall j > k\}$ in terms of $\{v_j', \forall j > k\}$. Then

$$\begin{aligned} f_k(v_k) &= v_k' - \underbrace{\sum_{j=k+1}^{M} r_{kj} h(\tilde{r}_{jj} v_j + b_j)}_{\text{some function of } \{v_j', \forall j > k\}} \\ &=: g_k(v_k', v_{k+1}', \dots, v_M') \\ v_k &= f_k^{-1}(g_k(v_k', v_{k+1}', \dots, v_M')). \end{aligned} \tag{5.21}$$

Thus we have expressed $\{v_j, \forall j \geq k\}$ in terms of $\{v_j', \forall j \geq k\}$. By induction, we can express $\{v_j, \forall j\}$ in terms of $\{v_j', \forall j\}$ and hence the transformation is invertible.

Case 3: R and $\tilde{\mathbf{R}}$ triangular

Now consider the general case when $\tilde{\mathbf{R}}$ is triangular. As before we only need to consider the effect of the transformation in $\mathcal{W}$.

$$\underbrace{\mathbf{Q}^T\mathbf{z}'}_{\mathbf{v}'} = \underbrace{\mathbf{Q}^T\mathbf{z}}_{\mathbf{v}} + \mathbf{R}h(\tilde{\mathbf{R}}\underbrace{\mathbf{Q}^T\mathbf{z}}_{\mathbf{v}} + \mathbf{b}) \tag{5.22}$$

Let g be the function $g(\mathbf{v}) = \tilde{\mathbf{R}}\mathbf{v}$. By assumption, g is invertible with inverse g^{-1}. Multiplying (5.22) by $\tilde{\mathbf{R}}$ gives:

$$g(\mathbf{v}') = \underbrace{g(\mathbf{v}) + \tilde{\mathbf{R}}\mathbf{R}h(g(\mathbf{v}) + \mathbf{b})}_{=: f(g(\mathbf{v}))} \tag{5.23}$$

Since $\tilde{\mathbf{R}}\mathbf{R}$ is upper triangular with diagonal entries $\tilde{r}_{jj}r_{jj}$, f is covered by case 2 considered before and is invertible. Thus, $\mathbf{v}$ can be written as:

$$\mathbf{v} = g^{-1}(f^{-1}(g(\mathbf{v}'))). \tag{5.24}$$

Hence the transformation in (5.22) is invertible. □

5.3.2 Preserving Orthogonality of Q

Orthogonality is a convenient property, mathematically, but hard to achieve in practice. In this paper we consider three different flows based on the theorem above and various ways to preserve the orthogonality of $\mathbf{Q}$. The first two use explicit differentiable constructions of orthogonal matrices, while the third variant assumes a specific fixed permutation matrix as the orthogonal matrix.

Orthogonal Sylvester flows. First, we consider a Sylvester flow using matrices with M orthogonal columns (O-SNF). In this flow we can choose $M < D$, and thus introduce a flexible bottleneck. Similar to Hasenclever et al. (2017b), we ensure orthogonality of $\mathbf{Q}$ by applying the following differentiable iterative procedure proposed by (Björck and Bowie, 1971, Kovarik, 1970):

$$\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} \left(\mathbf{I} + \frac{1}{2} \left(\mathbf{I} - \mathbf{Q}^{(k)\top} \mathbf{Q}^{(k)} \right) \right). \tag{5.25}$$

with a sufficient condition for convergence given by $\|\mathbf{Q}^{(0)\top}\mathbf{Q}^{(0)} - \mathbf{I}\|_2 < 1$. Here, the 2-norm of a matrix $\mathbf{X}$ refers to $\|\mathbf{X}\|_2 = \lambda_{\max}(\mathbf{X})$, with $\lambda_{\max}(\mathbf{X})$ representing the largest singular value of $\mathbf{X}$. In our experimental evaluations we ran the iterative procedure until $\|\mathbf{Q}^{(k)\top}\mathbf{Q}^{(k)} - \mathbf{I}\|_F \leq \epsilon$, with $\|\mathbf{X}\|_F$ the Frobenius norm, and ϵ a small convergence threshold. We observed that running this procedure up to 30 steps was sufficient to ensure convergence with respect to this threshold. To minimize the computational overhead introduced by orthogonalization we perform this orthogonalization in parallel for all flows.

Since this orthogonalization procedure is differentiable, it allows for the calculation of gradients with respect to $\mathbf{Q}^{(0)}$ by backpropagation, allowing for any standard optimization scheme such as stochastic gradient descent to be used for updating the flow parameters.

Householder Sylvester flows. Second, we study Householder Sylvester flows (H-SNF) where the orthogonal matrices are constructed by products of Householder reflections. Householder transformations are reflections about hyperplanes. Let $\mathbf{v} \in \mathbb{R}^D$, then the reflection about the hyperplane orthogonal to $\mathbf{v}$ is given by:

$$H(\mathbf{z}) = \mathbf{z} - \frac{\mathbf{v}\mathbf{v}^T}{\|\mathbf{v}\|^2}\mathbf{z} \tag{5.26}$$

It is worth noting that performing a single Householder transformation is very cheap to compute, as it only requires D parameters. Chaining together several Householder transformations results in more general orthogonal matrices, and it can be shown (Bischof and Sun, 1997, Sun and Bischof, 1995) that any $M \times M$ orthogonal matrix can be written as the product of $M - 1$ Householder transformations. In our Householder Sylvester flow, the number of Householder transformations H is a hyperparameter that trades off the number of parameters and the generality of the orthogonal transformation. Note that the use of Householder transformations forces us to use $M = D$, since Householder transformations result in square matrices.

Triangular Sylvester flows. Third, we consider a triangular Sylvester flow (T-SNF), in which all orthogonal matrices $\mathbf{Q}$ alternate per transformation between the identity matrix and the permutation matrix corresponding to reversing the order of $\mathbf{z}$. This is equivalent to alternating between lower and upper triangular $\tilde{\mathbf{R}}$ and $\mathbf{R}$ for each flow.

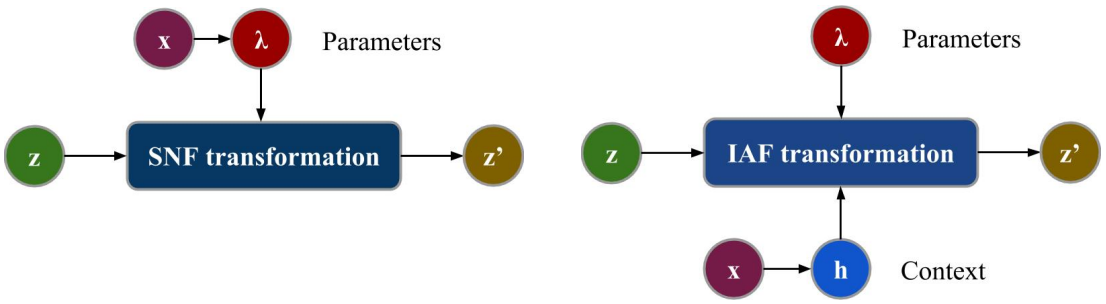


Figure 5.2: Different amortization strategies for Sylvester normalizing flows and Inverse Autoregressive Flows. Left: our inference network produces amortized flow parameters. This strategy is also employed by planar flows. Right: IAF has a large number of parameters, and introduces a measure of $\mathbf{x}$ dependence through a context $\mathbf{h}(\mathbf{x})$. This context acts as an additional input for each transformation. The flow parameters themselves are independent of $\mathbf{x}$.

5.3.3 Amortizing Flow Parameters

When using normalizing flows in an amortized inference setting, the parameters of the base distribution as well as the flow parameters can be functions of the data point $\mathbf{x}$ (Rezende and Mohamed, 2015). Figure 5.2 (left) shows a diagram of one SNF step and the amortization procedure. The inference network takes datapoints $\mathbf{x}$ as input, and provides as an output the mean and variance of $\mathbf{z}^0$ such that $\mathbf{z}^0 \sim \mathcal{N}(\mathbf{z}|\mu^0, \sigma^0)$. Several SNF transformations are then applied to $\mathbf{z}^0 \rightarrow \mathbf{z}^1 \rightarrow \ldots \mathbf{z}^K$, producing a

flexible posterior distribution for $\mathbf{z}^K$. All of the flow parameters ($\mathbf{R}$, $\tilde{\mathbf{R}}$ and $\mathbf{Q}$ for each transformation) are produced as an output by the inference network, and are thus fully amortized.

5.4 Related Work

5.4.1 Normalizing Flows for Variational Inference

A number of invertible transformations with tractable Jacobians have been proposed in recent years. Rezende and Mohamed (2015) first discussed such transformations in the context of stochastic variation inference, coining the term normalizing flows.

Rezende and Mohamed (2015) proposed two different parametric families of transformations with tractable Jacobians: planar and radial flows. While effective for small problems, these transformations are hard to scale to large latent spaces and often require a large number of transformations. The transformation corresponding to planar flows is given in Eq. (5.8).

More recently, a successful class of flows called Inverse Autoregressive Flows was introduced in (Kingma et al., 2016). As the name suggests, one IAF transformation can be seen as the inverse of an autoregressive transformation. Consider the following autoregressive transformation:

$$\begin{aligned} z_0 &= \bar{\mu}_0 + \bar{\sigma}_0 \cdot \epsilon_0 \\ z_i &= \bar{\mu}_i(\mathbf{z}_{1:i-1}) + \bar{\sigma}_i(\mathbf{z}_{1:i-1}) \cdot \epsilon_i, \quad i = 1, \ldots, D \end{aligned} \tag{5.27}$$

with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This transformation models the distribution over the variable $\mathbf{z}$ with an autoregressive factorization $p(\mathbf{z}) = p(z_0) \prod_{i=1}^{D} p(z_i | z_{i-1}, \ldots, z_0)$. Since the parameters of transformation for z_i are dependent on $\mathbf{z}_{1:i-1}$, this procedure requires D sequential steps to sample a single vector $\mathbf{z}$. This is undesirable for variational inference, where sampling occurs for every forward pass.

However, the inverse transformation (which exists if $\bar{\sigma}_i > 0 \; \forall i$) is easy to sample from:

$$\epsilon_i = \frac{z_i - \bar{\mu}_i(\mathbf{z}_{1:i-1})}{\bar{\sigma}_i(\mathbf{z}_{1:i-1})}. \tag{5.28}$$

For this inverse transformation, ϵ_i is no longer dependent on the transformation of ϵ_j for $j \neq i$. Hence, this transformation can be computed in parallel: $\boldsymbol{\epsilon} = (\mathbf{z} -$

$\bar{\boldsymbol{\mu}}(\mathbf{z}))/\bar{\boldsymbol{\sigma}}(\mathbf{z})$. Rewriting $\sigma_i(z_{1:i-1}) = 1/\bar{\sigma}_i(z_{1:i-1})$ and $\mu_i(z_{1:i-1}) = -\bar{\mu}(z_{1:i-1})/\bar{\sigma}_i(z_{1:i-1})$, yields the IAF transformation:

$$z_i^t = \mu_i^t(\mathbf{z}_{1:i-1}^{t-1}) + \sigma_i^t(\mathbf{z}_{1:i-1}^{t-1}) \cdot z_i^{t-1}, \quad i = 1, ..., D. \tag{5.29}$$

Starting from $\mathbf{z}^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, multiple IAF transformations can be stacked on top of each other to produce flexible probability distributions.

If $\boldsymbol{\mu}^t$ and $\boldsymbol{\sigma}^t$ depend on $\mathbf{z}^{t-1}$ linearly, IAF can model full covariance Gaussian distributions. In order to move away from Gaussian distributions to more flexible distributions, it is important that $\boldsymbol{\mu}^t$ and $\boldsymbol{\sigma}^t$ are nonlinear functions of $\mathbf{z}^{t-1}$.

In practice, wide MADEs (Germain et al., 2015) or deep PixelCNN layers (van den Oord et al., 2016) are needed to increase the flexibility of IAF transformations. This results in transformations with a large number of parameters. As shown in Figure 5.2 (right), amortization is achieved through a context $\mathbf{h}(\mathbf{x})$ that is fed into the autoregressive networks as an additional input at every IAF step.

Our Triangular Sylvester flows are strongly related to mean-only IAF transformations ($\boldsymbol{\sigma}^t = 1$). As mentioned in Kingma et al. (2016), between every IAF transformation the order of $\mathbf{z}$ is reversed, in order to ensure that on average all dimensions get warped equally. In T-SNF, the same effect is achieved by using the permutation matrix that reverses the order of $\mathbf{z}$ in every other transformation as the orthogonal matrix. However, mean-only IAF is a volume-preserving transformation, i.e. the determinant of the Jacobian has absolute value one. T-SNF is not volume preserving due to the nonzero elements on the diagonals of $\mathbf{R}$ and $\tilde{\mathbf{R}}$. Note, that in Kingma et al. (2016) it was shown that the empirical difference in performance between mean-only IAF and the general IAF transformation is negligible.

The most important difference between IAF and T-SNF is the way parameters are amortized. In T-SNF, $\mathbf{R}$ and $\tilde{\mathbf{R}}$ are directly amortized functions of the input $\mathbf{x}$ (see Fig. 5.2). This is equivalent to amortizing the MADE parameters in mean-only IAF. Having input dependent MADE parameters allows for flexible transformations with fewer parameters.

Householder Sylvester flows can also be seen as a non-linear extension of Householder flows (Tomczak and Welling, 2016). Householder flows are volume-preserving flows, which transform the variational posterior with a diagonal covariance matrix to a full-covariance posterior. Householder flows are a special case of H-SNF if $h(\mathbf{z}) = \mathbf{z}$, $\mathbf{R}$ is the identity matrix, and the residual connection in Eq. (5.13) is left out.

5.4.2 Normalizing Flows for Density Estimation

A number of invertible transformations have been proposed in the context of density estimation. Note that density estimation requires the inverse of the flow to be tractable. Having a provably invertible transformation is not the same as being able to compute the inverse.

For density estimation with normalizing flows, we are interested maximizing the log-likelihood of the data:

$$\log p(\mathbf{x}) = \log p_0(f^{-1}(\mathbf{x})) + \log \left| \det \left(\frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|. \tag{5.30}$$

Thus, the goal is to transform a complicated data distribution back to a simple distribution. In general, both directions of an invertible transformations need not be tractable. Hence, methods developed for density estimation are generally not directly applicable to variational inference.

Non-linear independent component estimation (NICE, Dinh et al. (2014)) and the related Real NVP (Dinh et al., 2016), and Masked Autoregressive Flow (MAF, Papamakarios et al. (2017)) are recent examples of normalizing flows for density estimation.

In NICE, each transformation splits the variables into two disjoint subsets $\mathbf{z}_A, \mathbf{z}_B$. One of the subsets is transformed as $\mathbf{z}'_A = \mathbf{z}_A + f(\mathbf{z}_B)$, while $\mathbf{z}_B$ is left unchanged. In the next transformation a different subset of variables is transformed. This results in a transformation which is trivially invertible and has a tractable Jacobian. Real NVP uses the same fundamental idea. Appealingly, because of the tractable inverse, NICE and real NVP can generate data and estimate density with one forward pass. However due to fact that only a subset of variables is updated in each transformation many transformations are needed in practice. Rezende and Mohamed (2015) compared NICE to planar flows in the context of variational inference and found that planar flows empirically perform better.

Finally, Papamakarios et al. (2017) showed that fitting an MAF can be seen as fitting an implicit IAF from the data distribution to the base distribution. However, generating data from an MAF density model requires D passes, making it unappealing for variational inference.

5.5 Number of Parameters

Here, we briefly compare the number of parameters needed by planar flows, IAF and the three Sylvester normalizing flows. We denote the size of the stochastic variables

z with D, and the number of output units of the inference network with E.

Planar flows use amortized parameters $\mathbf{u}, \mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ for each flow transformation. Therefore, the number of parameters related to K flow transformations is equal to $2EDK + EK$.

For the implementation of IAF as described in Section 5.6, the inference network needs to produce a context of size C, where C denotes the width of the MADE layers. The total number of flow related learnable parameters then comes down to $EC + K \times (C^2 + 3CD)$.

In the case of Orthogonal Sylvester flows with a bottleneck of size M, we require $KE \times (MD + 2M^2 + M)$ parameters. For Householder Sylvester flows with H Householder reflections per flow transformation, $KE \times (HD + 2D^2 + D)$ parameters are needed. Finally, for triangular Sylvester flows $KE \times (2D^2 + D)$ parameters require optimization.

Planar flows require the smallest number of parameters but generally result in worse results. IAFs on the other hand require a number of parameters that is quadratic in the width of the MADE layers. For good results this has to be quite large and cannot be amortized directly. Instead amortization is achieved through a context vector. In contrast, for SNFs the number of parameters is quadratic in the dimension of the latent space and while large, this can still be amortized directly.

5.6 Experiments

We perform empirical studies of the performance of Sylvester flows on four datasets: statically binarized MNIST, Freyfaces, Omniglot and Caltech 101 Silhouettes. The baseline model is a plain VAE with a fully factorized Gaussian distribution. We furthermore compare against planar flows and Inverse Autoregressive Flows of different sizes.

We use annealing to optimize the lower bound, where the prefactor of the KL divergence is linearly increased from 0 to 1 during 100 epochs as suggested by Bowman et al. (2015) and Sønderby et al. (2016). A learning rate of 0.0005 was used in all experiments. In order to obtain estimates for the negative log likelihood we used importance sampling (as proposed in (Rezende et al., 2014)). Unless otherwise stated, 5000 importance samples were used.

While this form of evaluation is standard in the literature, it is potentially problematic for two reasons: Firstly, due to the high dimensional nature of the latent space importance weighting could lead to degenerate importance weights and high

variances estimates. Secondly, estimates of the marginal log likelihood obtained via importance sampling will be upper bounds on the true log likelihood. Since different models are evaluated with the different proposal distributions it is not completely clear whether higher estimates of the log likelihood are evidence of higher log likelihoods and thus of better models or simply a tighter estimates of the same underlying log likelihood. Ultimately it seems likely that more expressive encoder distributions lead to both tighter bounds and better models.

In order to assess the performance of the different flows properly, we use the same base encoder and decoder architecture for all models. We use gated convolutions and transposed convolutions as base layers for the encoder and decoder architecture respectively. The inference network consists of several gated convolution layers that produce a hidden unit vector. After being flattened, these hidden units act as an input to two fully connected layers that predict the mean and variance of $\mathbf{z}^0$.

For planar and Sylvester flows, the flattened hidden units are passed to a separate linear layer that output the amortized flow parameters. For IAF, the flattened hidden units are also passed to a linear layer to produce the context vector $\mathbf{h}_{\text{context}}(\mathbf{x})$. For details of the architecture see Section A of the appendix. In all models the latent space is of dimension 64.

We use the following implementation for each IAF transformation[1]: one IAF transformation first applies one MADE Layer (denoted as MaskedLinear) followed by a nonlinearity to the input z, upscaling it to a hidden variable of size M. At this point the context vector $\mathbf{h}_{\text{context}}(\mathbf{x})$ is added to the hidden units, after which two more masked layers are applied to produce the mean and scale of the IAF transformation:

$$\begin{aligned}
&\mathbf{h}_z \leftarrow \text{ELU}(\text{MaskedLinear}(\mathbf{z})) \\
&\mathbf{h} \leftarrow \mathbf{h}_z + \mathbf{h}_{\text{context}}(\mathbf{x}) \\
&\mathbf{h} \leftarrow \text{ELU}(\text{MaskedLinear}(\mathbf{h})) \\
&\boldsymbol{\mu} \leftarrow \text{MaskedLinear}(\mathbf{h}), \quad \mathbf{s} \leftarrow \text{MaskedLinear}(\mathbf{h}) \\
&\mathbf{z}' \leftarrow \sigma(\mathbf{s}) \odot \mathbf{z} + (1 - \sigma(\mathbf{s})) \odot \boldsymbol{\mu}.
\end{aligned} \tag{5.31}$$

Here, $\sigma(\ )$ denotes the sigmoid activation function. In Kingma et al. (2016) it was mentioned that the gated form of IAF in Eq. (5.31) is more stable than the form of Eq. (5.29). Note that the size of $\mathbf{h}_{\text{context}}(\mathbf{x})$ scales with the width of the MADE layers C.

[1]This implementation is based on the open source code for IAF available at `https://github.com/openai/iaf`

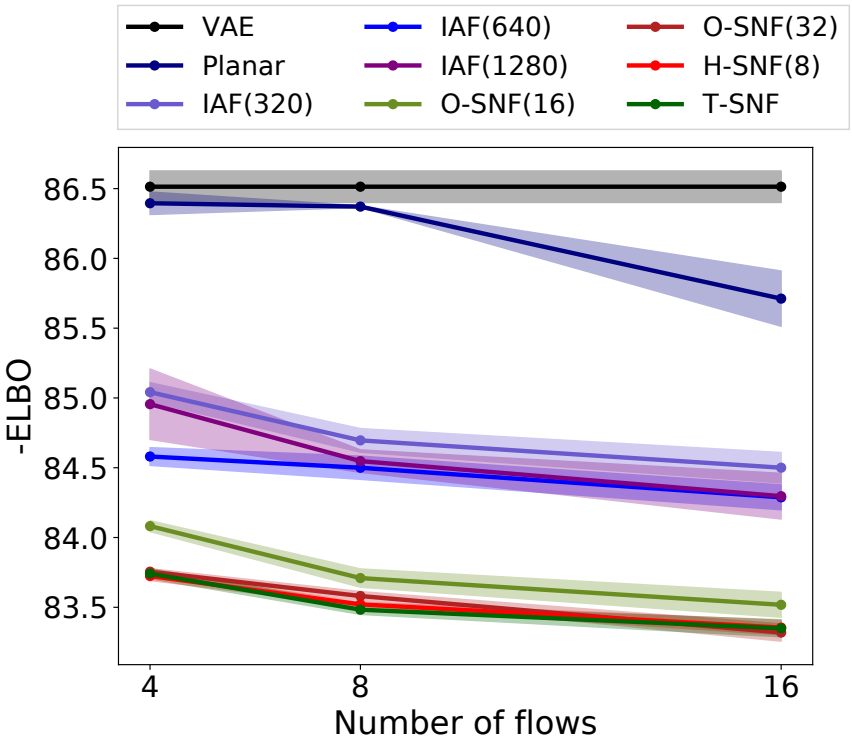


Figure 5.3: The negative evidence lower bound for static MNIST. The results for H-SNF with 4 reflections per orthogonal matrix are left out for clarity, as they are very similar to the results with 8 reflections. Each model is evaluated 3 times. The shaded areas indicate $\pm$ one standard deviation.

Table 5.1: Negative log-likelihood and free energy (negative evidence lower bound) for static MNIST. Numbers are produced with 3 runs per model with different random initializations. Standard deviations over the 3 different runs are also shown.

Model	**-ELBO**	**NLL**
VAE	86.55 ± 0.06	82.14 ± 0.07
Planar	86.06 ± 0.31	81.91 ± 0.22
IAF	84.20 ± 0.17	80.79 ± 0.12
O-SNF	$\mathbf{83.32 \pm 0.06}$	$\mathbf{80.22 \pm 0.03}$
H-SNF	83.40 ± 0.01	80.29 ± 0.02
T-SNF	83.40 ± 0.10	80.28 ± 0.06

5.6.1 MNIST

Figure 5.3 shows the dependence of the negative evidence lower bound (or free energy) on the number of flows and the type of flow for static MNIST. The exact numbers corresponding to the figure are shown in Section C in the appendix.

For all models the performance improves as a functions of the number of flows. For 4 flows the difference between the baseline VAE and planar flows is very small. However, planar flows clearly benefit from more flow transformations.

For IAF three different widths of the MADE layers were used: $C = 320$, 640 and 1280. Surprisingly, for 4 flows the widest IAF with 1280 hidden units is outperformed by an IAF with 640 hidden units in the MADE layers. We expect this to be due to the fact that this model has more parameters and can therefore be harder to train, as indicated by the larger standard deviation for this model.

All three Sylvester flows outperform IAF and planar flows. For Orthogonal Sylvester flows, we show results for $M = 16$ and $M = 32$ orthogonal vectors per orthogonal matrix, thus corresponding to bottlenecks of size 16 and 32 respectively for a latent space of size $D = 64$. Clearly, a larger bottleneck improves performance. For Householder Sylvester flows we experimented with $H = 4$ and $H = 8$ Householder reflections per orthogonal matrix. Since the results were nearly indistinguishable between these two variants, we have left out the curve for $H = 4$ to avoid clutter. O-SNF with $M = 32$, H-SNF and T-SNF seem to perform on par.

In Table 5.1, the negative evidence lower bound and the estimated negative log-likelihood are shown for the baseline VAE, together with all flow models for 16 flows. The reported result for IAF is for a MADE width of 1280. The O-SNF model has a bottleneck of $M = 32$, and H-SNF contains 8 Householder reflections per orthogonal

matrix. Again, all Sylvester flows outperform planar flows and IAF, both in terms of the free energy and the negative log-likelihood.

As discussed in Section 5.4, T-SNF is closely related to mean-only IAF, but with the MADE parameters directly amortized. The fact that T-SNF outperforms IAF indicates that amortizing the parameters directly leads to a more flexible transformation compared to taking a very wide MADE with a data dependent context as an additional input.

Table 5.2: Results for Freyfaces, Omniglot and Caltech 101 Silhouettes datasets. For the Freyfaces dataset the results are reported in bits per dim. For the other datasets the results are reported in nats. For each flow model 16 flows are used. For IAF a MADE width of 1280 was used, and for O-SNF flow a bottleneck of $M = 32$ was used. For H-SNF 8 householder reflections were used to construct orthogonal matrices. For all datasets 3 runs per model were performed.

Model	**Freyfaces**		**Omniglot**		**Caltech 101**	
	-ELBO	NLL	-ELBO	NLL	-ELBO	NLL
VAE	4.53 ± 0.02	4.40 ± 0.03	104.28 ± 0.39	97.25 ± 0.23	110.80 ± 0.46	99.62 ± 0.74
Planar	$\mathbf{4.40 \pm 0.06}$	$\mathbf{4.31 \pm 0.06}$	102.65 ± 0.42	96.04 ± 0.28	109.66 ± 0.42	98.53 ± 0.68
IAF	4.47 ± 0.05	4.38 ± 0.04	102.41 ± 0.04	96.08 ± 0.16	111.58 ± 0.38	99.92 ± 0.30
O-SNF	4.51 ± 0.04	4.39 ± 0.05	99.00 ± 0.29	93.82 ± 0.21	106.08 ± 0.39	94.61 ± 0.83
H-SNF	4.46 ± 0.05	4.35 ± 0.05	$\mathbf{99.00 \pm 0.04}$	$\mathbf{93.77 \pm 0.03}$	$\mathbf{104.62 \pm 0.29}$	$\mathbf{93.82 \pm 0.62}$
T-SNF	4.45 ± 0.04	4.35 ± 0.04	99.33 ± 0.23	93.97 ± 0.13	105.29 ± 0.64	94.92 ± 0.73

5.6.2 FreyFaces, Omniglot and Caltech 101 Silhouettes

We further assess the performance of the different models on Freyfaces, Omniglot and Caltech 101 Silhouettes. The results are shown in Table 5.2. The model settings are the same[2] as those used for Table 5.1.

Freyfaces is a very small dataset of around 2000 faces. All normalizing flows increase the performance, with planar flows yielding the best result, closely followed by Triangular and Householder Sylvester flows. We expect planar flows to perform the best in this case since it is the least sensitive to overfitting.

For Omniglot and Caltech 101 Silhouettes the results are clearer, with the Sylvester normalizing flows family resulting in the best performance. Both H-SNF and T-SNF perform better than O-SNF. This could be attributed to the fact that O-SNF has a bottleneck of $M = 32$ for a latent space size of $D = 64$. The IAF scores for Caltech 101 are surprisingly bad. We expect this could be the case due to the large number of parameters that need to be trained for IAF(1280). Therefore we also evaluated the result for MADEs of width 320 for 16 flows. The resulting free energy and estimated negative log-likelihood are 111.23 ± 0.45 and 99.74 ± 0.28 respectively, only slightly improving on the results of 1280 wide IAFs.

5.7 Conclusion

We present a new family of normalizing flows: Sylvester normalizing flows. These flows generalize planar flows, while maintaining an efficiently computable Jacobian determinant through the use of Sylvester's determinant identity. We ensure invertibility of the flows through the use of orthogonal and triangular parameter matrices. Three variants of Sylvester flows are investigated. First, orthogonal Sylvester flows use an iterative procedure to maintain orthogonality of parameter matrices. Second, Householder Sylvester flows use Householder reflections to construct orthogonal matrices. Third, triangular Sylvester flows alternate between fixed permutation and identity matrices for the orthogonal matrices. We show that the triangular Sylvester flows are closely related to mean-only IAF, with directly amortized MADE parameters. While performing comparably with planar flows and IAF for the Freyfaces dataset, our proposed family of flows improve significantly upon planar flows and IAF on the three other datasets.

[2]For Caltech 101 Silhouettes we used 2000 importance samples for the estimation of the negative log-likelihood.

Acknowledgements We would like to thank Christos Louizos for useful discussions and helping with the implementation of inverse autoregressive flows. LH is funded by the UK EPSRC OxWaSP CDT through grant EP/L016710/1. JMT is funded by the European Commission within the MSC- IF (Grant No. 702666). RvdB is funded by SAP SE.

5.8 Supplementary Material

5.8.1 Architecture

In the experiments we used convolutional layers for both the encoder and the decoder. Moreover, we used the gated activation function for convolutional layers:

$$\mathbf{h}_l = (\mathbf{W}_l * \mathbf{h}_{l-1} + \mathbf{b}_l) \odot \sigma(\mathbf{V}_l * \mathbf{h}_{l-1} + \mathbf{c}_l),$$

where $\mathbf{h}_{l-1}$ and $\mathbf{h}_l$ are inputs and outputs of the l-th layer, respectively, $\mathbf{W}_l, \mathbf{V}_l$ are weights of the l-th layer, $\mathbf{b}_l, \mathbf{c}_l$ denote biases, $*$ is the convolution operator, and $\sigma(\cdot)$ is the sigmoid activation function.

We used the following architecture of the encoder (k is a kernel size, p is a padding size, and s is a stride size):[3]

$$\begin{aligned}
&\text{Conv}(\text{in} = 1, \text{out} = 32, \text{k} = 5, \text{p} = 2, \text{s} = 1)\\
&\text{Conv}(\text{in} = 32, \text{out} = 32, \text{k} = 5, \text{p} = 2, \text{s} = 2)\\
&\text{Conv}(\text{in} = 32, \text{out} = 64, \text{k} = 5, \text{p} = 2, \text{s} = 1)\\
&\text{Conv}(\text{in} = 64, \text{out} = 64, \text{k} = 5, \text{p} = 2, \text{s} = 2)\\
&\text{Conv}(\text{in} = 64, \text{out} = 64, \text{k} = 5, \text{p} = 2, \text{s} = 1)\\
&\text{Conv}(\text{in} = 64, \text{out} = 64, \text{k} = 5, \text{p} = 2, \text{s} = 1)\\
&\text{Conv}(\text{in} = 64, \text{out} = 256, \text{k} = 7, \text{p} = 0, \text{s} = 1)
\end{aligned}$$

Notice the last layer acts as a fully-connected layer. Eventually, fully-connected linear layers were used to parameterized diagonal Gaussian distribution and amortized parameters of a flow.

The decoder mirrors the structure of the encoder with transposed convolutional

[3]We use a PyTorch convention of defining convolutional layers.

layers (op is an outer padding):

$$\begin{aligned}
&\text{ConvT}(\text{in} = 64, \text{out} = 64, \text{k} = 7, \text{p} = 0, \text{s} = 1)\\
&\text{ConvT}(\text{in} = 64, \text{out} = 64, \text{k} = 5, \text{p} = 2, \text{s} = 1)\\
&\text{ConvT}(\text{in} = 64, \text{out} = 32, \text{k} = 5, \text{p} = 2, \text{s} = 2, \text{op} = 1)\\
&\text{ConvT}(\text{in} = 32, \text{out} = 32, \text{k} = 5, \text{p} = 2, \text{s} = 1)\\
&\text{ConvT}(\text{in} = 32, \text{out} = 32, \text{k} = 5, \text{p} = 2, \text{s} = 2, \text{op} = 1)\\
&\text{ConvT}(\text{in} = 32, \text{out} = 32, \text{k} = 5, \text{p} = 2, \text{s} = 1)\\
&\text{ConvT}(\text{in} = 32, \text{out} = 1, \text{k} = 1, \text{p} = 0, \text{s} = 1)
\end{aligned}$$

5.8.2 Description of datasets

In the experimetns we used the following four image datasets: static MNIST[4], OMNIGLOT[5], Caltech 101 Silhouettes[6], and Frey Faces[7]. Frey Faces contains images of size 28×20 and all other datasets contain 28×28 images.

MNIST consists of hand-written digits split into 60,000 training datapoints and 10,000 test sample points. In order to perform model selection we put aside 10,000 images from the training set.

OMNIGLOT is a dataset containing 1,623 hand-written characters from 50 various alphabets. Each character is represented by about 20 images that makes the problem very challenging. The dataset is split into 24,345 training datapoints and 8,070 test images. We randomly pick 1,345 training examples for validation. During training we applied dynamic binarization of data similarly to dynamic MNIST.

Caltech 101 Silhouettes contains images representing silhouettes of 101 object classes. Each image is a filled, black polygon of an object on a white background. There are 4,100 training images, 2,264 validation datapoints and 2,307 test examples. The dataset is characterized by a small training sample size and many classes that makes the learning problem ambitious.

Frey Faces is a dataset of faces of a one person with different emotional expressions. The dataset consists of nearly 2,000 gray-scaled images. We randomly split them into 1,565 training images, 200 validation images and 200 test images. We repeated the experiment 3 times.

[4] `http://yann.lecun.com/exdb/mnist/`

[5] `https://github.com/yburda/iwae/blob/master/datasets/OMNIGLOT/chardata.mat`.

[6] `https://people.cs.umass.edu/~marlin/data/caltech101_silhouettes_28_split1.mat`.

[7] `http://www.cs.nyu.edu/~roweis/data/frey_rawface.mat`

5.8.3 MNIST experiments

The exact numbers for the evidence lower bound as shown in Fig. 3 are listed in Table 5.3.

Table 5.3: Negative evidence lower bounds for the test set on MNIST. All results are obtained with stochastic hidden units of size 64.

Model	**-ELBO**
VAE	86.51 ± 0.11
Planar ($K = 4$)	86.40 ± 0.08
Planar ($K = 8$)	86.37 ± 0.006
Planar ($K = 16$)	85.71 ± 0.20
IAF ($W = 320$, $K = 4$)	85.04 ± 0.07
IAF ($W = 320$, $K = 8$)	84.70 ± 0.08
IAF ($W = 320$, $K = 16$)	84.50 ± 0.11
IAF ($W = 640$, $K = 4$)	84.58 ± 0.06
IAF ($W = 640$, $K = 8$)	84.50 ± 0.08
IAF ($W = 640$, $K = 16$)	84.29 ± 0.09
IAF ($W = 1280$, $K = 4$)	84.96 ± 0.25
IAF ($W = 1280$, $K = 8$)	84.55 ± 0.08
IAF ($W = 1280$, $K = 16$)	84.30 ± 0.16
O-SNF ($M = 16$, $K = 4$)	84.08 ± 0.04
O-SNF ($M = 16$, $K = 8$)	83.71 ± 0.07
O-SNF ($M = 16$, $K = 16$)	83.52 ± 0.09
O-SNF ($M = 32$, $K = 4$)	83.76 ± 0.02
O-SNF ($M = 32$, $K = 8$)	83.58 ± 0.03
O-SNF ($M = 32$, $K = 16$)	83.32 ± 0.06
H-SNF ($K = 4$, $H = 4$)	83.73 ± 0.05
H-SNF ($K = 8$, $H = 4$)	83.49 ± 0.08
H-SNF ($K = 16$, $H = 4$)	83.36 ± 0.04
H-SNF ($K = 4$, $H = 8$)	83.72 ± 0.03
H-SNF ($K = 8$, $H = 8$)	83.52 ± 0.01
H-SNF ($K = 16$, $H = 8$)	83.35 ± 0.05
T-SNF ($K = 4$)	83.74 ± 0.04
T-SNF ($K = 8$)	83.48 ± 0.03
T-SNF ($K = 16$)	83.35 ± 0.06

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	Sylvester Normalizing Flows for Variational Inference
Publication Status	xPublished □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work
Publication Details	Rianne van den Berg*, Leonard Hasenclever*, Jakub M. Tomczak, Max Welling, Sylvester Normalizing Flows for Variational Inference, in *International Conference on Uncertainty in Artificial Intelligence, Oral Presentation*, 2018.

Student Confirmation

Student Name:	Leonard Hasenclever	
Contribution to the Paper	Sylvester Normalizing Flows are based on my idea and I contributed the theory and wrote most of the paper. I contributed to the experiments and analysis. Most of the implementation and experiments were run by Rianne van den Berg.	
Signature	Date	

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: Professor Yee Whye Teh		
Supervisor comments		
Signature	Date	

Part II

Applications to Continuous Control

Chapter 6

A Brief Review of Reinforcement Learning

The first part of this thesis studied both classification problems and generative models. Classification is an example of *supervised learning*, where given a set of inputs and outputs the task is to model the input-output mapping. Generative models are an example of *unsupervised learning* where we model the generative process that leads to a certain data distribution. Both supervised and unsupervised learning share one important property. In both cases the data distribution is independent of the model.

Most learning problems in the real world are not of this form. A toddler does not study a large set of human motions once in order to learn how to walk. Instead, toddlers take more than 2000 steps an hour, falling 17 times on average (Adolph et al., 2012), slowly learning by trial and error. Crucially, a toddler acts in the world learning from successes and failures. The distribution of "data" is not static, but directly affected by the toddler's behaviour. More generally, many problems involve interactions with an environment that is affected by the actions taken, and sequential decision making. In fact, it is difficult to think of problems that cannot be cast in this framework. *Reinforcement learning (RL)* is the study of such problems involving sequential decision making.

Because it studies such a large class of problems, RL is a field with many different problem settings, subfields and applications and it would be beyond the scope of this thesis to offer a comprehensive review. For an excellent introduction, we refer the reader to Sutton and Barto (1998). Szepesvári (2010) provides a slightly more technical treatment. In the rest of this chapter we will briefly review reinforcement learning and introduce background material relevant to the contributions in chapters 7 and 8.

6.1 A Brief Introduction to Reinforcement Learning

Reinforcement learning addresses the problem of learning to achieve a goal by interacting with an environment based on a numerical feedback or reward signal. A simple example of reinforcement learning is playing the classic ATARI game of space invaders in which the player controls a movable laser canon at the bottom of the screen. The player gets points for shooting down incoming aliens. For an example from a completely different domain, consider a robot in a warehouse, tasked with picking up and moving boxes. Here the robot could receive rewards for moving towards the right box and picking it up successfully, while being penalised for picking up the wrong items or falling over.

These two examples show that RL spans a large range of different problems. Indeed, they differ in important aspects:

Continuous vs. discrete actions Robot actions are continuous, whereas in ATARI games, a player has only a small number of discrete joy stick commands at their disposal.

Deterministic vs. stochastic environment The robot environment is fairly deterministic, albeit difficult to model. However, in ATARI games, the times at which new opponents or objects enter the screen can be random.

However, there are a few common characteristics of reinforcement learning that make it challenging.

Reward function not directly available Rather than being told explicitly which actions are good, the agent has to discover rewarding actions by trying them. The reward is not available as a differentiable function, but can only be observed through interactions with the environment.

Credit assignment problem Actions taken by the agent can affect the dynamics of the environment and through it future rewards. In addition, the dynamics of the environment are unknown and cannot be used directly for planning. This creates a credit-assignment problem for the agent: A reward received at a certain time t may in fact depend on actions taken a number of steps earlier, rather than on the last action.

Exploration problem Agents can only adapt to rewards they have observed. This creates a problem of exploration. In order to avoid suboptimal solutions, learning agents have to balance exploration of new but potentially suboptimal actions and strategies with exploitation of its current knowledge.

6.1.1 Formalising Reinforcement Learning

Reinforcement learning is typically formalised as an Markov Decision Process (MDP). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, p(s_0), p(r, s'|s, a))$. $\mathcal{S}$ is called the state space, the set of possible states of the environment. $\mathcal{A}$ is the set of actions available to the agent. The initial distribution of states is given by $p(s_0)$. The transition model or transition dynamics $p(r, s'|s, a)$ models the resulting reward r and next state s', given a state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$. Often the reward $r(s, a)$ is deterministic. An MDP is Markovian in the sense that the reward and next state only depend on the previous state and action. There are a number of variations of this definition; perhaps most importantly, the MDP can be partially observed, i.e. instead of observing the full state s of the environment, the agent merely receives (possibly noisy) observations o that depend on s. This is called a partially-observed Markov Decision Process (POMDP).

An MDP can be episodic, i.e. there exist absorbing or terminal states that are impossible to leave. An episode describes one chunk of environment interactions before encountering a terminal state. In the example of space invaders, an episode ends when the player loses. In the case of a warehouse robot, episode terminations are less clear. However, the episode could end if the robot falls (indicating failure) or when it delivers the box to the correct location.

The agent acts according to a policy $\pi(\cdot|s)$ that specifies a probability distribution over actions given the state s.[1] Let Π be the set of all possible policies. A policy π together with the transition model $\mathcal{P}$ induces a distribution $\mathcal{P}^\pi$ over trajectories $\{s_0, a_0, r_0, s_1, a_1, r_1, \ldots\}$. The goal of reinforcement learning is typically to find the policy π that maximises the sum of expected rewards. This sum of rewards is known

[1] It is not immediately obvious why it is beneficial to consider stochastic policies. Indeed, for any MDP there is a deterministic optimal policy. This is not true for POMDPs and, more generally, stochastic policies help with exploration during learning. In addition, stochastic policies naturally encourage a certain robustness: if we can achieve our goal even under some amount of randomness, we may be less susceptible to small environment variations. Stochastic policies also yield interesting connections with probabilistic inference (Levine, 2018).

as the return. In an episodic MDP the RL problem can be written as:

$$\max_{\pi \in \Pi} \mathbb{E}_{\mathcal{P}^\pi} \left[\sum_{t=0}^{T} r_t(s_t, a_t) \right], \tag{6.1}$$

where T denotes the (possibly random) number of time steps in an episode. In an non-episodic MDP, we can introduce a discount factor γ to make the problem well-posed because there are potentially infinitely many timesteps. The discount factor describes how much more we value a reward at the current timestep relative to rewards in the future. Alternatively, the discount factor can be interpreted as a probability of continuing. The corresponding problem is given by:

$$\max_{\pi \in \Pi} \mathbb{E}_{\mathcal{P}^\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \right] \tag{6.2}$$

In practice, discount factors, though not strictly necessary, are often also used for episodic MDPs. Often, a small entropy term is added to the objective:

$$\max_{\pi \in \Pi} \mathbb{E}_{\mathcal{P}^\pi} \left[\sum_{t=0}^{\infty} \gamma^t \left(r_t(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right) \right]. \tag{6.3}$$

Here, the policy is encouraged to solve the problem specified by the reward while acting as randomly as possible. The role of the entropy term is to encourage exploration and avoid premature convergence to suboptimal solutions. As discussed in Ziebart (2010) and Haarnoja et al. (2018b), entropy regularisation encourage more robust solutions and this objective has deep connections with probabilistic inference (Levine, 2018).

Having introduced MDPs, we can now briefly discuss other important concepts in reinforcement learning. Firstly, we can reason about the value function $V^\pi(s)$ of a policy, i.e. the expected return following a policy π starting in state s:

$$V^\pi(s) = \mathbb{E}_{\mathcal{P}^\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \middle| s_0 = s \right] \tag{6.4}$$

While the value function allows evaluation and comparison of different policies, it does not directly offer a way to improve a policy. Another important concept is the action-value function of a policy. It is the expected return, starting in state s and executing action a and then following policy π:

$$Q^\pi(s, a) = \mathbb{E}_{\mathcal{P}^\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \middle| s_0 = s, a_0 = a \right] \tag{6.5}$$

Given an action-value function Q^π, we can define another policy π' that acts greedily with respect to Q^π, i.e. in every state s choose the action a that maximises $Q^\pi(s,a)$. It is easy to see that this greedy policy is at least as good as π. This is known as *policy improvement*. In the case of deterministic dynamics, Q^π and V^π are related by:

$$Q^\pi(s,a) = r(s,a) + \gamma V^\pi(s'), \tag{6.6}$$

where s' is the unique next state following action a in state s. Similarly:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}\left[r(s,a)\right] + \gamma V^\pi(s') \tag{6.7}$$

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{a'\sim\pi}\left[Q^\pi(s',a')\right] \tag{6.8}$$

These equations are fixed point equations for Q^π and V^π and known as Bellman equations. Associated with these equations are the Bellman operators T_Q^π and T_V^π:

$$T_V^\pi V(s) = \mathbb{E}_{a\sim\pi}\left[r(s,a)\right] + \gamma V(s') \tag{6.9}$$

$$T_Q^\pi Q(s,a) = r(s,a) + \gamma \mathbb{E}_{a'\sim\pi}\left[Q(s',a')\right] \tag{6.10}$$

It can be shown that T_Q^π and T_V^π are maximum norm contractions and, thanks to the Banach fixed-point theorem, that Q^π and V^π are the unique fixed points (Szepesvári, 2010). Repeatedly applying the Bellman operators offers a strategy to compute value functions for a policy π. This procedure is called *policy evaluation*. Iterating between policy evaluation and policy improvement yields the *policy iteration* algorithm.

We can also reason about the optimal value function $V^\star(s)$, which is the maximum expected return starting in a state s among all possible policies π.

Similar to Q^π, we can define $Q^\star$, the optimal action-value function. $Q^\star(s,a)$ is the maximum expected return among all policies, starting in state s and first performing action a. It is easy to see that, in the case of deterministic dynamics:

$$V^\star(s) = \sup_{a\in\mathcal{A}} Q^\star(s,a) \tag{6.11}$$

$$Q^\star(s,a) = r(s,a) + \gamma V^\star(s') \tag{6.12}$$

The optimal policy $\pi^\star$ acts greedily w.r.t. $Q^\star$, choosing the action that maximises $Q^\star(s,a)$ in every state. Thus, if we know $Q^\star$, we can derive the optimal policy. However, computing $Q^\star$ or $V^\star$ is non-trivial. Note that:

$$V^\star(s) = \sup_{a\in\mathcal{A}}\{r(s,a) + \gamma V^\star(s')\} =: T_V^\star V^\star(s) \tag{6.13}$$

$$Q^\star(s,a) = r(s,a) + \gamma \sup_{a'\in\mathcal{A}} Q^\star(s',a') =: T_Q^\star Q^\star(s,a). \tag{6.14}$$

The operators $T_V^\star$ and $T_Q^\star$ are not linear due to the supremum, but it can be shown that for $0 < \gamma < 1$, they are maximum norm contractions with unique fixed points $V^\star$ and $Q^\star$ (Szepesvári, 2010).

This fact gives rise to another classic reinforcement learning algorithm called *value iteration*. Value iteration proceeds by iteratively applying $T_Q^\star$ to Q or $T_V^\star$ to V, which leads to geometric convergence to $V^\star$ and $Q^\star$. At convergence, we can then use $Q^\star$ to read off the optimal policy $\pi^\star$.

While the Bellman equations above elucidate the structure of the RL problem and offer a way to construct (optimal) value functions and (optimal) action-value functions given infinite data to estimate the expectation, they do not immediately suggest a practical algorithm. Generally, algorithms for learning value functions maintain an approximation that is updated iteratively based on a Bellman equation. For instance, in the tabular case, in which we have a small number of states, the classic Q-learning algorithm (Sutton and Barto, 1998) implements the following value iteration update:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) \right) \tag{6.15}$$

This algorithm maintains a separate approximation for each state-action pair and relies on (6.14). It updates $Q(s_t, a_t)$ in the direction of the "target" value $r_t + \gamma \max_a Q(s_{t+1}, a)$, since we know that the two values should be equal at convergence.

For RL problems with a large number of states or continuous states, approximating the value or action-value functions for each state or state-action pair separately is impractical. Function approximators are used to approximate the functions instead. The famous Deep Q-Learning algorithm (Mnih et al., 2015), which first demonstrated human level performance on ATARI games, uses the same underlying Bellman equation (6.14). It uses a neural network to parametrise the action-value function $Q_\theta(s, a)$ and, in each iteration, takes gradient steps w.r.t. the following loss:

$$\mathbb{E}_{s,r,s'} \left[\left(Q_\theta(s, a) - (r_t + \gamma \max_a Q_{\theta_{\text{target}}}(s_{t+1}, a)) \right)^2 \right]. \tag{6.16}$$

Crucial in this algorithm is the use of target networks, i.e. forming the "target" value $r_t + \gamma \max_a Q_{\theta_{\text{target}}}(s_{t+1}, a)$ using a separate copy of the Q network with stale parameters. This allows the target values to be more robust to the influence of noisy updates to the action-value function. The parameters of the target network are only updated every once in a while, empirically leading to much more stable training.

Many reinforcement learning algorithms are based on generalisations of policy iteration or value iteration and make use of such Bellman updates. Note that the

Bellman backup presented here is the 1-step Bellman backup, but instead of relying on one timestep, one can also consider n-step Bellman backups, which reduce the bias arising from approximating the value of the end state with our current value function, but increases the variance due to the stochasticity of the n-step rollout.

Reinforcement learning algorithms can be *on-policy*, i.e. the data used to update the policy and the value function was collected using the current policy, or *off-policy*, i.e. the data used to update the policy was collected using other policies. If we use off-policy data, we have to correct for this mismatch to ensure that our Bellman update equations remain valid. Many such off-policy corrections exist (Precup et al., 2000, Harutyunyan et al., 2016, Munos et al., 2016).

In the following algorithms, we use the Retrace off-policy correction (Munos et al., 2016) for Q-learning. This is a standard algorithmic component although the details are somewhat technical. The exact mechanism of learning Q is orthogonal to the contributions in this chapter. For clarity, we simple denote the Retrace target values as $\hat{Q}^R = Retrace(\tau, r, Q^T, b, \pi)$, where τ denotes a partial trajectory, r denotes the associated rewards, Q^T denotes our Q target network, b denotes the behaviour policy that was used to collect τ and π denotes the current policy. Retrace, as well as other off-policy corrections, require $b(a_t|s_t)$ and $\pi(a_t|s_t)$, the probabilities of the action a_t under the behaviour policy and under the current policy, to form importance weights used in the correction.

In the following, we primarily focus on continuous control, which is characterised by continuous actions. This allows us to improve a policy by differentiating with respect to the action, an approach taken by stochastic value gradients (Heess et al., 2015). The SVG(0) version of the algorithm is a state-of-the-art algorithm in continuous control. We use SVG(0) with the Retrace off-policy correction as the base algorithm the contributions described in 7 and briefly describe it in the next section.

6.1.2 SVG(0) with Retrace

Given a stochastic policy π and a corresponding action-value function Q^π, we can improve π by taking optimizing:

$$\mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a)\right]. \tag{6.17}$$

This can be seen as an incomplete policy improvement step.

Instead of iteratively improving π and computing its action-value function Q^π, the SVG(0) version of stochastic value gradients (Heess et al., 2015) proposes to learn π and Q jointly and parametrises them with neural networks. For deterministic

environments and Gaussian policies we can use the reparametrisation trick (Kingma and Welling, 2013, Rezende et al., 2014) to calculate gradients with respect to (6.17) (see section 2.3.2).

SVG(0) is typically used in a distributed setting where many actors continuously execute the current policy and write the resulting trajectories to a replay buffer. The size of the replay buffer is fixed at a certain size and old data are removed as new data become available. A learner node then samples partial trajectories from the replay buffer and updates the policy and action-value function. We use Retrace (Munos et al., 2016) to learn the action-value function and use target networks. See algorithm 9.

Algorithm 9 SVG(0) with Retrace

online and target policy: π^O and π^T, with parameters θ_O and θ_T, respectively.
online and target Q-function: Q^O and Q^T, with parameters ψ_O and ψ_T, respectively.
target update period: P
replay buffer: $\mathcal{B}$
unroll length: K

Actor loop ▷ Many actors in parallel
while not converged **do**
  Synchronise with learner to get newest online policy π^O
  Unroll environment for K steps and collect trajectory
  $\tau = (s_1, a_1, \pi^O(a_1|s_1), r_1, \dots r_K)$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{\tau\}$
end while

Learner loop
while not converged **do**
  $\tau = (s_1, a_1, b(a_1|s_1), r_1, \dots r_K)$ ▷ Sample from replay buffer
  $\hat{Q}^R = Retrace(\tau, r, Q^T, b, \pi_T)$ ▷ Compute Retrace Q targets

  $\hat{L}_Q = \sum_{t=1}^{K} \|\hat{Q}_t^R - Q^O(s_t, a_t)\|^2$ ▷ Q loss
  $\hat{L}_\pi = \sum_{t=1}^{K} \mathbb{E}_{\pi^O(\cdot|s_t)}[Q^T(s_t, a)]$ ▷ Policy loss

  $\theta_O \leftarrow \theta_O + \beta_\pi \nabla_\theta \hat{L}_\pi$ ▷ Take gradient steps
  $\psi_O \leftarrow \psi_O - \beta_Q \nabla_\psi \hat{L}_Q$

  if $j \mod P = 0$ **then** ▷ update target networks
    $\theta_T \leftarrow \theta_O$
    $\psi_T \leftarrow \psi_O$
  end if
end while

Chapter 7

Priors in Reinforcement Learning

This chapter contains results from

> Galashov, A., Jayakumar, S., **Hasenclever, L.**, Tirumala, D., Schwarz, J., Desjardins, G., Czarnecki, W., Teh, Y.W., Pascanu, R. and Heess, N., 2019. Information Asymmetry in KL-regularized RL. International Conference on Learning Representations.

We present some key results here. Most of the final experiments presented in the paper were conducted by Alexandre Galashov. I was heavily involved in designing the algorithm, the implementation, and exploratory experiments.

Abstract

The solutions to related real-world tasks often share structure, that is repeated in both state space and time. For example, walking behaviour is useful in almost all continuous control tasks with complex bodies, regardless of the specific task objective. In this chapter, we consider learning priors in reinforcement learning to leverage such structure and enable both faster learning and and transfer to new tasks. We describe the KL-regularized RL objective and how it relates to the broader literature. Further, we discuss why learning priors is useful and outline different settings for learning priors. In addition, we show how to learn more flexible priors such as priors with explicit latent variables or LSTMs. We demonstrate the efficacy of our approach on a range of continuous control tasks with rich task structure.

7.1 Entropy and KL-regularized Reinforment Learning

A common objective in reinforcement learning is the entropy-regularized objective:

$$\max_{\pi \in \Pi} \mathbb{E}_{\mathcal{P}^\pi} \left[\sum_{t=0}^{\infty} \gamma^t \left(r_t(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right) \right]. \tag{7.1}$$

For bounded action spaces, this objective is equivalent to regularizing the action space towards a uniform distribution. In this chapter, we consider a related objective (known as the KL-regularized objective) that explicitly introduces a prior distribution $\pi_0(\cdot|s)$:

$$\max_{\pi \in \Pi} \mathbb{E}_{\mathcal{P}^\pi} \left[\sum_{t=1}^{T} \gamma^t \left(r_t(a_t, s_t) - \alpha \mathsf{KL}(\pi(\cdot|s_t) \| \pi_0(\cdot|s_t)) \right) \right]. \tag{7.2}$$

Note that we can interpret this objective as the classical RL objective with an auxiliary reward term based on the prior. If rewards are sparse, this term will encourage the policy to behave like the prior. Since rewards effectively correspond to observations in RL as inference, a dense reward setting can be thought of as a setting with more data. The rewards themselves constrain the distribution of rewarding trajectories more. Thus, if rewards are given at every step, the effect of the prior will generally be smaller. The auxiliary reward interpretation also means that it is easy to use off-the-shelf RL algorithms to implement this objective by simply augmenting the rewards.

For uniform prior distributions, (7.2) reduces to the entropy-regularised objective (7.1), which has been studied extensively, initially in robotics and control (e.g. Toussaint (2009), Ziebart (2010), Kappen et al. (2012)) and more recently in the deep reinforcement learning community (Fox et al., 2015, Schulman et al., 2017, Haarnoja et al., 2017, 2018b, Hausman et al., 2018, Nachum et al., 2017).

A related class of algorithms are expectation maximisation policy search algorithms (see Peters et al. (2010), Rawlik et al. (2012), Levine and Koltun (2013), Abdolmaleki et al. (2018). They proceed in a variational EM style fashion although the exact details differ from paper to paper, and repeatedly replace the prior with a learned policy.

In section 7.2, we discuss why informative, learned priors might be beneficial in RL. Section 7.3 reviews Distral (Teh et al., 2017), an algorithm for multi-task reinforcement learning that leverages the probabilistic perspective to allow transfer

learning through a shared, learned prior. Finally, section 7.4 introduces an off-policy algorithm for learning priors in continuous control. This section is based on my contributions to a larger project on priors in continuous control culminating in Galashov et al. (2019). It presents key results from Galashov et al. (2019) as well as additional exploratory results using continuous latent variables in the prior.

7.2 Why learn priors?

Why might an informative prior over actions be desirable? Action spaces in reinforcement learning are somewhat arbitrary and can be easily reparametrised. Indeed, in some domains, such as DeepMind Lab (Beattie et al., 2016), a set of complex simulated 3D environments, many authors use small, hand-designed subsets of all available actions to introduce biases that enable faster exploration (Mnih et al., 2016, Espeholt et al., 2018) and, empirically, using larger action spaces leads to substantially slower training and worse final performance (Czarnecki et al., 2018). Similarly, in continuous control, the choice of parametrisation of actuators can lead to differences in task difficulty. For example, torque-controlled bodies tend to be harder to control than position-controlled ones. The specific parametrisation of the actuators can be thought of as a form of prior knowledge.

Continuous control environments are a particularly promising area for priors over actions, because the action spaces are high dimensional and most random actions lead to "motor babbling" not sensible behaviours, posing a substantial exploration problem. Consider humans solving physical tasks in the real world: in any given body configuration there are only a few movements that are plausible regardless of the specific task at hand. Learning about these default behaviours could potentially significantly speed up learning and improve exploration in complex control tasks.

Thus, having access to an appropriate, informative prior could potentially speed up learning in many reinforcement learning problems. However, it is not clear how to obtain such a prior. In some cases, it may be possible to hand-engineer a suitable prior (for example, in discrete action domain with relatively few actions). However, this approach will not scale to problems with large or high dimensional action spaces. Therefore, it is interesting to consider methods that can learn suitable, informative priors automatically.

In many domains, we are interested in a large number of related tasks. For example, we could be interested in a number of different 3D navigation problems (such as DeepMind Lab) or in solving diverse tasks with the same physical body in

continuous control. Here, we can use related tasks to learn informative priors. There is a whole spectrum of different, possible settings for learning priors, among them:

Learning from existing expert policies In the simplest scenario, we assume that we have access to expert policies on a number of related tasks. We can then easily learn the prior via supervised learning on trajectories of the task policies. This is known as distillation (or behavioural cloning) and the approach in chapter 8 can be seen as a specific instantiation of this idea. Using such a learned prior for transfer learning is similar to the kick-starting framework (Schmitt et al., 2018), which uses KL-regularisation towards an existing solution on the same task as a means to speed up experiments with new models.

Multitask learning If we do not have access to task solutions, we can try to learn task solution and a prior at the same time. In this set-up each task policy is regularised towards a shared prior. We can simultaneously learn the prior via distillation on trajectories of the task policies. This approach, which we review in section 7.3, is called Distill & Transfer Learning (Distral) and was first proposed by Teh et al. (2017).

Single task with information asymmetry Here, we consider learning priors on a single task at the same time as the task solution. Crucially, the prior receives less information than the task policy (e.g. only proprioceptive information, i.e. information about the internal state of the body, in continuous control tasks), allowing it to learn faster and encourages behaviour sharing across the state space. This allows the prior to act as a useful regulariser and speed up learning in certain settings. We proposed this approach in Galashov et al. (2019). We describe this approach below and present experimental results.

Figure 7.1 shows the relationship between prior and policy in this framework.

The rest of this chapter is structured as follows. The next section reviews Distral (Teh et al., 2017), a method for multi-task reinforcement learning that builds on the KL regularised objective in equation (7.2). Section 7.4 describes our work in Galashov et al. (2019) and shows how to augment the prior with continuous latent variables. Finally, section 7.5 presents key experimental results from Galashov et al. (2019) and additional exploratory results with continuous latent variables in the prior.

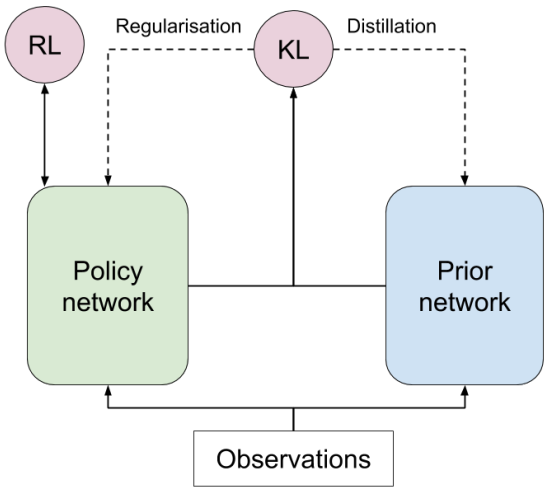


Figure 7.1: Figure adapted from Galashov et al. (2019) showing the relationship between policy and prior. The policy receives observations and produces actions that interact with the environment. The prior also receives (possibly different parts of the) observations and regularises the policy through a KL term. In this framework, the prior can either be fixed or learned at the same time as the policy via distillation.

7.3 Distill and Transfer Learning

Teh et al. (2017) apply the idea of learning informative priors in KL-regularised RL in the context of multi-task learning and propose learning a shared prior at the same time as learning to solve individual tasks. Their approach is called Distill & Transfer Learning (Distral). They consider k different tasks that share the state space $\mathcal{S}$ and action space $\mathcal{A}$, but differ in the reward function (and possibly transition dynamics). Let π_i be the policy for the i-th task and let π_0 denote the prior as before. The Distral objective is then given by:

$$J(\pi_0, \{\pi_i\}_{i=1}^k) = \sum_{i=1}^{k} \mathbb{E}_{\pi_i} \left[\sum_{t \geq 0} \gamma^t r_t(a_t, s_t) - c_{\mathrm{KL}} \gamma^t \log \frac{\pi_i(a_t|s_t)}{\pi_0(a_t|s_t)} - c_{\mathrm{ENT}} \gamma^t \log \pi_i(a_t|s_t) \right], \tag{7.3}$$

where γ is a discount factor and c_{KL} and c_{ENT} are coefficients trading off the strength of the regularisation through the prior and the entropy regularisation, respectively. This objective is closely related to (7.2), but includes an (optional) entropy term that may help the exploration. Distral jointly optimises the above objective. Note that only the second term in (7.3) depends on the prior π_0 and optimising with respect to π_0 amounts to distilling the task-specific policies into the prior. Optimising the objective with respect to the task-specific policies can be achieved through policy-gradient

methods such as advantage actor-critic. Teh et al. (2017) use the asynchronous advantage actor-critic (A3C, Mnih et al. (2016)) for their experiments.

Teh et al. (2017) present extensive experiments in complex, simulated 3D environments with discrete actions (Beattie et al., 2016). They find that learning a shared prior leads to substantially more robust and stable learning behaviours across hyperparameter settings, and often speeds up the learning process and yields better final performance. A related approach is Divide and Conquer Reinforcement Learning (Ghosh et al., 2018). In Divide and Conquer RL, different policies specialise in different contexts (such as different parts of the initial state distribution) and are regularised towards one another with a symmetric KL term. All policies are then distilled in to a single policy.

7.4 Learning Priors in Continuous Control

The idea of learning informative priors using the objective in equation (7.2) or (7.3) is largely orthogonal to the specific algorithm used to optimise said objective. In this section we first present an extension of SVG(0) (see section 6.1.1) that is able to learn priors. We then show how the prior can be augmented with latent variables and present experimental results.

7.4.1 Learning priors within SVG(0)

Augmenting SVG(0) with a KL-regularisation is relatively straightforward. The KL regularisation in (7.2) and (7.3) simply acts like an auxiliary reward term (see algorithm 10). To use this algorithm in a multi-task setting, we can apply the same algorithm to each task, but use a shared prior and prior update.

7.4.2 Information Asymmetry between Policy and Prior

In Galashov et al. (2019) we study the effects of introducing an information asymmetry between policy and prior. Tasks in continuous control often have structured sets of observations. Typically there are proprioceptive observations, i.e. information about the state of the robot or walker itself and task observations that specifies the task at hand, such as observations from vision or a goal location.

Formally, let us assume that the observations s_t decompose into s_t^P, which the prior has access to, and s_t^G, which will be hidden from the prior. Thus, the prior learns about the average behaviour in states with the same s_t^P. Since in our setting

Algorithm 10 Learning a prior with SVG(0).

online and target policy: π^O and π^T, with parameters θ_O and θ_T, respectively.
online and target Q-function: Q^O and Q^T, with parameters ψ_O and ψ_T, respectively.
online and target prior policy: π_0^O and π_0^T, with parameters ϕ_O and ϕ_T, respectively.
target update period: P
replay buffer: $\mathcal{B}$
unroll length: K

Actor loop as in algorithm 9

Learner loop
while not converged **do**
$\tau = (s_1, a_1, b(a_1|s_1), r_1, \dots r_K)$ ▷ Sample from replay buffer
$\widehat{\mathsf{KL}}_t^O = \mathsf{KL}[\pi^O(\cdot|s_t) \| \pi_0^O(\cdot|s_t)]$ ▷ compute KL to online prior for prior loss
$\widehat{\mathsf{KL}}_t^T = \mathsf{KL}[\pi^O(\cdot|s_t) \| \pi_0^T(\cdot|s_t)]$ ▷ compute KL to target prior for Q loss
$\tilde{r}_t = r_t - \alpha \widehat{KL}_t^T$ ▷ Augment rewards
$\hat{Q}^R = Retrace(\tau, \tilde{r}, Q^T, b, \pi_T)$ ▷ Compute Retrace Q targets

$\hat{L}_Q = \sum_{t=1}^K \|\hat{Q}_t^R - Q^O(s_t, a_t)\|^2$ ▷ Q loss
$\hat{L}_\pi = \sum_{t=1}^K \mathbb{E}_{\pi^O(\cdot|s_t)}[Q^T(s_t, a)]$ ▷ Policy loss
$\hat{L}_{\pi_0} = \sum_{t=1}^K \widehat{\mathsf{KL}}_t^O$ ▷ Prior loss

$\theta_O \leftarrow \theta_O + \beta_\pi \nabla_\theta \hat{L}_\pi$ ▷ Take gradient steps with respect to the losses
$\phi_O \leftarrow \phi_O - \beta_{\pi^0} \nabla_\phi \hat{L}_{\pi^0}$
$\psi_O \leftarrow \psi_O - \beta_Q \nabla_\psi \hat{L}_Q$

if $j \mod P = 0$ **then** ▷ Update target networks
$\theta_T \leftarrow \theta_O$
$\phi_T \leftarrow \phi_O$
$\psi_T \leftarrow \psi_O$
end if
end while

the prior is learned alongside the policy, the effect of the regularisation through the prior is to encourage the policy to behave similarly in states with similar s_t^P. It does not a priori bias the behaviour towards a specific solution. This sharing of behaviours potentially allows for faster learning and better generalisation.

7.4.3 Flexible priors

An important question is which form a prior should take. Perhaps the simplest possible prior policy in continuous control is a diagonal Gaussian parametrised by a feed-forward neural network. However, in general, sensible behaviours are likely to be correlated through time. Similarly, it is possible that in a given state there are multiple sensible actions; this might be hard to capture with a single Gaussian. In the following section we will investigate two classes of more flexible priors: Long Short Term Memory networks (LSTMs, Hochreiter and Schmidhuber (1997)) and priors with continuous latent variables.

LSTMs are recurrent neural networks with a special gating mechanism that helps to mitigate the well-known vanishing gradient problem in training recurrent neural networks. LSTMs are straightforward to use as prior within the framework we have presented.

As another option, we consider priors with latent variables. Explicit latent variables may provide a better representation of behaviour for transfer. For the experiments in this section, we will focus on priors with one latent variable per time-step in which the latent variables are coupled through a prior following an autoregressive process. Of course, this is only one possible choice and we hope to explore other priors in future work. For example, we could use discrete latent variables to capture distinct behaviour modes or could use one latent variable for the whole length of the roll-out. We focus on this model choice because a very similar architecture proves very effective in modelling humanoid behaviours in chapter 8. More formally, we consider priors over trajectories that factorise as follows:

$$\pi_0(\tau, z_{1:T}) = p(s_1)\pi_0(z_1)\prod_{t=1}^{T}\left[\pi_0(a_t|s_t, z_t)p(s_{t+1}|s_t, a_t)\right]\prod_{t=1}^{T-1}\pi_0(z_{t+1}|z_t), \tag{7.4}$$

where $\pi_0(\cdot|s_t, z_t)$ is a Gaussian distribution parametrised by a neural network and z_t denotes the latent variable for time step t and follows an AR(1) process

$$z_0 \sim \mathcal{N}(0, I) \tag{7.5}$$

$$z_t = \alpha z_{t-1} + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{7.6}$$

Here $\alpha \in [0, 1]$ determines how strongly latent variables are coupled. The noise standard deviation $\sigma = \sqrt{1 - \alpha^2}$ is chosen such that marginally, $z_t \sim \mathcal{N}(0, I)$.

Recall the KL-regularised objective in (7.2). With this type of prior, the KL term does not decompose per time-step:

$$\mathsf{KL}(\pi_\theta(a_{1:T}|s_{1:T})\|\pi_0(a_{1:T}|s_{1:T})) = \mathbb{E}_{\pi_\theta}\left[\log \pi_\theta(a_{1:T}|s_{1:T}) - \log \pi_0(a_{1:T}|s_{1:T})\right], \tag{7.7}$$

since the marginal prior probability of a sequence of actions given states, is generally intractable. However, we can construct a lower bound to form an approximate KL estimate:

$$\log \pi_0(a_{1:T}|s_{1:T}) \geq \mathbb{E}_{f(z_{1:T}|\tau)}\left[\log \pi_0(a_{1:T}, z_{1:T}|s_{1:T}) - \log f(z_{1:T}|a_{1:T}, s_{1:T})\right], \tag{7.8}$$

where f denotes an inference network. Specifically, we investigate inference networks of the form $f(z_t|z_{t-1}, a_{t:t+k}, s_{t:t+k})$, where k is a small integer, effectively providing a future lookahead. With this choice of inference network the latent space can be thought of as an embedding space for short term motor intentions. A latent variable z_t corresponds to an intended sequence of states (and actions) in the next k steps and the autoregressive prior enforces a degree of temporal smoothness. Other conditioning sets for the inference network would result in different semantics of the latent space.

With this choice, the evidence lower bound can be written as:

$$\log \pi_0(a_{1:T}|s_{1:T}) \geq \mathbb{E}_{f(z_{1:T}|\tau)}\left[\sum_{t=1}^{T} \log \frac{\pi_0(a_t, z_t|s_t, z_{t-1})}{f(z_t|z_{t-1}, a_{t:t+k}, s_{t:t+k})}\right], \tag{7.9}$$

We can then use it to upper bound the KL term as follows:

$$\mathsf{KL}(\pi_\theta(a_{1:T}|s_{1:T})\|\pi_0(a_{1:T}|s_{1:T})) = \mathbb{E}_{\pi_\theta}\left[\log \frac{\pi_\theta(a_{1:T}|s_{1:T})}{\pi_0(a_{1:T}|s_{1:T})}\right] \tag{7.10}$$

$$\leq \mathbb{E}_{\pi_\theta}\left[\log \pi_\theta(a_{1:T}|s_{1:T}) - \mathbb{E}_{f(z_{1:T}|\tau)}\left[\sum_{t=1}^{T} \log \frac{\pi_0(a_t, z_t|s_t, z_{t-1})}{f(z_t|z_{t-1}, a_{t:t+k}, s_{t:t+k})}\right]\right] \tag{7.11}$$

$$= \mathbb{E}_{\pi_\theta}\left[\mathbb{E}_{f(z_{1:T}|\tau)}\left[\sum_{t=1}^{T} \underbrace{\log \frac{\pi_\theta(a_t|s_t) f(z_t|z_{t-1}, a_{t:t+k}, s_{t:t+k})}{\pi_0(a_t, z_t|s_t, z_{t-1})}}_{=:K_t}\right]\right] \tag{7.12}$$

Note, that, given a sample from the inference distribution, this bound decomposes per timestep. We can now modify algorithm 10 by replacing the per-timestep KL by the per timestep parts of (7.12), denoted K_t. See algorithm 11.

One aspect of this algorithm is worth commenting on: In algorithm 10 we use the analytic KL, whereas our approximation in algorithm 11 relies on sampled actions from the replay buffer. This introduces a bias, which could be corrected via importance weighting. Without the use of target networks, we found this additional importance weighting helped stabilise the algorithm. However, when target networks are used, the importance weighting did not result in meaningfully different results, and thus we omit a more detailed discussion here.

7.5 Experimental results

We evaluated the algorithm in a number of experiments in challenging continuous control experiments.

7.5.1 Information Asymmetry

In Galashov et al. (2019), we conducted an extensive experimental study of priors in continuous control using algorithm 10.

We considered a number of different tasks in continuous control with different characteristics, among them:

- A dense-reward walking task, in which the agent has to move forward, backward, left, or right at a fixed speed. The direction is randomly sampled at the beginning of an episode and changed to a different direction half-way through the episode.
- A sparse reward go-to-target task, in which the agent has to move to a target whose location is supplied to the agent as a feature vector.
- A sparse reward box-moving task, in which the agent has to push a box to one (of many) targets and optionally has to go to a different target location afterwards. The box and target positions are supplied to the agent as feature vectors.
- A sparse reward foraging task in a 3D maze. In this task, the agent has vision from an egocentric camera and has to collect apples in a maze.

Additional details about the tasks can be found in the supplementary material in section 7.7.1. These tasks vary in reward type as well as task complexity. A shared

Algorithm 11 Learning a prior with latent variables with SVG(0).

online and target policy: π^O and π^T, with parameters θ_O and θ_T, respectively.
online and target Q-function: Q^O and Q^T, with parameters ψ_O and ψ_T, respectively.
online and target prior policy: π_0^O and π_0^T, with parameters ϕ_O and ϕ_T, respectively.
online and target inference network: f^O and f^T, with parameters ω_O and ω_T, resp.
target update period: P
replay buffer: $\mathcal{B}$
unroll length: K

Actor loop as in algorithm 9

Learner loop
while not converged **do**
$\tau_{1:K} = (s_1, a_1, b(a_1|s_1), r_1, \dots r_K)$ ▷ Sample from replay buffer
$z^O_{1:T} \sim f^O(\cdot|\tau)$ and $z^T_{1:T} \sim f^T(\cdot|\tau)$ ▷ Unroll inference networks
$\widehat{\mathsf{KL}}^O_t = K_t(z^O_{1:t})$ ▷ compute KL approximation to online prior via Eq. (7.12)
$\widehat{\mathsf{KL}}^T_t = K_t(z^T_{1:t})$ ▷ compute KL approximation to target prior via Eq. (7.12)
$\tilde{r}_t = r_t - \alpha \widehat{KL}^T_t$ ▷ Augment rewards
$\hat{Q}^R = Retrace(\tau, \tilde{r}, Q^T, b, \pi_T)$ ▷ Compute Retrace Q targets

$\hat{L}_Q = \sum_{t=1}^K \|\hat{Q}^R_t - Q^O(s_t, a_t)\|^2$ ▷ Q loss
$\hat{L}_\pi = \sum_{t=1}^K \mathbb{E}_{\pi^O(\cdot|s_t)}[Q^T(s_t, a)]$ ▷ Policy loss
$\hat{L}_{\pi_0} = \sum_{t=1}^K \widehat{\mathsf{KL}}^O_t$ ▷ Prior loss

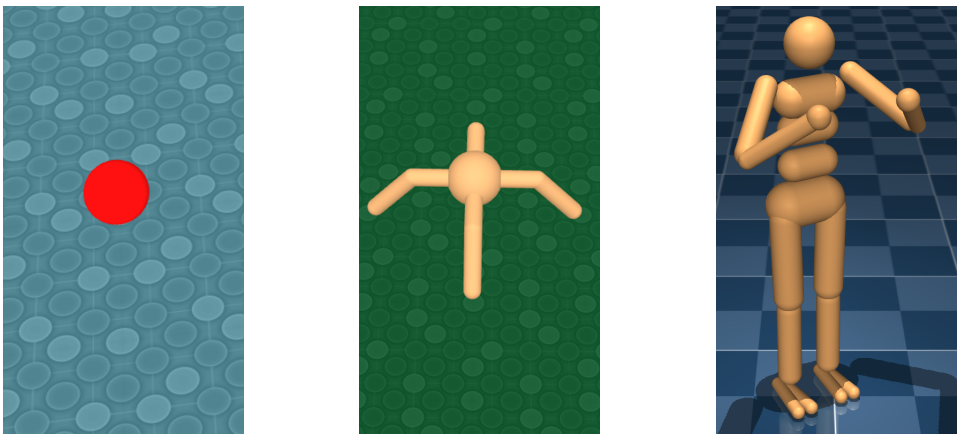
$\theta_O \leftarrow \theta_O + \beta_\pi \nabla_\theta \hat{L}_\pi$ ▷ Take gradient steps with respect to the losses
$\phi_O \leftarrow \phi_O - \beta_{\pi^0} \nabla_\phi \hat{L}_{\pi^0}$
$\omega_O \leftarrow \omega_O - \beta_{\pi^0} \nabla_\omega \hat{L}_{\pi^0}$
$\psi_O \leftarrow \psi_O - \beta_Q \nabla_\psi \hat{L}_Q$

if $j \mod P = 0$ **then** ▷ Update target networks
$\theta_T \leftarrow \theta_O$
$\phi_T \leftarrow \phi_O$
$\psi_T \leftarrow \psi_O$
$\omega_T \leftarrow \omega_O$
end if
end while

component of these tasks is that low-level motor skills are necessary for good performance and similar low level behaviours should be useful across different parts of the state space. For this reason, it seems plausible that an appropriate prior could regularize the learning process and speed up learning. We vary motor complexity by using three different walkers:

- a jumping ball with 3 degrees of freedom and 3 actuators,
- a quadruped (or ant) with 12 degrees of freedom and 8 actuators,
- and a simple humanoid body with 28 degrees of freedom and 21 actuator.

Figure 7.2 shows visualisations of the different walkers. In each of the tasks, the



(a) Jumping Ball (b) Ant (c) Humanoid

Figure 7.2: Walker visualisations reproduced from Galashov et al. (2019).

observations consist of proprioceptive information about the state of the walker body and task information. On the different task-walker combinations, we compare the effect of passing different sets of information to the prior policy. We compare to SVG(0) with entropy regularisation as a baseline.

In our experiments we found that learned priors substantially sped up learning for the more complex walkers (ant, humanoid) in sparse reward tasks. On the other hand, with the jumping ball, or in tasks with a dense reward, there was no consistent gain. Since dense rewards constrain the distribution of rewarding trajectories substantially more, the effect of a prior is smaller. In addition, solving tasks with the jumping ball is relatively straightforward, since there is no substantial exploration problem. Figure 7.3 shows the results on sparse reward tasks. The left panel shows the go to target task with the simple humanoid. In the middle, we show results on the

foraging task in the maze results with ant. On the right, we show result for the moving one box to one of two targets and go to another target task with ant. The legend shows the information passed to the prior policy in addition to proprioceptive information. All results were averaged over 5 seeds and the hyperparameters were tuned per task-walker combination. For further experimental details, see appendix 7.7.

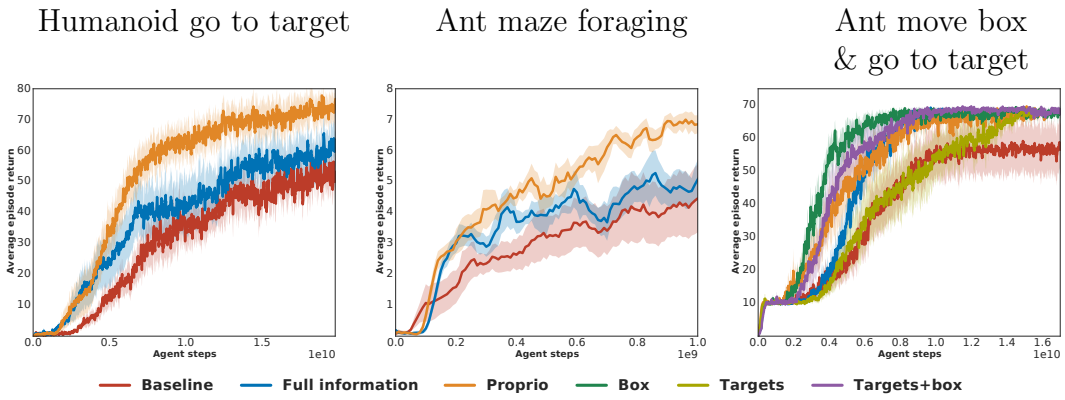


Figure 7.3: Figure reproduced from Galashov et al. (2019) showing results on sparse reward tasks.

7.5.2 Exploratory Results with more flexible Priors

For our experiments with latent variables, we used the simple humanoid and the walking task. We consider two separate scenarios: single-task learning, where the prior is learned at the same time as the policy, and transfer learning, where we first learn a prior and then transfer it to a new task that requires similar behaviours. As the new task, we consider the sparse go-to-target task described above. See appendix 7.7 for details on the environments as well as the network architectures and hyper parameters.

The results in this section are meant as exploratory results that show that it is possible to learn priors with latent variables and reuse them to speed up learning. This motivates the results (and architecture) in chapter 8.

We tune the hyperparameters separately for each prior type and used two-layer MLPs with about 300 hidden units per layer for all components. Additional experimental details can be found in appendix 7.7.

In a first experiment, we compare a simple feed-forward prior (multi layer perceptron, MLP), an LSTM and a latent variable model with one latent variable per

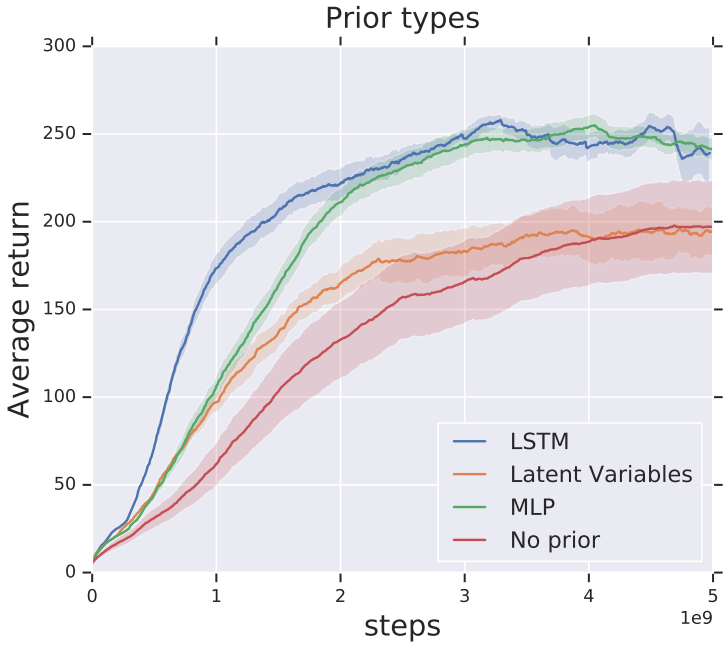


Figure 7.4: Results showing that priors speed up learning on the walking task. Average over 3 runs.

timestep. See figure 7.4. All priors studied in this comparison gave a modest speed-up in learning.

In a first set of transfer experiments, we used the learned priors to learn the same set of tasks again (see figure 7.6a). The learned priors led to a modest speed-up relative to learning from scratch, comparable with the results in figure 7.4. LSTMs performed best in this setting.

We also experimented with reusing the latent space directly as follows; we can freeze the conditional distribution $\pi_0(a|s,z)$ and learn a new policy $\pi(z|s)$, effectively using the learned latent space as an action space. This idea, illustrated by figure 7.5, is related to the approach taken by Haarnoja et al. (2018a), but differs in the fact that we do not use a deterministic, invertible transformation of the action and in the training details. It allows us to use a lower-dimensional latent space as the action space. Most of the latent space should encode sensible behaviours, potentially dramatically simplifying the exploration problem. On the other hand, it means that if the new tasks require a behaviour that is not representable in the latent space we may not be able to solve the task at all. This approach proved remarkably successful, learning to solve the task in an order of magnitude fewer steps.

In a second set of experiments, we study how well a prior learned on the walking

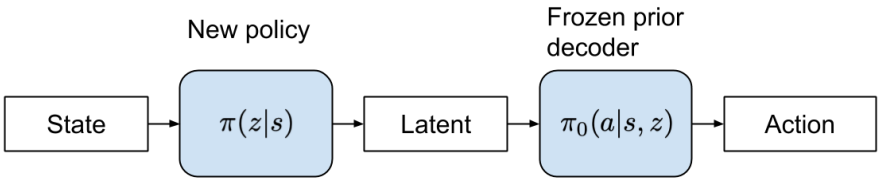
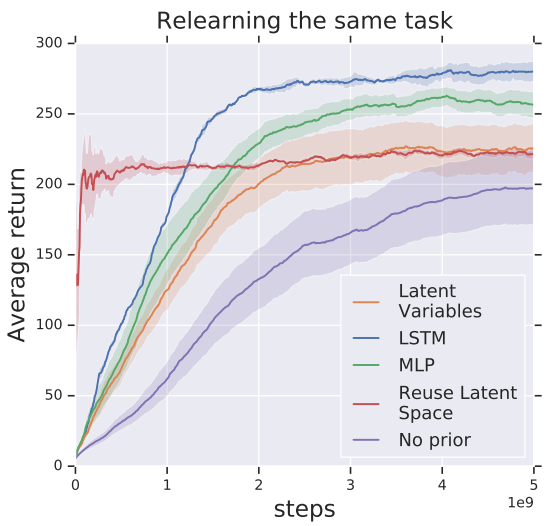
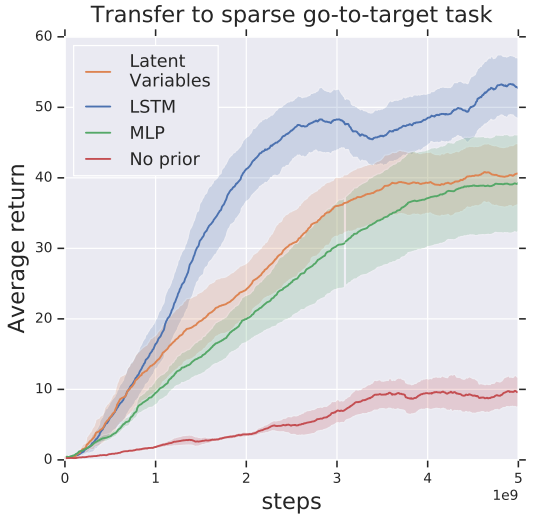


Figure 7.5: We can reuse the learned latent space as a new, low-dimensional action space.

task can transfer to the sparse go-to-target task (see figure 7.6b). Here all learned priors led to a significant speed up relative to learning from scratch, again consistent with our findings in Galashov et al. (2019). Small return differences aside, all three different priors considered here solved the task with clear goal directed movements. On the other hand, the baseline only learned to go to very close-by targets. Reusing the latent space did not work well on this task. We speculate that the required turns are not easy to represent in the latent space resulting from the walking task.



(a) Using the learned prior on the same set of tasks results in substantially faster training. It is also possible to reuse the latent space as a new action space, allowing the agent to solve the tasks in a fraction of the time.



(b) Priors learned in the dense running tasks can be used to speed up learning on a sparse go-to-target task. All types of priors substantially speed up learning relative to the baseline.

Figure 7.6: Results using latent variables in the prior.

7.6 Conclusion

In this chapter, we have reviewed RL as inference and described methods to learn prior policies in continuous control. We have shown that learning a prior policy can speed up learning in sparse reward tasks with complex bodies in continuous control. We also presented some exploratory experiments with priors containing latent variables, which performed broadly similar to MLP or LSTM priors on the task we examined. However, it should be noted that latent variable priors introduce additional complexity into the learning problem. For simple problems MLPs and LSTMs are easier to train and preferable. We showed that it is sometimes possible to reuse the learned latent space as a new action space, an idea that proves extremely effective when scaled to a very large number of tasks with a similar latent variable model in the next chapter.

7.7 Supplementary Material

This appendix contains additional experimental details about the tasks and the architecture. This is largely adapted from the appendix of Galashov et al. (2019).

7.7.1 Task details

In this section we describe the tasks used in chapter 7 in detail.

Walking task.
Type. Dense-reward task.
Description. At the beginning and half-way through an episode a random direction from left, right, forward or backwards is sampled. The required direction is supplied to the agent via a one-hot encoding. The walker is required to move in this direction with the target speed v_t.
Reward. Let v_{cur} denote the instantaneous velocity in the required direction. The reward is then given by $r = \exp^{-|v_{cur}-v_t|^2}$.
Technical details. Target speed, $v_t = 3$. The episode length is 10 seconds. For the humanoid task we terminated episode when the position of the head of the humanoid dropped below $h < 0.95$.

Go to one of K targets.
Type. Sparse-reward task.
Description. At the beginning of each episode K targets are randomly placed in a

room. The walker is also randomly initialised. The walker is required to go one of the K targets, as specified by a feature vector. Once the walker gets sufficiently close to the target, the episode terminates and the walker receives the reward r.
Reward. $r = 60$.
Technical details. The episode length is 20 seconds.

Go to a moving target.
Type. Sparse-reward task.
Description. Similar to the previous one, but there is only one target and once the walker achieves it, the target reappears in a new random place. The walker receives r for 10 consecutive steps staying on the target before the target reappears in a new location.
Reward. $r = 1$ when the walker is close to the target.
Technical details. The episode length is 25 seconds.

Move a box to one of the K targets and go to another target location.
Type. Sparse-reward task.
Description. A box and K target locations are randomly placed in a room. The walker receives the location of the box and the target. In addition, the walker is required to move to another target location. The walker receives the reward r_{task} for each task, and an additional reward r_{end} for solving both tasks.
Reward. $r_{task} = 10$, $r_{end} = 50$.
Technical details. The episode length is 30 seconds. The control timestep, i.e. the time interval at which actions are taken, is 0.05 for *ant* and 0.025 for *jumping ball*.

Foraging in the maze.
Type. Sparse-reward task from vision.
Description. In this task there is a maze with 8 apples which the agent must collect. For each apple, it receives reward r. The episode terminates once the walker collects all the apples or after 90 seconds.
Reward. $r = 1$ per apple.
Technical details. The episode length is 90 seconds. The control timestep, i.e. the time interval at which actions are taken, is 0.025 for *jumping ball*, and 0.05 for *ant*.

7.7.2 Architecture and Algorithm Details

In our experiments in Galashov et al. (2019), we optimised hyperparameters separately for each walker-task combination. Generally, we used a target network update period $P = 100$ and MLPs with two hidden layers with about 300 hidden units per layer. We generally used 128 actors to collect partial trajectories of length $K = 10$ steps and used a batchsize of 512. The value of the hyperparameter α trading off the reward and the KL to the prior was generally set to $\alpha = 0.01$. To encode the vision inputs we used a ResNet block (He et al., 2015) with filter sizes (16, 32, 32). We generally used the Adam optimizer (Kingma and Ba, 2015) with learning rates between 0.0001 and 0.0005. For our exploratory results on latent variable priors, we used a latent space of dimension 2 (although dimensions 5 and 10 lead to qualitatively similar results). We used similar hyperparameter settings to the ones in Galashov et al. (2019). We used MLPs of similar size to the other components for the inference network.

The benefits of priors were very consistent in our experiments and did not depend heavily on the exact hyperparameters. For additional details on the experiments, please see the appendix of that work in Galashov et al. (2019).

Chapter 8

Neural Probabilistic Motor Primitives

The following chapter is a self-contained paper that has been accepted for publication as:

> Merel*, J., **Hasenclever*, L.**, Galashov, A., Ahuja, A., Pham, V., Wayne, G., Teh, Y.W. and Heess, N., 2019. Neural Probabilistic Motor Primitives for Humanoid Control. International Conference on Learning Representations.

There is a statement of authorship at the end of this chapter.

Abstract

We focus on the problem of learning a single motor module that can flexibly express a range of behaviors for the control of high-dimensional physically simulated humanoids. To do this, we propose a motor architecture that has the general structure of an inverse model with a latent-variable bottleneck. We show that it is possible to train this model entirely offline to compress thousands of expert policies and learn a motor primitive embedding space. The trained *neural probabilistic motor primitive* system can perform one-shot imitation of whole-body humanoid behaviors, robustly mimicking unseen trajectories. Additionally, we demonstrate that it is also straightforward to train controllers to reuse the learned motor primitive space to solve tasks, and the resulting movements are relatively naturalistic. To support the training of our model, we compare two approaches for offline *policy cloning*, including an experience efficient method which we call *linear feedback policy cloning*. We encourage readers to view a supplementary video[1] summarizing our results.

[1] https://youtu.be/1NAHsrrH2t0

8.1 Introduction

A broad challenge in machine learning for control and robotics is to produce policies capable of general, flexible, and adaptive behavior of complex, physical bodies. To build policies that can effectively control simulated humanoid bodies, researchers must simultaneously overcome foundational challenges related to high-dimensional control, body balance, and locomotion. Recent progress in deep reinforcement learning has raised hopes that such behaviors can be learned end-to-end with minimal manual intervention. Yet, even though significant progress has been made thanks to better algorithms, training regimes, and computational infrastructure, the resulting behaviors still tend to exhibit significant idiosyncrasies such as wild arm movements (e.g. Heess et al., 2017, Bansal et al., 2018).

One advantage of working with humanoids in this context is that motion capture data is widely available and can serve to help design controllers that produce apparently humanlike movement. Indeed, recent developments are now allowing for the production of highly specialized expert policies which robustly, albeit narrowly, reproduce single motion capture clips (e.g. Liu et al. (2010), Peng et al. (2018)).

A remaining challenge on the way to truly flexible and general purpose control is to be able to sequence and generalize individual movements or "skills" in a task-directed manner. Achieving this goal requires not just the ability to acquire individual skills in the first place, but also an architecture and associated training procedure that then allows to represent, recruit, and compose such a large number of skills in a robust manner with as little additional tuning as possible.

This paper presents a step in this direction. Specifically, the setting we focus on will be one in which we have a large number of robust expert policies that perform single skills well and we wish to transfer these skills into a shared policy that can do what each expert does as well as the expert, while also generalizing to unseen behaviors within the distribution of skills. The desiderata in designing our approach were to enable robust control for diverse skills in a form that allowed for one-shot imitation as well as permitting straightforward reuse of skills in a task-directed manner. Furthermore, we would like our approach to scale to a very large number of individual skills while also minimizing the level of manual intervention that is required to enable their composition and sequencing.

We assume that during training time, we have access to the expert policies, either in the form of a large number of trajectories of states and actions $\{(s_t^{(i)}, a_t^{(i)})_{t=1}^{T}\}_{i=1}^{N}$ logged from the experts, or in the form of one trajectory of states and actions per

expert as well as action-state Jacobians of the expert policies along this trajectory. The large number of trajectories or the Jacobian information allow us to capture to robustness of the expert policies to slight perturbations. At test time for the one-shot imitation problem, we assume that we are presented with a held-out trajectory $(s_1^{\text{test}}, \ldots, s_T^{\text{test}})$ of states to imitate. In addition we assume that the body is set to the initial state s_1^{test}. The problem is then to produce actions $(a_1, \ldots, a_{T-1})$ to imitate the held-out state sequence.

Our primary contribution is the development of a neural network architecture for representing a space of probabilistic motor primitives, which we refer to as *neural probabilistic motor primitives*. This architecture is designed to perform one-shot imitation, while learning a dense embedding space of a large number of individual motor skills. Once trained, this module does not just reproduce individual behaviors in the training data, but enables to sequence these behaviors in a controlled fashion, and to synthesize novel movements consistent with the training data distribution. Empirically, we also find that training controllers to reuse this learned motor primitive module for new tasks generates surprisingly human-like movement and the behavior generated seems to interpolate the space of behaviors well.

In order to facilitate transfer and compression of expert skills at the scale of thousands of behaviors, we wish to avoid closed-loop RL training. We call the general, offline, functional transfer of policy content *policy transfer* or *policy cloning* and consider two approaches. The natural baseline approach involves the application of behavioral cloning to data gathered by executing experts many times, with noise, and logging intended expert actions, resembling the approach of Laskey et al. (2017). This works well, as it ensures the student behaves like the expert not only along nominal expert rollouts but also at points arrived at by perturbing the expert. However, this approach may require many rollouts, which can be costly to obtain in many settings. As a more efficient alternative we therefore consider a second solution that operates by comprehensively transferring the functional properties of an expert to a student policy by matching the local noise-feedback properties along one or a small number of representative expert reference trajectories. We call this specific proposal *linear feedback policy cloning (LFPC)*, and we demonstrate that it is competitive with behavioral cloning from many more rollouts in our setting.

8.1.1 Background & Related Work

Recent efforts in RL for *humanoid control* build on a large body of research in robotics and animation. While contemporary results for learning from scratch (Schulman

et al., 2015, Heess et al., 2017) can be impressive the behaviors are not consistently human-like. Learning from motion capture (mocap) can provide strong constraints, especially for running (Peng et al., 2017, Merel et al., 2017). Several recent approaches have demonstrated that it is possible to acquire specific behavioral skills, possibly jointly with external RL objectives (Merel et al., 2017, Peng et al., 2018, Liu and Hodgins, 2018). At present, the policies produced tend to be restricted to single skills/behaviors and can require very large quantities of environment interactions, motivating us to seek methods which reuse existing single-skill expert policies.

Knowledge transfer refers to the broad class of approaches which transfer the input-output functional mapping, to some extent or another, from a *teacher* (or *expert*) to a *student* (Hinton et al., 2015, Srinivas and Fleuret, 2018, Furlanello et al., 2018). *Distillation* connotes the transfer of function from one or more expert systems into a single student system often with the goal of compression or of combining multiple experts qualities (Hinton et al., 2015, Parisotto et al., 2015, Rusu et al., 2015, Teh et al., 2017). *Imitation learning* is the control-specific term for the production of a student policy from either an expert policy or the behavioral demonstrations of an expert. One basic algorithm is *behavioral cloning*, which refers to supervised training of the policy from state-action pairs. In the most simple case it only requires examples from the expert. A broader setting is that in which more liberal queries to the expert are permitted; e.g. for the online-imitation setting as in DAgger (Ross et al., 2011), where we assume that we can evaluate the expert policy at new states during training. This setting is often satisfied e.g. if we wish to combine behavior from multiple experts.

One-shot imitation is a concept which means that a trained system, at test time, can watch an example behavior and imitate it, as, for instance, in Duan et al. (2017). More similar to our work is the setting examined by Wang et al. (2017), in which full-body humanoid movements were studied. Compared with this latter work, we will employ an architecture here that encourages imitation of motor details, rather than overall movement type, and we scale our approach to more expert demonstrations. The most similar work to the present work was contemporaneously published (while this paper was on OpenReview) and also demonstrates impressive, large-scale one-shot humanoid tracking – the approach described by Chentanez et al. (2018) encourages direct tracking as well as failure recovery, but relative to the present work does not easily enable skill reuse.

The notion of *motor primitives* is widespread in neuroscience, where there is evidence that lower dimensional control signals can selectively coordinate and blend

behaviors produced by spinal circuits (Bizzi et al., 2008), and that the cortex organizes the behavioral space of primitive motor behaviors (Graziano, 2006). In this work a motor primitive refers to a context triggered execution of a self-stabilized motor pattern (i.e. a robust policy execution of a behavior in an environment), and the particular architecture we are considering is inspired by the formalization presented in Todorov and Ghahramani (2003), which places a probabilistic latent bottleneck on the sensory-motor mapping.

In the robotics literature, there is a rich line of research into various parameterizations of motion trajectories used for robot control. A class of these are referred to as "Movement Primitives" (e.g. Schaal et al., 2003), including the "Probabilistic Movement Primitives" of Paraschos et al. (2013) (see also e.g. Neumann et al., 2014). These approaches can be seen as specific implementation choices for a certain notion of motor primitive, emphasizing the parameterization and learning of target motion/kinematic trajectories from repeated demonstrations (Paraschos et al., 2013, Meier and Schaal, 2016), rather than learning the actuation/stabilization element, which is often handled by a pre-specified PID controller. In our setting, neural probabilistic motor primitives refers to the reusable embedding space learned from many related behaviors, which is capable of generating sensory-feedback-stabilized motor behavior when executed in an environment.

It has previously been recognized that linear-feedback policies can work well around optimal trajectories or limit cycles even for high DoF bodies. These can be obtained by sample-based optimization (e.g. Ding et al. (2015)) or by differential dynamic programming (Morimoto and Atkeson, 2003, Tassa et al., 2012, 2014). For linear-quadratic-Gaussian control (Athans, 1971) or differential dynamic programming (Mayne, 1966, Jacobson and Mayne, 1970), we obtain feedback policies where the feedback terms are computed from the value function, amounting effectively to feedback-stabilized plans. Work by Mordatch et al. (2015) has shown that linear-feedback policies resulting from trajectory optimization can be used to train neural networks. We employ a similar idea to transfer optimal behavior from an existing policy, observing that an optimal policy implicitly reflects the structure of the (local) value landscape and appropriately functions as a feedback controller.

8.2 Transfer and compression of expert behaviors

In this section, we will first briefly describe the expert policies used in this work (Sec. 8.2.1). We then describe the Neural Probabilistic Motor Primitive architecture and objective (Sec. 8.2.2). We then describe two approaches for training the module offline (Sec. 8.2.3).

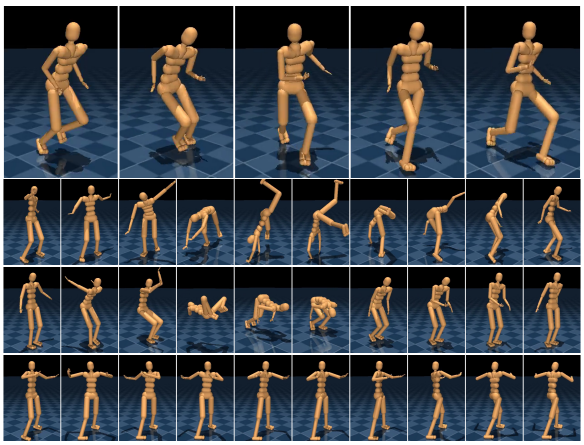


Figure 8.1: Examples of representative experts learned from motion capture. From top to bottom, these are "run and dodge", "cartwheel", "backflip", and "twist". See accompanying video. Note that these four behaviors will be used as representative examples for validation in single-skill transfer experiments.

8.2.1 Obtaining experts from motion capture data

In order to study how to transfer and consolidate experts, we must be able to generate adequate quantities of expert data. For this work, we use expert policies trained to reproduce motion capture clips. The approach we use for producing experts is detailed more fully in Merel et al. (2018) and largely follows Peng et al. (2018). It yields time-indexed neural network policies that are robust to moderate amounts of action noise (see appendix 8.5.1 for additional details on the training procedure). Some examples of the resulting single-skill time-indexed policies that are obtained from this procedure are depicted in Fig. 8.1. All our experts were trained in MuJoCo environments (Todorov et al., 2012).

Data We use the CMU Mocap database[2], which contains more than 2000 clips of varying lengths from more than 100 subjects. The motions in this dataset are quite varied, including many clips of walking, turning, running, jumping, dancing, various hand movements, and many more idiosyncratic behaviors. From this, we selected various clips of generic whole-body movements – any clips longer than 6 seconds were cut into smaller pieces yielding approximately 3000, roughly 2-6 second snippets. Just over half of these are generic locomotion such as walking, running, jumping and turning. The rest of the clips mostly contained diverse hand movements while standing. We trained one expert policy per selected snippet, yielding 2707 expert policies in our training set.

[2]The CMU motion capture database is available at mocap.cs.cmu.edu.

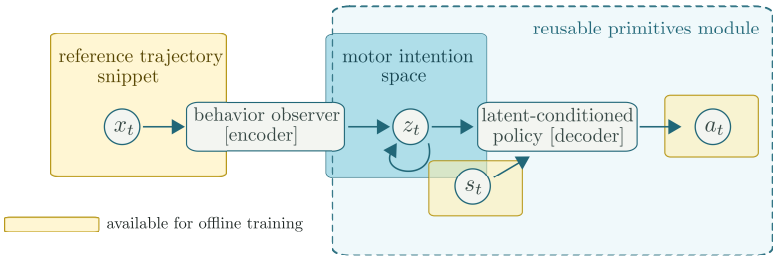


Figure 8.2: Neural probabilistic motor primitive architecture for one-shot skill deployment. The yellow-highlighted information is available for offline, supervised training. Once the full model has been learned, the decoder can be reused as a policy in other settings.

8.2.2 Neural probabilistic motor primitives

Our goal is to obtain a motor primitive module that can flexibly and robustly deploy, sequence, and interpolate a diverse set of skills from a large database of reference trajectories without any manual alignment or other processing of the raw experts. This requires a representation that does not just reliably encode all behavioral modes but also allows effective indexing of behaviors for recall. To ensure plausible and reliable transitions it is further desirable that the encoding of similar behaviors should be close in some sense in the representation space.

Compression of many expert skills via a latent variable inverse model We achieve this goal by training an autoregressive latent variable model of the state-conditional action sequence which, at training time, is conditioned on short look-ahead snippets of the nominal/reference trajectory (see Fig. 8.2). This architecture has the general structure of an inverse model, which produces actions based on the current state and a target. The embedding space produced by training this architecture should reflect short-term motor intention. As we demonstrate below, this allows for the selective execution of particular behavioral modes and also admits one-shot imitation via the trajectory encoder.

We use a model with a latent variable z_t at each time step, modelling the state conditional action distribution. The encoder and decoder are distributions $q(z_t|z_{t-1}, x_t)$ and $\pi(a_t|z_t, s_t)$ where s_t is the state as in preceding sections and x_t is concatenation of a small number of future states $x_t = [s_t, ..., s_{t+K}]$. The encoder and decoder are MLPs with two and three layers, respectively. For architecture and experimental details see

appendix 8.5.4. The generative part of the model is given by:

$$p(a_{1:T}, z_{1:T}|s_{1:T}) = \prod_{t=1}^{T} p_z(z_t|z_{t-1})\pi(a_t|z_t, s_t). \tag{8.1}$$

Temporally nearby trajectory snippets should have a similar representation in the latent space. To implement this intuition, we choose an AR(1) process as a weak prior:

$$z_t = \alpha z_{t-1} + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \tag{8.2}$$

where $\sigma = \sqrt{1-\alpha^2}$, ensuring that marginally $z_t \sim \mathcal{N}(0, I)$, and set $\alpha = 0.95$ in experiments unless otherwise stated. In future, it is interesting to investigate different values of α and learnable priors.

In order to train this model, we consider the evidence lower bound (ELBO):

$$\mathbb{E}_q \left[\sum_{t=1}^{T} \log \pi(a_t|s_t, z_t) + \beta\big(\log p_z(z_t|z_{t-1}) - \log q(z_t|z_{t-1}, x_t) \big) \right], \tag{8.3}$$

with a β parameter to tune the weight of the prior. For $\beta = 1$ this objective forms the well-known variational lower bound to $\log p(a_{1:T}|s_{1:T})$. This objective can be optimized using supervised learning (i.e. behavioral cloning from noisy rollouts) offline.

Note we chose not to condition the encoder on actions, since we are interested in one-shot imitation in settings where actions are unobserved. We experimented with different values of K and obtained similar performance. All the results reported in this paper use $K = 5$.[3]

Our architecture effectively implements a conditional information bottleneck between the desired future trajectory x_t and the action a_t given the past latent state z_{t-1} (similar to Alemi et al. (2017)). As discussed above the auto-correlated prior encourages an encoding in which temporally nearby latent states from the same trajectory tend to be close in the latent space, and the information bottleneck more generally encourages a limited dependence on x_t with z_t forming a compressed representation of the future trajectory as required for the action choice.

[3]We also experimented with ways to look further into the future by conditioning on $x_t = [s_t, s_{t+1}, s_{t+3}, s_{t+6}, s_{t+10}, s_{t+15}]$. This gave broadly similar results.

8.2.3 Training a student policy from a set of examples

When transferring knowledge from an expert policy to a student we would like the student to replicate the expert's behavior in the full set of states plausibly visited by the expert. In our case, experts trained to reproduce single clips can be conceptualized as nonlinear feedback controllers around a nominal trajectory, and the manifold of states visited by experts can be thought of as a tube around that reference. We require the student to be able to operate successfully in and remain close to this tube even in the face of small perturbations.

Formally, to ensure that the student retains expert robustness, we would like expert actions $\mu_E(s)$ and student actions $\mu_\theta(s)$ to be close under a plausible (noisy) expert state distribution ρ_E. A surrogate loss used in imitation learning as well as knowledge transfer is the quadratic loss between actions (Ross et al., 2011) (or activations Srinivas and Fleuret (2018)).

$$\min_\theta \mathbb{E}_{s\sim\rho_E}[(\mu_E(s) - \mu_\theta(s))^2] \tag{8.4}$$

Behavioral cloning can refer to optimization of this objective, where ρ_E is replaced with an empirical distribution of a set of state-action pairs $\mathcal{S}$. This works well if $\mathcal{S}$ adequately covers the state distribution later experienced by the student. Anticipating and generating an appropriate set of states on which to train the student typically requires many rollouts and can thus be expensive.

Since we are aiming to compress the behavior of thousands of experts we desire a computationally efficient method. We investigate two schemes that allow us to record the experts' state-action mappings on a small-sample estimate of the experts' state distributions and to then train the student via supervised learning. Both schemes are convenient to implement in a regular supervised learning pipeline and require neither querying many experts simultaneously (which limits scalability when dealing with thousands of experts) nor execution of the student at training time.

Behavioral cloning from noisy rollouts The first approach amounts to simply gathering a number of noisy trajectories from the expert (either under a stochastic policy or with noise injection) while logging the optimal/mean action of the expert instead of the noisy action actually executed. A version of this is equivalent to the DART algorithm of Laskey et al. (2017). We then perform behavioral cloning from that data.

Specifically, given an expert policy π_E, let $\mu_E(s)$ be the mean action of the expert in state s. To obtain noisy rollouts, we run π_E^η, the expert with moderate action noise

(η) to obtain a set of data $\{s_k^\eta, \mu_k\}_{1...K}$, where $\mu_k = \mu_E(s_k^\eta)$. And we optimize the policy according to Eqn. 8.4, with the expectation over $s \sim \rho_E$ being approximated by a sum over the set of state and expert-actions collected. While we expect this approach can work well, we do not expect it to be particularly efficient insofar as the expert may need to be executed for many rollouts.

Linear-feedback policy cloning (LFPC) The second approach, which we refer to as linear-feedback policy cloning (LFPC), logs the action-state Jacobian as well as the expert action along a single nominal trajectory. The Jacobian can be used to construct a linear feedback controller which gives target actions in nearby perturbed states during training (described below). This approach is not intended to outperform behavioral cloning, as this should not be possible for arbitrary quantities of expert rollout data. Instead the motivation for LFPC is to do as well as behavioral cloning while using considerably fewer expert rollouts.

As pointed out above, experts trained to reproduce single clips robustly can be thought of as nonlinear feedback controllers around this nominal trajectory. The nominal trajectory refers to the sequence of nominal state-action pairs $\{s_t^\star, a_t^\star\}_{1...T}$ obtained by executing $\mu_E(s)$ recursively from an initial point $s_0^\star$. Since expert behavior in our setting is well characterized by single nominal trajectories, we expect we can capture the relevant behavior of the expert by a linearization around the nominal trajectory[4].

Let δs be a small perturbation of the state and let $\boldsymbol{J} = \frac{d\mu_E(s)}{ds}|_{s=s}$ be the Jacobian. Then

$$\mu_E(s + \delta s) = \mu_E(s) + \boldsymbol{J}\delta s + O\left(\|\delta s\|^2\right). \tag{8.5}$$

This linearization induces a linear-feedback-stabilized policy that at each time-step has a nominal action $a_t^\star$, but also expects to be in state $s_t^\star$, and correspondingly adjusts the nominal action with a linear correction based on discrepancy between the nominal and actual state at time t:

$$\mu_{FB}(s_t) = a_t^\star + \boldsymbol{J}_t^\star(s_t - s_t^\star), \quad \text{where} \quad \boldsymbol{J}_t^\star = \left.\frac{d\mu_E(s)}{ds}\right|_{s=s_t^\star}. \tag{8.6}$$

We empirically validated that a linear feedback policy about the nominal trajectory of the expert can approximate the expert behavior reasonably well for clips we examine (see results Fig. 8.3).

[4]Note that in contemporary neural network languages, it is straightforward to automatically compute the Jacobian of the actions with respect to observation inputs.

Above we presented the expert as a feedback controller operating in a tube around some nominal trajectory with states $s_1^\star, \ldots, s_T^\star$, actions $a_1^\star, \ldots, a_T^\star$, and Jacobians $\boldsymbol{J}_1^\star, \ldots, \boldsymbol{J}_T^\star$. We approximate ρ_E with the distribution of states introduced by state perturbations around this nominal trajectory:

$$\min_\theta \frac{1}{T} \sum_i \mathbb{E}_{\delta s_i \sim \Delta(s)}[\|\mu_E(s_i + \delta s_i) - \mu_\theta(s_i + \delta s_i)\|^2]. \tag{8.7}$$

However, this objective still requires expert evaluations at the perturbed states. Using the linearization described above we can replace the expert action $\mu_E(s + \delta s)$ with the Jacobian-based linear-feedback policy $\mu_{FB}(s+\delta s)$, which is available offline. This yields the LFPC objective:

$$\min_\theta \frac{1}{T} \sum_i \mathbb{E}_{\delta s_i \sim \Delta(s)}[||\mu_\theta(s_i^\star + \delta s_i) - a_i^\star - \boldsymbol{J}_i^\star \delta s_i||_2^2], \tag{8.8}$$

One potentially important choice is the perturbation distribution $\Delta(s)$. Ideally, we would like $\Delta(s)$ to be the state-dependent distribution induced by physically plausible transitions, but estimating this distribution may require potentially expensive rollouts which we are trying to avoid. A cheaper object to estimate is the *stationary* transition noise distribution induced by noisy actions, which can be efficiently approximated from a small number of trajectories. Empirically, we found the objective (8.8) to be relatively robust to some variations in Δ, and we use a fixed marginal distribution for all clips.

Objective 8.8 bears interesting similarities to approaches such as denoising autoencoders (Vincent et al., 2008), where networks can learn to ignore local noise perturbations on inputs sampled from a high-dimensional noise distribution. Further, Mordatch et al. (2015) successfully distill feedback policies obtained from a planner. One question left open by this latter work is that of how much data might be required. Empirically we show in the experiments below that the augmented objective (8.8) can produce the desired robustness even from a very limited set of states.

There are multiple, relevant perspectives on LFPC. From one perspective, LFPC amounts to a data augmentation method. From another vantage, the approach attempts to match the mean action as well as the Jacobian at the set of relevant behavioral states, here sampled along the nominal trajectory. In settings where expert behavior is more diverse or multimodal, LFPC should be applied to states which representatively cover relevant behavioral modes or perhaps are expanded backwards from goal states (roughly similar to the procedure used to expand LQR-trees Tedrake (2009)). Explicit Jacobian matching has been proposed elsewhere, for example in

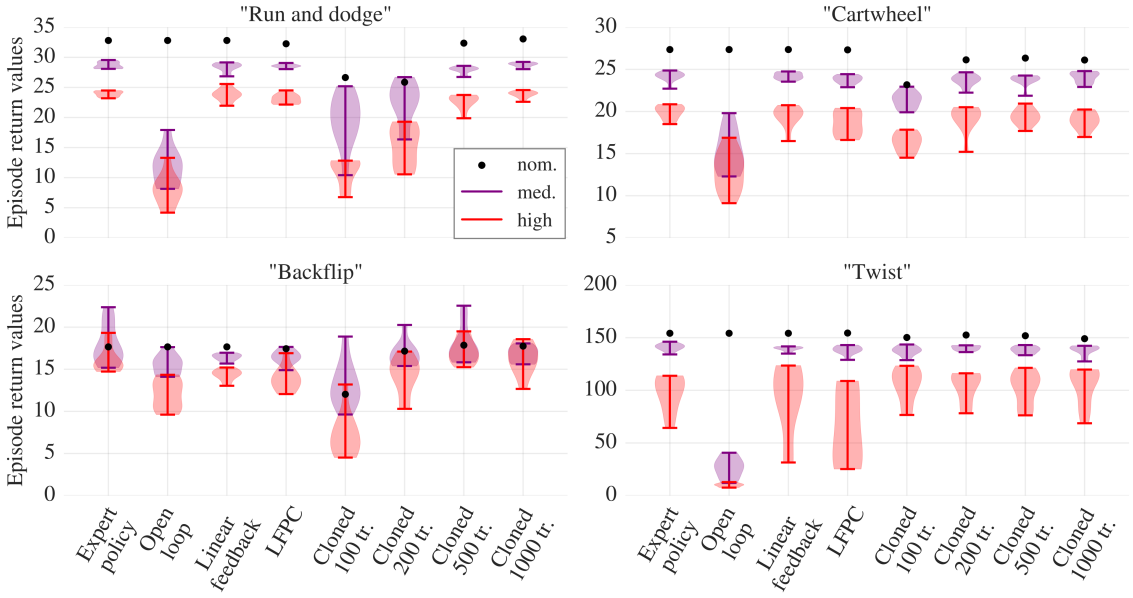


Figure 8.3: Comparisons of trajectory rollouts for 4 reference behaviors for the nominal trajectory and at varying noise levels. Note that the score is determined by similarity to motion-capture reference and the expert may be slightly suboptimal so slight improvements on the expert may arise by chance.

Czarnecki et al. (2017). See appendix 8.5.2 for further disambiguation relative to other approaches.

To train our Neural Probabilistic Motor Primitive architecture using LFPC we can adapt the objective in Eqn. 8.3 as follows:

$$\mathbb{E}_{\delta s,q}\left[\sum_{t=1}^{T}\log\pi(a_t+\boldsymbol{J}_t\delta s_t|s_t+\delta s_t,z_t)+\beta\big(\log p_z(z_t|z_{t-1})-\log q(z_t|z_{t-1},x_t+\delta x_t)\big)\right], \tag{8.9}$$

where δs_t are i.i.d. perturbations drawn from suitable perturbation distribution Δ and δx_t is the concatenation of $[\delta s_t, \delta s_{t+1}, ..., \delta s_{t+K}]$.

8.3 Experiments

8.3.1 Validation: transfer of single-behavior policies

To ground our results in a simple setting, we begin with transfer of a single-skill, time-indexed policy from one network to another. We compare the performance of various time-indexed policies for each of the experts depicted in Fig. 8.1. We compare the original expert policy, an open-loop action sequence along the experts nominal (i.e. mean) trajectory, a linear feedback policy along the expert nominal trajectory,

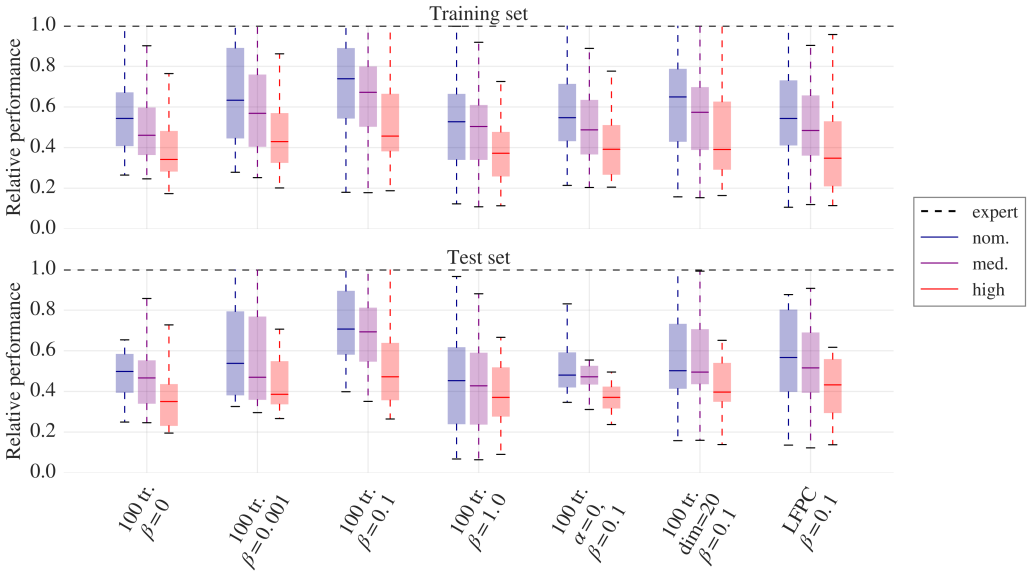


Figure 8.4: Performance relative to expert policies for trained neural probabilistic motor primitive models. Performance of model variations are compared on training and testing data. We compare models trained using cloning with 100 trajectories per expert for different levels of regularization, using a smaller latent space of dimension 20 rather than 60 in all other experiments, as well as LFPC.

as well as the network trained to match the linear-feedback behavior (LFPC). In addition we compare to policies trained from 100, 200, 500 or 1000 trajectories with behavioral cloning. We compare each approach with no action noise, small action noise, and moderate action noise (noise is i.i.d. normal per actuator with standard deviation magnitude .05 and .1 respectively, for action ranges normalized to $[-1, 1]$). Note that, open loop control almost always fails if the state is perturbed by even a small ϵ (though perhaps surprisingly, the backflip can almost be executed open loop due to limited ground contact). Remarkably, LFPC with a single trajectory performs on par with behavioral cloning based on hundreds of trajectories (see Fig. 8.3).

8.3.2 Core results: compressing thousands of experts

Having validated that single skills can be transferred, we next consider how well we can compress behaviors of the 2707 experts in our training set into the neural probabilistic motor primitive architecture. Assessing the models using the action-reconstruction loss is not very intuitive since it does not capture model behavior in the environment. Instead we report a more relevant measure based on expert imitation. Here we encode an expert state trajectory $\{s_1, \ldots, s_T\}$ into a sequence of latent variables $\{z_1, \ldots, z_{T-K}\}$ using our encoder network and then *execute the (decoder)*

policy $\pi(a_t|\cdot, z_t)$ *in the environment* conditioned on this sequence starting in the initial state s_1 of the expert trajectory. Note that this approach is open-loop with respect to the latents while being closed-loop with respect to state. We can then compare the performance of the trained system against experts on training and held-out clips according to the tracking reward used to train the experts originally. To account for different expert reward scales we report performance relative to the expert policy. Importantly, that this approach works is itself a partial validation of the premise of this work, insofar as open-loop execution of action sequences usually trivially fails with minor perturbations. The trained neural probabilistic motor primitive module can execute behaviors conditioned on an open-loop noisy latent variable trajectory, implying that the decoder has learned to stabilize the body during latent-conditioned behavior.

There are a few key takeaways from the comparisons we have run (see Fig. 8.4). Most saliently cloning based on 100 trajectories from each expert with a medium regularization value ($\beta = 0.1$) works best. LFPC with comparable parameters works less well here, but has qualitatively fairly similar performance. Our ablations show that regularization and a large latent space are important for good results. We also set the autoregressive parameter $\alpha = 0$ (.95 in other runs), making the latent variables i.i.d.. This hurts performance, validating our choice of prior.[5]

8.3.3 Analysis of the trained model

We have no expectation that trajectories well outside the training distribution are likely to be either representable by the encoder or executable by the decoder. Nevertheless, when one-shot imitation of a trajectory fails, a natural question is whether the decoder is incapable of expressing the desired actions, or the encoder fails to encode the trajectory in such a way that the decoder will produce it. We propose an analysis to distinguish this for borderline cases. For held out trajectories that yield unsatisfying performance on one-shot imitation, we can simply optimize directly:

$$\min_{z_1 \ldots z_T} \sum_{t=1}^{T} ||\mu_\theta(s_t, z_t) - a_t^\star||_2^2, \tag{8.10}$$

where μ_θ is the decoder mean. Empirically we see that this optimization meaningfully improves the executed behavior, and we visualize the shift in a three-dimensional space given by the first three principal components in Fig. 8.5.

[5] One other feature of the training pipeline we experimented with is mirroring of policies according to bilateral symmetry (thereby approximately doubling the expert data) – this improves results slightly and all models compared here use mirroring.

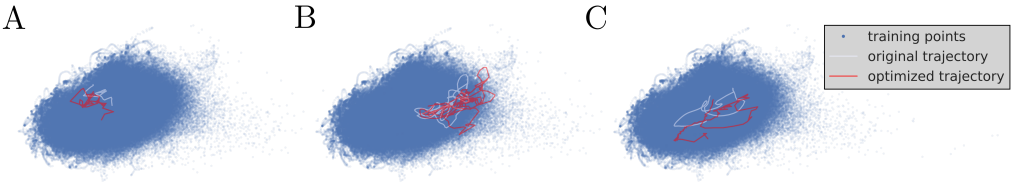


Figure 8.5: These panels consist of visualizations of the PCA latent space with comparisons in this space between one-shot latent-variable sequences and optimized latent variable sequences for various behaviors: A. Run B. Backwards walking C. Jumping. Running executes well based on the one-shot trajectory so serves as a reference for which optimization is not noticeably different. Walking backwards and jumping one-shot imitations fail, but are noticeably improved by optimization.

We exhibit three examples where we visualize the original latent trajectory as well as the optimized latent trajectory. Performance is significantly improved (see supplementary video[6]), showing the latent space can represent behaviors for which one-shot imitation fails. However execution remains imperfect suggesting that while much of the fault may lie with the encoder, the decoder still may be slightly undertrained on these relatively rare behavior categories. Quantitatively, among a larger set of clips with less than 50% relative expert performance for one-shot imitation we found that optimization as described above improved median relative expert performance from 43% to 78%.

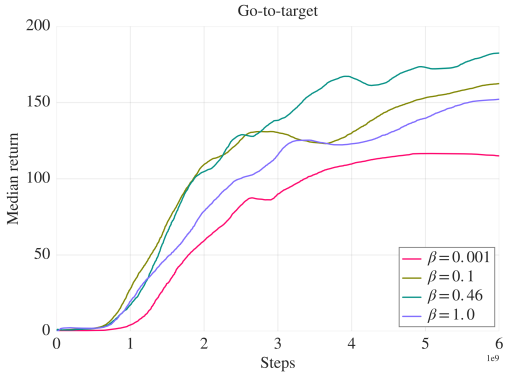
Other exploratory probes of the module suggest that it is possible in certain cases to obtain seamless transitioning between behaviors by concatenating latent-variable trajectories and running the policy conditioned on this sequence (e.g. in order to perform a sequence of turns). See additional supplementary video[7].

Reuse of motor primitive module Finally, we experimented with reuse of the decoder as a motor primitive module. We treat the latent space as a new custom action space and train a new high-level (HL) policy to operate in this space. At each time-step the high-level policy outputs a latent-variable z_t. The actual action is then given by the motor primitive module $p(a_t|s_t, z_t)$. A natural locomotion task that can challenge the motor module is a task which requires abrupt, frequently redirected movement with sharp turns and changes of speed. To implement this we provide the higher-level controller with a target that is constant until the humanoid is near it for a few timesteps at which point it randomly moves to another nearby location. While no single task will comprehensively probe the module, performing well in this task demands a wide range of quick locomotion behavior. For training we used SVG(0)

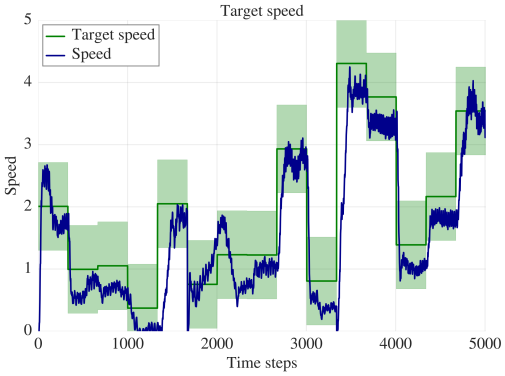
[6]https://youtu.be/1NAHsrrH2t0
[7]https://youtu.be/whGyw9nfPsc

(Heess et al., 2015) with the Retrace off-policy correction for learning the Q-function (Munos et al., 2016). With only a sparse task reward, the HL-controller can learn to control the body through the learned primitive space, and it produces rather humanlike task-directed movement. We observed that more regularized motor primitive modules had more stable initial behavior when connected to the untrained high-level controller (i.e. were less likely to fall at the beginning of training). Compared to a very weakly regularized module ($\beta = 0.001$), more regularized motor primitives modules both trained faster and achieved higher final performance (see Fig. 8.6a). We also investigated a go-to-target task with bumpy terrain that is unobserved by the agent. The fact that our model can learn to solve this task demonstrates its robustness to unseen perturbations for which the motor primitive module was not explicitly trained. In another experiment we investigated a task in which the agent has to move at a random, changing target speed. This requires transitions between qualitatively different locomotion behavior such as walking, jogging, and running (see Fig. 8.6b). See an extended video[8] of these experiments. In a final reuse experiment, we consider an obstacle course requiring the agent to jump across gaps (as in Merel et al. (2018)). We were able to solve this challenging task with a high-level controller that operated using egocentric visual inputs (see the main supplementary video).

[8]https://youtu.be/gRy7yOJgd5Q



(a) Median return value across 10 seeds for the go-to-target task vs learner steps. Compared to a very weakly regularized module ($\beta = 0.001$), more regularized motor primitives modules both trained faster and achieved higher final performance.



(b) Our model is able to track the target speed accurately. Shown here are target speed and actual speed in the egocentric forward direction for three episodes. The reward function is a Gaussian centered at the target speed. The shaded region corresponds to ± one standard deviation.

Figure 8.6: Reuse of neural probabilistic motor primitive modules.

We emphasize a few points about these results to impact their importance: (1) Using a pretrained neural probabilistic motor primitives module, new controllers can be trained effectively from scratch on sparse reward tasks, (2) the resulting movements are visually rather humanlike without additional constraints implying that the learned embedding space is well structured, and (3) the module enables fairly comprehensive and smooth coverage for the purposes of physics-based control.

8.4 Discussion

In this paper we have described approaches for transfer and compression of control policies. We have exhibited a motor primitive module that learns to represent and execute motor behaviors for control of a simulated humanoid body. Using either a variant of behavioral cloning or linear feedback policy cloning we can train the neural probabilistic motor primitive module capable of robust one-shot-imitation, and with the latter we can use relatively restricted data consisting of only single rollouts from each expert. While LFPC did not work quite as well in the full-scale model as cloning from noisy rollouts, we consider it remarkable that it is possible in our setting to transfer expert behaviour using a single rollout. We believe LFPC holds promise insofar as it may be useful in rollout-limited settings, and there is room for further improvement as we did not carefully tune certain parameters, most saliently the marginal noise distribution Δ. In future work, we envision LFPC could be adapted for use in settings where rollouts are costly to obtain, such as real-world robots.

The resulting neural probabilistic motor primitive module is interpretable and reusable. We are optimistic that this kind of architecture could serve as a basis for further continual learning of motor skills. This work has been restricted to motor behaviors which do not involve interactions with objects and where a full set of behaviors are available in advance. Meaningful extensions of this work may attempt to greatly enrich the space of behaviors or demonstrate how to perform continual skll learning and reuse.

Acknowledgments

The data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

8.5 Supplementary Material

8.5.1 Motion capture experts

The approach we use for producing experts is detailed more fully in Merel et al. (2018). In short, this approach for producing experts largely follows Peng et al. (2018). We took the energy function proposed in SAMCON (Liu et al., 2010), and use it as a per timestep reward to train a time-indexed policy that tracks/imitates a motion capture reference clip (Peng et al., 2018). As proposed in Merel et al. (2017), Peng et al. (2018), episodes are initialized to poses throughout the motion capture reference and episodes are early-terminated when the character falls. Here we use an off-policy RL algorithm, SVG(0) (Heess et al., 2015) with Retrace (Munos et al., 2016). As done in Merel et al. (2017), Peng et al. (2018) and elsewhere, we train stochastic policies and use the mean (i.e. noiseless) action as the expert policy.

8.5.2 Relationship to other knowledge transfer ideas

Firstly, we note that the emphasis of the proposal in this work is to match the responsivity of the expert policy in a neighborhood around each state. This is distinct from activation matching or KL matching where the emphasis is on matching the action/activation distribution for a particular state (Rusu et al., 2015, Teh et al., 2017). Secondly, we emphasize that the kind of robust knowledge transfer we discuss here is distinct from that which is seen to be important in other settings. For example Srinivas and Fleuret (2018) provide a line of reasoning that involves training a student system to match the exact activations of a teacher in the presence of perturbations on the student inputs. This logic is sound in the setting of large-scale vision systems. However in the context of control policies, this would look like:

$$\min_{\theta} \sum_{s \in \mathcal{S}^\star} \mathbb{E}_{\delta s \sim \Delta(s)}[(\mu_E(s) - \mu_\theta(s + \delta s))^2] \tag{8.11}$$

This essentially means that the student policy is learning to "blindly" reproduce the action of the expert exactly, despite input perturbations. While this is well motivated if the noise is thought to be orthogonal to the proper functioning of the system, this is a very bad idea for control, where you need to pay close attention to small input perturbations. Technically, this amounts to setting the local feedback to zero, and behaving in a sort of *open-loop*-like fashion.

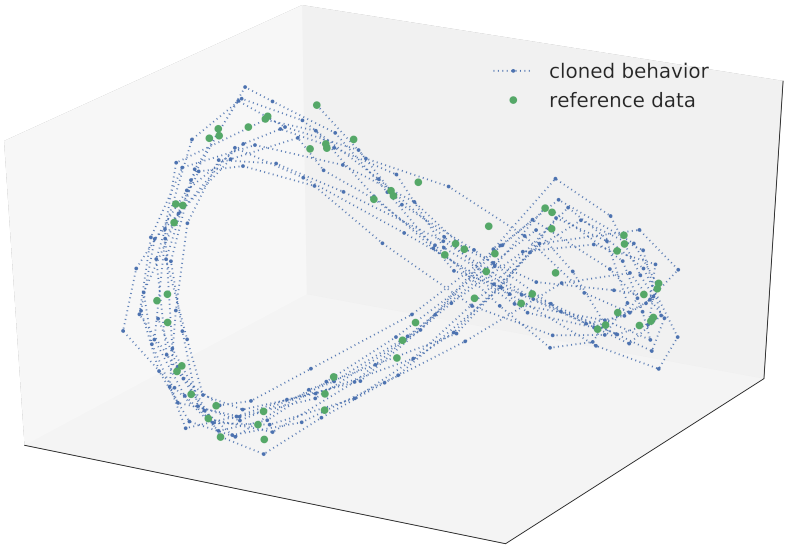


Figure 8.7: Dimensionality reduction (PCA) performed on set of poses obtained from noisy rollouts of the stationary cloned policy (blue). The limited reference data originating from a time-indexed policy has been projected into the same space (green). Observe that the rollouts are considerably noisier and consistently deviate from the reference trajectory, nevertheless the cloned-policy trajectories return to the limit cycle.

8.5.3 Visualization of stationary policy behavior

Locomotion behavior is, at least in the simplest case roughly a limit cycle. In an additional experiment to test LFPC we gathered three gait cycles of running behavior and performed LFPC. Note that here the student policy need not be time-indexed even when the demonstrations were time-indexed. This restricted case shows striking generalization in the presence of noise (see Fig. 8.7).

8.5.4 Architecture and training details

The decoder $p(a_t|s_t, z_t)$ in our experiments was a MLP with three layers with 1024 hidden units taking as input the concatenation of state s_t and latent variable z_t. The decoder output distribution is a multivariate Gaussian with fixed standard deviation of 0.1 (action values are normalized to $[-1, 1]$). We found that fixing the standard deviation made it significantly easier to prevent overfitting. Note that in this setting varying the β parameter is equivalent to varying the fixed output variance (up to a constant). The encoder $q(z_t|z_{t-1}, x_t)$ in our experiments was also an MLP with two layers of 1024 hidden units each. The inputs were simply concatenated at the input.

The encoder output distribution was a multivariate Gaussian with learnt variance. In most of our experiments, we used a 60-dimensional latent space.

We used the reparametrization trick (Kingma and Welling, 2013, Rezende et al., 2014) to train the model and used stochastic gradient descent with ADAM (Kingma and Ba, 2015) with a learning rate of 0.0001. In the case of models trained on 100 trajectories per expert we used minibatches of 512 subsequences of length 30.

For LFPC we sampled 32 subsequences of length 30 and produced 5 perturbed state sequences per subsequence. In preliminary experiments the length of the subsequences did not have a major impact on model performance.

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	Neural Probabilistic Motor Primitives for Humanoid Control
Publication Status	□Published x Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work
Publication Details	Josh Merel*, Leonard Hasenclever*, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, Nicolas Heess, Neural Probabilistic Motor Primitives for Humanoid Control, in *International Conference on Learning Representations*, 2019.

Student Confirmation

Student Name:	Leonard Hasenclever	
Contribution to the Paper	I set up most of the infrastructure required for this paper and performed the experiments with the NPMP architecture. I trained the expert policies based on previous work by Josh Merel. I contributed to designing the NPMP architecture and writing the paper. The LFPC idea as well as most of the literature review is due to Josh Merel.	
Signature	Date	

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: Professor Yee Whye The		
Supervisor comments		
Signature	Date	

Chapter 9

Conclusion

This thesis focused on probabilistic machine learning and described several contributions to probabilistic methods for machine learning as well as two applications of probabilistic thinking in reinforcement learning. In this chapter, we discuss the results of this thesis at a high level and suggest directions for future work.

Before discussing the individual contributions in slightly more detail, it is worth stepping back and critically discussing the rationale for probabilistic methods and particularly Bayesian methods. It is often argued that probabilistic models and the Bayesian framework (e.g. Ghahramani (2015)) are desirable because they can represent uncertainty and allow us to reason about it. Indeed, it is possible to argue that probability and the Bayesian framework is essentially the only principled calculus of uncertainty (Cox, 1946, Jaynes, 2003). However, while this is appealing in an idealised setting, it is less clear in real applications when we have to appeal to approximate methods. Indeed, large parts of the literature on Bayesian machine learning do not evaluate the quality and calibration of the uncertainty estimate and instead reports metrics such as the classification accuracy or mean squared error on a held-out test set. It is well known that variational inference methods in particular tend to underestimate the uncertainty in the parameters (see e.g. Wang and Titterington (2005)). Sampling methods can do better, but pose computational challenges. Often optimisation based methods with suitable regularisation could yield similar results at substantially lower computation cost (see Green et al. (2015)).

It is clear that Bayesian modelling is useful in situations in which we have substantial prior knowledge, a moderate number of data points and relatively low dimensional models (Gelman et al., 2013). However, in the "Big Data" setting with tall data ($n \gg p$), there is no clear evidence of such benefits. See e.g. Green et al. (2015), Bardenet et al. (2017). Ultimately, practitioners should carefully consider different methods and choose the best tool for the specific problem setting.

One area in which probabilistic thinking has been quite successful in this thesis is representation learning, making it a crucial ingredient for the neural probabilistic motor primitives presented in chapter 8.

9.1 On Methods for Probabilistic Inference

In part I, we discussed three different strands of methods for probabilistic inference: sampling methods, variational inference, and expectation propagation. We developed new methods relating to each of these strands. First, in chapter 3 we presented Relativistic Monte Carlo (Lu et al., 2017), a new class of HMC and stochastic gradient HMC algorithms with empirically favourable robustness properties. Recently, Livingstone et al. (2017) studied the choice of kinetic energy more theoretically and find that the standard Gaussian kinetic energy is not always optimal.

Second, in chapter 4 we presented stochastic natural gradient EP (SNEP), an novel EP-style algorithm that enables distributed learning. We showed that SNEP is quite effective for large scale logistic regression, a setting in which we know EP-style algorithms to perform well. We also applied SNEP to distributed training of deep learning models, where it performs competitively with other distributed training algorithms. However, it is not clear that the SNEP (or EP style methods in general) provide meaningful estimates of uncertainty in this setting. Deep Learning toolkits such as Tensorflow (Abadi et al., 2016) and Pytorch (Paszke et al., 2017) have come a long way since the beginning of the SNEP project and it would be interesting to reimplement SNEP and related methods in one of these frameworks to conduct additional experiments. An interesting avenue for future work could be to consider the federated learning setting (Konečnỳ et al., 2016), in which many devices learn a shared prediction model based on user interactions. It would also be interesting to compare SNEP for Bayesian neural networks to distributed algorithms using cheaper approximations such a Laplace approximation based on the Fisher information matrix (Pascanu and Bengio, 2013, Kirkpatrick et al., 2017). Finally, results by Lienart (2017) suggest an EP algorithm based on damped updates in the mean parameter space can perform very well and a comparison with SNEP on a large scale problem would be interesting.

In chapter 5, we propose Sylvester Normalising Flows (SNFs), a novel normalizing flow for variational inference that is competitive with and possible better than existing methods, while requiring drastically fewer flow parameters. The results are quite

encouraging and we hope that SNFs will become a standard tool in the variational inference toolbox.

9.2 On applications in Continuous Control

In part II, we introduced two applications of probabilistic thinking to continuous control. We showed that learning priors with access to less information than the task policy can significantly speed up learning in sparse continuous control tasks. Priors are easy to implement within SVG(0) and are almost always beneficial. We hope learning such priors will become a standard tool in continuous control research.

In chapter 8, we introduced neural probabilistic motor primitives (NPMP), a model architecture for humanoid behaviours that is closely related to the latent variable priors in chapter 7. We showed that such an architecture is able to compress thousands of expert policies into one model enabling surprisingly successful one-shot imitation of humanoid behaviours. Because of the encoder structure, the latent space can be viewed as an embedding space of short term motor intentions. Crucially, we can reuse the decoder part of our model as a low-level controller by treating the latent space as a new action space, yielding human-like motions on very hard continuous control tasks.

The success of NPMP opens up a range of exciting directions for future work. Firstly, what are good skill representations in continuous control? Are continuous latent variables sufficient or could better results be obtained with discrete latent variables or latent variables that are held constant for more than one time-step? Can we use more sophisticated inference techniques such as filtering variational objectives (FIVO, Maddison et al. (2017)) to further improve results?

Another interesting question is whether we can learn similar latent spaces through RL alone with methods similar to the latent variable priors discussed in chapter 7.

One of the downsides of NPMP reuse as described in the paper is that it inevitably fails if the behaviour required to solve the task cannot be well represented in the latent space. For an example of this problem, recall the transfer experiment in chapter 7. This raises the question of how NPMP can be adapted to new tasks. This is particularly interesting in a continual learning setting in which we encounter tasks sequentially.

Finally, the one-shot imitation results in chapter 8 are very good, but not perfect. It would be interesting to adapt NPMP to meta-learning to demonstrate a model that can (almost) perfectly imitate human behaviours after a small number of attempts.

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin A. Riedmiller. Maximum a posteriori policy optimisation. *arXiv*, abs/1806.06920, 2018.

Karen E Adolph, Whitney G Cole, Meghana Komati, Jessie S Garciaguirre, Daryaneh Badaly, Jesse M Lingeman, Gladys LY Chan, and Rachel B Sotsky. How do you learn to walk? thousands of steps and dozens of falls per day. *Psychological science*, 23(11):1387–1394, 2012.

A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola. Scalable inference in latent variable models. In *ACM International Conference on Web Search and Data Mining (WSDM)*, 2012.

Sungjin Ahn, Anoop Korattikara, Nathan Liu, Suju Rajan, and Max Welling. Large-Scale Distributed Bayesian Matrix Factorization using Stochastic Gradient MCMC. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, pages 9–18, New York, New York, USA, aug 2015. ACM Press. ISBN 9781450336642. doi: 10.1145/2783258.2783373. URL `http://dl.acm.org/citation.cfm?id=2783258.2783373`.

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *International Conference on Learning Representations*, 2017.

S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10: 251–276, 1998.

S. Amari and H. Nagaoka. *Methods of Information Geometry*. American Mathematical Society, 2001.

S. Amari, A. Cichocki, and H. Yang. A new algorithm for blind signal separation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 8, pages 757–763, 1996.

C. Andrieu and A. Doucet. Simulated annealing for maximum a posteriori parameter estimation of hidden Markov models. *IEEE Transactions on Information Theory*, 46(3):994–1004, 2000.

C Andrieu, A Doucet, and R Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society B*, 72:269–342, 2010.

V. I. Arnol'd. *Mathematical Methods of Classical Mechanics*. Springer, 1989.

Michael Athans. The role and use of the stochastic linear-quadratic-gaussian problem in control system design. *IEEE transactions on automatic control*, 16(6):529–552, 1971.

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

Jack Baker, Paul Fearnhead, Emily B. Fox, and Christopher Nemeth. Control variates for stochastic gradient mcmc. *Statistics and Computing*, Aug 2018. ISSN 1573-1375. doi: 10.1007/s11222-018-9826-2. URL `https://doi.org/10.1007/s11222-018-9826-2`.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *International Conference on Learning Representations*, 2018.

R. Bardenet, A. Doucet, and C. Holmes. Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach. In *International Conference on Machine Learning (ICML)*, 2014.

Rémi Bardenet, Arnaud Doucet, and Chris Holmes. On markov chain monte carlo methods for tall data. *The Journal of Machine Learning Research*, 18(1):1515–1557, 2017.

Alessandro Barp, Franois-Xavier Briol, Anthony D. Kennedy, and Mark Girolami. Geometry and dynamics for markov chain monte carlo. *Annual Review of Statistics and Its Application*, 5(1):451–471, 2018. doi: 10.1146/annurev-statistics-031017-100141. URL `https://doi.org/10.1146/annurev-statistics-031017-100141`.

S. Barthelmé and N. Chopin. ABC-EP: Expectation propagation for likelihood-free Bayesian computation. In *International Conference on Machine Learning (ICML)*, 2011.

H. Battey, J. Fan, H. Liu, J. Lu, and Z. Zhu. Distributed estimation and inference with statistical guarantees. *ArXiv:1509.05457*, 2015.

Leonard E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Oved Shisha, editor, *Inequalities III: Proceedings of the Third Symposium on Inequalities*, pages 1–8, University of California, Los Angeles, 1972. Academic Press.

M. J. Beal. *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Kttler, Andrew Lefrancq, Simon Green, Vctor Valds, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016. URL `https://arxiv.org/abs/1612.03801`.

A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

A. Beskos, N. Pillai, G. O. Roberts, J. M. Sanz-Serna, and A. M. Stuart. Optimal tuning of hybrid Monte Carlo algorithm. *Bernoulli*, 19:1501–1534, 2013. doi: 10.3150/12-BEJ414. URL `http://dx.doi.org/10.3150/12-BEJ414`.

M. Betancourt. A Conceptual Introduction to Hamiltonian Monte Carlo. *ArXiv e-prints*, January 2017.

Christian Bischof and Xiaobai Sun. On orthogonal block elimination. Technical Report MCS-P450-0794, Argonne National Laboratory, Argonne, IL, 10 1997.

Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

Christopher M Bishop, Neil D Lawrence, Tommi Jaakkola, and Michael I Jordan. Approximating posterior distributions in belief networks using mixtures. In *Advances in neural information processing systems*, pages 416–422, 1998.

E Bizzi, VCK Cheung, A d'Avella, P Saltiel, and Matthew Tresch. Combining modules for movement. *Brain research reviews*, 57(1):125–133, 2008.

Åke Björck and Clazett Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 8(2):358–364, 1971.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational Inference: A Review for Statisticians. *ArXiv e-prints*, January 2016.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Network. In *International Conference on Machine Learning (ICML)*, pages 1613–1622, 2015.

Alexandre Bouchard-Côté, Sebastian J Vollmer, and Arnaud Doucet. The bouncy particle sampler: A nonreversible rejection-free markov chain monte carlo method. *Journal of the American Statistical Association*, pages 1–13, 2018.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating Sentences from a Continuous Space. *arxiv*, nov 2015. URL `http://arxiv.org/abs/1511.06349`.

Lawrence D Brown. *Fundamentals of statistical exponential families: with applications in statistical decision theory*. Institute of Mathematical Statistics, 1986.

Russel E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta Numerica*, 7: 149, 1998. doi: 10.1017/S0962492900002804.

Olivier Cappé, Arnaud Guillin, Jean-Michel Marin, and Christian P. Robert. Population monte carlo. *JOURNAL OF COMPUTATIONAL AND GRAPHICAL STATISTICS*, 13:907–929, 2004.

Olivier Cappé, Randal Douc, Arnaud Guillin, Jean-Michel Marin, and Christian P. Robert. Adaptive importance sampling in general mixture classes. *Statistics and Computing*, 18(4):447–459, Dec 2008. ISSN 1573-1375. doi: 10.1007/s11222-008-9059-x. URL `https://doi.org/10.1007/s11222-008-9059-x`.

Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software, Articles*, 76(1):1–32, 2017. ISSN 1548-7660. doi: 10.18637/jss.v076.i01. URL `https://www.jstatsoft.org/v076/i01`.

Changyou Chen, David Carlson, Zhe Gan, Chunyuan Li, and Lawrence Carin. Bridging the gap between stochastic gradient MCMC and stochastic optimization. *AISTATS*, 2016.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic Gradient Hamiltonian Monte Carlo. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1683–1691, 2014. URL `http://jmlr.org/proceedings/papers/v32/cheni14`.

Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. Physics-based motion capture imitation with deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, page 1. ACM, 2018.

Jean-Marie Cornuet, Jean-Michel Marin, Antonietta Mira, and Christian P. Robert. Adaptive multiple importance sampling. *Scandinavian Journal of Statistics*, 39(4):798–812, 2012. doi: 10.1111/j.1467-9469.2011.00756.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9469.2011.00756.x`.

R. T. Cox. Probability, frequency and reasonable expectation. *American Journal of Physics*, 14(1):1–13, 1946. doi: 10.1119/1.1990764. URL `https://doi.org/10.1119/1.1990764`.

Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems*, pages 4281–4290, 2017.

Wojciech Marian Czarnecki, Siddhant M Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Simon Osindero, Nicolas Heess, and Razvan Pascanu. Mix&match-agent curricula for reinforcement learning. *arXiv preprint arXiv:1806.01780*, 2018.

Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2933–2941, 2014.

J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1223–1231, 2012.

G. Dehaene and S. Barthelmé. Expectation propagation in the large-data limit. *ArXiv:1503.08060*, 2015.

P Del Moral, A Doucet, and A Jasra. Sequential Monte Carlo Samplers. *Journal of the Royal Statistical Society B*, 68(3):411–436, 2006.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39: 1–38, 1977.

Kai Ding, Libin Liu, Michiel Van de Panne, and KangKang Yin. Learning reduced-order feedback policies for motion skills. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 83–92. ACM, 2015.

Nan Ding, Youhan Fang, Ryan Babbush, Changyou Chen, Robert D. Skeel, and Hartmut Neven. Bayesian Sampling Using Stochastic Gradient Thermostats. In *Advances in Neural Information Processing Systems*, pages 3203–3211, 2014.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. *arXiv*, abs/1410.8516, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

Joseph L. Doob. Application of the theory of martingales. *Colloq. Intern. du C.N.R.S (Paris)*, 13:2327, 1949.

A. Doucet, N. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*. Statistics for Engineering and Information Science. New York: Springer-Verlag, May 2001.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

S. Duane, A.D. Kennedy, B.J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987a.

Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216 – 222, 1987b. ISSN 0370-2693. doi: https://doi.org/10.1016/0370-2693(87)91197-X. URL `http://www.sciencedirect.com/science/article/pii/037026938791197X`.

Kumar Avinava Dubey, Sashank J. Reddi, Sinead A Williamson, Barnabas Poczos, Alexander J Smola, and Eric P Xing. Variance reduction in stochastic gradient langevin dynamics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1154–1162. Curran Associates, Inc., 2016.

J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *JMLR*, 12:2121–2159, July 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

A. Einstein. On the Electrodynamics of Moving Bodies. *Annalen der Physik*, 17, 1905. URL `http://www.fourmilab.ch/etexts/einstein/specrel/specrel.pdf`.

S. M. A. Eslami, D. Tarlow, P. Kohli, and J. Winn. Just-in-time learning for fast and flexible inference. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv:1802.01561*, 2018. URL `https://arxiv.org/abs/1802.01561`.

Roy Fox, Ari Pakman, and Naftali Tishby. G-learning: Taming the noise in reinforcement learning via soft updates. *arXiv*, abs/1512.08562, 2015.

Tommaso Furlanello, Zachary C Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. *arXiv preprint arXiv:1805.04770*, 2018.

Alexandre Galashov, Siddhant Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojtek M. Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in KL-regularized RL. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=S1lqMn05Ym`.

Zhe Gan, Changyou Chen, Ricardo Henao, David Carlson, and Lawrence Carin. Scalable Deep Poisson Factor Analysis for Topic Modeling. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1823–1832, 2015. URL `http://jmlr.org/proceedings/papers/v37/gan15.html`.

A. Gelman, A. Vehtari, P. Jylänki, T. Sivula, D. Tran, S. Sahai, P. Blomstedt, J. P. Cunningham, D. Schiminovich, and C. Robert. Expectation propagation as a way of life: A framework for Bayesian inference on partitioned data. *arXiv prepriny*, 2014.

Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

S. Geman and C. Hwang. Diffusions for global optimization. *SIAM Journal on Control and Optimization*, 24(5):1031–1043, 1986.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked Autoencoder for Distribution Estimation. *ICML*, pages 881–889, 2015.

Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 36, 2014.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452, 2015.

Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. In *International Conference on Learning Representations*, 2018.

W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

M. Girolami and B. Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, March 2011. doi: 10.1111/j.1467-9868.2010.00765.x. URL `http://dx.doi.org/10.1111/j.1467-9868.2010.00765.x`.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

J. Gorham and L. W. Mackey. Measuring Sample Quality with Stein's Method. In *NIPS*, pages 226–234, 2015. URL `http://papers.nips.cc/paper/5768-measuring-sample-quality-with-steins-method`.

A. Graves. Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2348–2356, 2011.

Michael Graziano. The organization of behavioral repertoire in motor cortex. *Annu. Rev. Neurosci.*, 29:105–134, 2006.

Peter J. Green, Krzysztof Łatuszyński, Marcelo Pereyra, and Christian P. Robert. Bayesian computation: a summary of the current state, and samples backwards and forwards. *Statistics and Computing*, 25(4):835–862, Jul 2015. ISSN 1573-1375. doi: 10.1007/s11222-015-9574-5. URL `https://doi.org/10.1007/s11222-015-9574-5`.

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

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2455–2467. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.pdf`.

H. Haario, E. Saksman, and J. Tamminen. Adaptive Proposal Distribution for Random Walk Metropolis Algorithm. *Computational Statistics*, 14(3):375–396, 1999. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.3205&rep=rep1&type=pdf`.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. *arXiv*, abs/1702.08165, 2017.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, 2018a. URL `http://arxiv.org/abs/1804.02808`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv*, abs/1801.01290, 2018b. URL `http://arxiv.org/abs/1801.01290`.

Anna Harutyunyan, Marc G Bellemare, Tom Stepleton, and Rémi Munos. Q (lambda) with off-policy corrections. In *International Conference on Algorithmic Learning Theory*, pages 305–320. Springer, 2016.

L. Hasenclever, S. Webb, T. Lienart, S. Vollmer, B. Lakshminarayanan, C. Blundell, and Y.W. Teh. Distributed Bayesian Learning with Stochastic Natural Gradient Expectation Propagation and the Posterior Server. *Journal of Machine Learning Research*, 18, 2017a. ISSN 15337928.

Leonard Hasenclever, Jakub Tomczak, Rianne van den Berg, and Max Welling. Variational inference with orthogonal normalizing flows. In *Bayesian Deep Learning, NIPS 2017 workshop*, 2017b.

W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rk07ZXZRb`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015. URL `https://arxiv.org/abs/1512.03385`.

N. Heess, D. Tarlow, and J. Winn. Learning to pass expectation propagation messages. In *Advances In Neural Information Processing Systems (NIPS)*, 2013.

Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

R. Herbrich, T. Minka, and T. Graepel. Trueskill$^{\text{TM}}$: A Bayesian skill rating system. *Advances in Neural Information Processing Systems (NIPS)*, 19:569, 2007.

J. M. Hernandez-Lobato and Ryan Adams. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. In *International Conference on Machine Learning (ICML)*, pages 1861–1869, 2015.

J. M. Hernandez-Lobato, Y. Li, M. Rowland, T. Bui, D. Hernandez-Lobato, and R. Turner. Black-Box Alpha Divergence Minimization. In *International Conference on Machine Learning (ICML)*, pages 1511–1520, 2016.

T. Heskes and O. Zoeter. Expectation propagation for approximate inference in dynamic Bayesian networks. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 18, 2002.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

M. Hoffman, D. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347, 2013a.

M. D. Hoffman and A. Gelman. The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15, 2014.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347, 2013b.

Z. Huang and A. Gelman. Sampling for Bayesian computation with large datasets. Technical report, Department of Statistics, Columbia University, 2005.

D. R. Hunter and K. Lange. A tutorial on MM algorithms. *The American Statistician*, 2004.

Chii-Ruey Hwang, Shu-Yin Hwang-Ma, and Shuenn-Jyi Sheu. Accelerating gaussian diffusions. *The Annals of Applied Probability*, pages 897–913, 1993.

David H. Jacobson and David Q. Mayne. *Differential Dynamic Programming*. Elsevier, 1970.

E. T. Jaynes. Information theory and statistical mechanics. *Statistical Physics*, pages 181–218, 1963.

E. T. Jaynes. *Probability theory: The logic of science*. Cambridge University Press, Cambridge, 2003.

W. Jitkrittum, A. Gretton, N. Heess, S. M. Ali Eslami, B. Lakshminarayanan, D. Sejdinovic, and Z. Szabó. Kernel-based just-in-time learning for passing expectation propagation messages. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

Hilbert J. Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. *Machine Learning*, 87(2):159–182, May 2012. ISSN 1573-0565.

Diederik Kingma and Max Welling. Efficient gradient-based inference through transformations between bayes nets and neural nets. *ICML*, pages 1782–1790, 2014.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2013.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved Variational Inference with Inverse Autoregressive Flow. *NIPS*, pages 4743–4751, 2016.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society B*, 76, 2014.

Tomasz Komorowski, Claudio Landim, and Stefano Olla. *Fluctuations in Markov processes: time symmetry and martingale approximation*, volume 345. Springer Science & Business Media, 2012.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

A. Korattikara, Y. Chen, and M. Welling. Austerity in MCMC land: Cutting the Metropolis-Hastings budget. In *International Conference on Machine Learning (ICML)*, 2014.

Zdislav Kovarik. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 7(3):386–389, 1970.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

P. S. Laplace. Memoire sur les integrales difinies et leur application aux probabilités, et specialement à la recherche du milieu qu'il faut choisir entre les rśultats des observations, 1809.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on Robot Learning*, pages 143–156, 2017.

Normand M Laurendeau. *Statistical Thermodynamics: Fundamentals and Applications*. Cambridge University Press, 2005.

Lucien Le Cam. *Asymptotic Methods in Statistical Decision Theory*. Springer, 1986.

B. Leimkuhler and S. Reich. A Metropolis adjusted Nosé-Hoover thermostat. *ESAIM: Mathematical Modelling and Numerical Analysis*, 43:743–755, 7 2009. ISSN 1290-3841. doi: 10.1051/m2an/2009023. URL `http://www.esaim-m2an.org/article_S0764583X09000235`.

Benedict Leimkuhler and Xiaocheng Shang. Adaptive Thermostats for Noisy Gradient Systems. *SIAM Journal on Scientific Computing*, 38(2):A712–A736, mar 2016. ISSN 1064-8275. doi: 10.1137/15M102318X. URL `http://epubs.siam.org/doi/10.1137/15M102318X`.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arxiv*, abs/1805.00909, 2018. URL `http://arxiv.org/abs/1805.00909`.

Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems*, pages 207–215, 2013.

Chunyuan Li, Changyou Chen, David Carlson, and Lawrence Carin. Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks. *AAAI*, dec 2016a. URL `http://arxiv.org/abs/1512.07666`.

Chunyuan Li, Changyou Chen, Kai Fan, and Lawrence Carin. High-order stochastic gradient thermostats for bayesian learning of deep models. In *AAAI*, pages 1795–1801, 2016b.

Y. Li, J. M. Hernandez-Lobato, and R. E. Turner. Stochastic expectation propagation. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

T. Lienart, Y. W. Teh, and A. Doucet. Expectation particle belief propagation. In *Advances In Neural Information Processing Systems (NIPS)*, 2015.

Thibaut Lienart. *Inference on Markov random fields: methods and applications*. PhD thesis, University of Oxford, 2017.

Z. C. Lipton. Stuck in a what? adventures in weight space. *arXiv preprint arXiv:1602.07320*, 2016.

Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Publishing Company, Incorporated, 2008. ISBN 0387763694, 9780387763699.

Libin Liu and Jessica Hodgins. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4):142, 2018.

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. Sampling-based contact-rich motion control. In *ACM Transactions on Graphics (TOG)*, volume 29, page 128. ACM, 2010.

S. Livingstone, M. Betancourt, S. Byrne, and M. Girolami. On the Geometric Ergodicity of Hamiltonian Monte Carlo. *ArXiv e-print 1601.08057*, 2016.

Samuel Livingstone, Michael F Faulkner, and Gareth O Roberts. Kinetic energy choice in hamiltonian/hybrid monte carlo. *arXiv preprint arXiv:1706.02649*, 2017.

C. Louizos and M. Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning (ICML)*, volume 48, pages 1708–1716, 2016.

X. Lu, T. Rainforth, Y. Zhou, J.-W. van de Meent, and Y. Whye Teh. On Exploration, Exploitation and Learning in Adaptive Importance Sampling. *ArXiv e-prints*, October 2018.

Xiaoyu Lu, Valerio Perrone, Leonard Hasenclever, Yee Whye Teh, and Sebastian Vollmer. Relativistic Monte Carlo . In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1236–1245, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL `http://proceedings.mlr.press/v54/lu17b.html`.

Yi-An Ma, Tianqi Chen, and Emily B. Fox. A Complete Recipe for Stochastic Gradient MCMC. In *Advances in Neural Information Processing Systems*, jun 2015. URL `http://arxiv.org/abs/1506.04696`.

Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

D. J. C. MacKay. Maximum likelihood and covariant algorithms for independent component analysis. http://www.inference.org.uk/mackay/abstracts/ica.html, 1999.

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.

Stephan Mandt, Matthew D. Hoffman, and David M. Blei. A Variational Analysis of Stochastic Gradient Algorithms. In *International Conference on Machine Learning*, page 8, feb 2016. URL `http://arxiv.org/abs/1602.02666`.

David Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Franziska Meier and Stefan Schaal. A probabilistic representation for dynamic movement primitives. *arXiv preprint arXiv:1612.05932*, 2016.

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv*, abs/1707.02201, 2017.

Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. Hierarchical visuomotor control of humanoids. *arXiv preprint arXiv:1811.09656*, 2018.

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BJl6TjRcY7`.

T Minka. Power ep. Technical report, Microsoft Research, 2004.

T. Minka. Divergence measures and message passing. Technical report, Microsoft Research Cambridge, January 2005. URL `https://www.microsoft.com/en-us/research/publication/divergence-measures-and-message-passing/`.

T P Minka. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2001a.

Thomas P. Minka. Expectation Propagation for approximate Bayesian inference. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., aug 2001b. ISBN 1-55860-800-1. URL `http://dl.acm.org/citation.cfm?id=647235.720257`.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pages 1791–1799, 2014.

Andriy Mnih and Danilo Rezende. Variational inference for monte carlo objectives. In *International Conference on Machine Learning*, pages 2188–2196, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems*, pages 3132–3140, 2015.

Jun Morimoto and Christopher G Atkeson. Minimax differential dynamic programming: An application to robust biped walking. In *Advances in neural information processing systems*, pages 1563–1570, 2003.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *arXiv*, abs/1702.08892, 2017.

T. Nagapetyan, A. B. Duncan, L. Hasenclever, S. J. Vollmer, L. Szpruch, and K. Zygalakis. The True Cost of Stochastic Gradient Langevin Dynamics. *arxiv*, jun 2017. URL `http://arxiv.org/abs/1706.02692`.

Eric Nalisnick, Lars Hertel, and Padhraic Smyth. Approximate inference for deep latent gaussian mixtures. In *NIPS Workshop on Bayesian Deep Learning*, 2016.

R. M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162, 2011. doi: 10.1201/b10905-6.

R. M. Neal and G.E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1999.

Radford M. Neal. Probabilistic inference using markov chain monte carlo methods, 1993.

A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding gradient noise improves learning for very deep networks. In *International Conference on Learning Representations (ICLR) Workshop*, 2016.

W. Neiswanger, C. Wang, and E. P. Xing. Asymptotically exact, embarrassingly parallel MCMC. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

Gerhard Neumann, Christian Daniel, Alexandros Paraschos, Andras Kupcsik, and Jan Peters. Learning modular policies for robotics. *Frontiers in computational neuroscience*, 8:62, 2014.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

M. Opper and O. Winther. Gaussian Processes for Classification: Mean-Field Algorithms. *Neural Computation*, 12(11):2655–2684, 2000.

M. Opper and O. Winther. Expectation consistent approximate inference. *Journal of Machine Learning Research*, 6:2177–2204, 2005.

Manfred Opper. A bayesian approach to on-line learning. In *On-line learning in neural networks*. Cambridge University Press, 1998.

George Papamakarios, Iain Murray, and Theo Pavlakou. Masked Autoregressive Flow for Density Estimation. *NIPS*, pages 2335–2344, 2017.

Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems*, pages 2616–2624, 2013.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Autodiff Workshop*, 2017.

S. Patterson and Y. W. Teh. Stochastic Gradient Riemannian Langevin Dynamics on the Probability Simplex. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

G. A. Pavliotis. *Stochastic processes and applications*, volume 60 of *Texts in Applied Mathematics*. Springer, New York, 2014. ISBN 978-1-4939-1322-0; 978-1-4939-1323-7. doi: 10.1007/978-1-4939-1323-7. URL `http://dx.doi.org/10.1007/978-1-4939-1323-7`. Diffusion processes, the Fokker-Planck and Langevin equations.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo CA, 1988.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *arXiv preprint arXiv:1804.02717*, 2018.

Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

Doina Precup, Richard S. Sutton, and Satinder P. Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 759–766, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2. URL `http://dl.acm.org/citation.cfm?id=645529.658134`.

R. Ranganath, S. Gerrish, and D. Blei. Black box variational inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

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

G Raskutti and S Mukherjee. The Information Geometry of Mirror Descent. *IEEE Transactions on Information Theory*, 61:1451–1457, 2015.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *RSS*, 2012. *Runner Up Best Paper Award*.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. *ICML*, pages 1530–1538, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.

C. P. Robert and G. Casella. *Monte Carlo statistical methods*. Springer Verlag, 2004.

Gareth O Roberts and Jeffrey S Rosenthal. General state space Markov chains and MCMC algorithms. *Probability Surveys*, 1:20–71, 2004. ISSN 1549-5787. doi: 10.1214/154957804100000024. URL `http://arxiv.org/abs/math/0404033`.

Gareth O Roberts, Richard L Tweedie, et al. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

Anirban Roychowdhury, Brian Kulis, and Srinivasan Parthasarathy. Robust monte carlo sampling using riemannian nosé-poincaré hamiltonian dynamics. In *International Conference on Machine Learning*, pages 2673–2681, 2016.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

T. Salimans, D. P. Kingma, and M. Welling. Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. *ArXiv e-prints*, October 2014.

J. M. Sanz-Serna and M.P. Calvo. *Numerical Hamiltonian problems*. Applied mathematics and mathematical computation. Chapman & Hall, 1994. ISBN 9780412542909. URL `https://books.google.co.uk/books?id=iUjvAAAAMAAJ`.

Lawrence K Saul and Michael I Jordan. Exploiting tractable substructures in intractable networks. In *Advances in neural information processing systems*, pages 486–492, 1996.

Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. In *International Symposium on Robotics Research, (ISRR)*, 2003.

Simon Schmitt, Jonathan J. Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting deep reinforcement learning. *arXiv:1803.03835*, 2018. URL `https://arxiv.org/abs/1803.03835`.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

John Schulman, Pieter Abbeel, and Xi Chen. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

S. L. Scott, A. W. Blocker, F. V. Bonassi, H. Chipman, E. George, and R. McCulloch. Bayes and big data: the consensus Monte Carlo algorithm. In *Bayes 250*, 2013.

Xiaocheng Shang, Zhanxing Zhu, Benedict Leimkuhler, and Amos J. Storkey. Covariance-Controlled Adaptive Langevin Thermostat for Large-Scale Bayesian Sampling. In *Advances in Neural Information Processing Systems*, pages 37–45, 2015.

Umut Şimşekli, Roland Badeau, A Taylan Cemgil, and Gaël Richard. Stochastic Quasi-Newton Langevin Monte Carlo. *ICML*, 2016.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

Suraj Srinivas and François Fleuret. Knowledge transfer with jacobian matching. *arXiv*, abs/1803.00443, 2018. URL `http://arxiv.org/abs/1803.00443`.

Xiaobai Sun and Christian Bischof. A basis-kernel representation of orthogonal matrices. *SIAM Journal on Matrix Analysis and Applications*, 16(4):1184–1196, 1995.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Csaba Szepesvári. *Algorithms for reinforcement learning*. Morgan & Claypool, 2010.

EG Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Esteban G Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913. IEEE, 2012.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1168–1175. IEEE, 2014.

Russ Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. 2009.

Y. W. Teh and M. Welling. The unified propagation and scaling algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, 2002.

Y. W. Teh, A. H. Thiéry, and S. J. Vollmer. Consistency and fluctuations for stochastic gradient Langevin dynamics. *Journal of Machine Learning Research*, 2015.

Yee Whye Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4499–4509, 2017.

T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *International Conference on Machine Learning (ICML)*, 2008.

T. Tieleman and G. Hinton. Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude, 2012. COURSERA: Neural Networks for Machine Learning.

Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In *International Conference on Machine Learning*, pages 1971–1979, 2014.

Emanuel Todorov and Zoubin Ghahramani. Unsupervised learning of sensory-motor primitives. In *Engineering in Medicine and Biology Society, 2003. Proceedings of*

the 25th Annual International Conference of the IEEE, volume 2, pages 1750–1753. IEEE, 2003.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

Jakub M Tomczak and Max Welling. Improving Variational Auto-encoders using Householder Flow. *arXiv preprint arXiv:1611.09630*, 2016.

Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1049–1056, 2009. ISBN 978-1-60558-516-1.

D. Tran, M. Hoffman, D. Moore, C. Suter, S. Vasudevan, A. Radul, M. Johnson, and R. A. Saurous. Simple, Distributed, and Accelerated Probabilistic Programming. *ArXiv e-prints*, November 2018.

Dustin Tran, Rajesh Ranganath, and David M Blei. The variational Gaussian process. *arXiv preprint arXiv:1511.06499*, 2015.

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.

R. E. Turner and M. Sahani. Two problems with variational expectation maximisation for time-series models. In D. Barber, T. Cemgil, and S. Chiappa, editors, *Bayesian Time Series Models*, chapter 5, pages 109–130. Cambridge University Press, 2011.

Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester Normalizing Flows for Variational Inference. In *Uncertainty in Artificial Intelligence*, 2018. URL `http://auai.org/uai2018/proceedings/papers/156.pdf`.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, koray kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. *NIPS*, pages 4790–4798, 2016.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

S. J. Vollmer, K. C. Zygalakis, and Y. W. Teh. Exploration of the (non-)asymptotic bias and variance of stochastic gradient Langevin dynamics. *Journal of Machine Learning Research*, 2016.

Martin J. Wainwright and Michael I. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning*, 2008. ISSN 1935-8237. doi: 10.1561/2200000001.

Bo Wang and DM Titterington. Inadequacy of interval estimates corresponding to variational bayesian approximations. In *AISTATS*. Barbados, 2005.

X. Wang and D. B. Dunson. Parallelizing MCMC via Weierstrass sampler, 2013. arXiv:1312.4605.

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*, pages 5320–5329, 2017.

M. Welling and Y.-W. Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, pages 681–688. Omnipress, 2011.

W. Wiegerinck and T. Heskes. Fractional belief propagation. *Advances in Neural Information Processing Systems (NIPS)*, 2003.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

T. Xifara, C. Sherlock, S. Livingstone, S. Byrne, and M. Girolami. Langevin diffusions and the Metropolis-adjusted Langevin algorithm. *Statistics & Probability Letters*, 91(C):14–19, 2014. URL `http://EconPapers.repec.org/RePEc:eee:stapro:v:91:y:2014:i:c:p:14-19`.

M. Xu, B. Lakshminarayanan, Y. W. Teh, J. Zhu, and B. Zhang. Distributed Bayesian posterior sampling via moment sharing. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pages 689–695, 2001.

A.L. Yuille. CCCP algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14(7):1691–1722, 2002.

S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. *Advances in Neural Information Processing Systems (NIPS)*, 2015a.

Y. Zhang, J. C. Duchi, and M. J. Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *Journal of Machine Learning Research*, 2015b.

T. Zhao, G. Cheng, and H. Liu. A partially linear framework for massive heterogeneous data. *The Annals of Statistics*, 44(4):1400–1437, 08 2016.

Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Machine Learning Department, Carnegie Mellon University, Dec 2010.