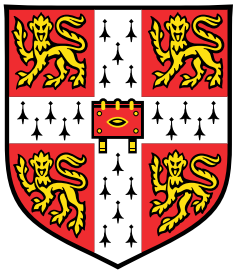
Bootstrap Your Flow



Laurence Illing Midgley

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Magdalene College

August 2021

I dedicate this thesis to my mom, dad, sisters, granny, Flo, Skollie, Taylor, Pippa, Phoebe and Glen beach.

Declaration

I, Laurence Illing Midgley of Magdalene College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. All software used in this report has been written by me using Python and PyTorch and the code repository is available at https://github.com/lollcat/FAB-MPHIL-2021. This dissertation contains 14962 words including appendices.

Laurence Illing Midgley
August 2021

Acknowledgements

I would like to thank my supervisors; Dr. José Miguel Hernández Lobato, Dr. Gregor Simm and Vincent Stimper. You guys were great! I felt like I had a constant source of advice and support whenever I needed it. Miguel: I really enjoyed our weekly meetings, especially towards the second half of the thesis when I was in less of a state of confusion. You had a great balance between directing me towards the most important components of the project, while giving me space to have fun and explore. Gregor: You gave me really good advice on how to generally approach my thesis. I also really appreciated the time you took during the beginning of thesis to sit and chat with me over some things I was confused by! Vincent: You gave me great tips on normalising flows and the Boltzmann distribution/Generators! You also pointed me to useful papers and code repositories that were a big help.

I would like to thank Kris for his friendship throughout the year. Getting to chat/argue about stuff really helped me build intuition of this work. Also, it was generally fun to frolic about Cambridge with you. I would like to thank my housemates Dey and Zac for ensuring that I subsisted on more than coffee each day, and for helping me get through my first quarantine.

Lastly, I would like to thank my family for sending pictures of the dogs.

Abstract

How can we approximate expectations over a target distribution (the target) that we cannot sample from? Two major approaches are Markov chain Monte Carlo (MCMC) and importance sampling. MCMC, the current state-of-the-art, generates samples in a Markov chain that converges to the target, which we can use for approximation by Monte Carlo (MC) estimation. To obtain unbiased estimates, MCMC has to converge, which often requires long simulations. In importance sampling, we rewrite the desired expectation to be over a proposal distribution (the proposal), allowing us to compute an unbiased MC estimate using samples from the proposal. If we can obtain a proposal similar to the target, this allows us to perform fast approximate inference. However, there are two challenges to such an approach: (1) if the proposal is not sufficiently expressive, it will not be able to capture the target's shape and (2) training the proposal is exceedingly difficult without samples from the target.

In this work, we combine importance sampling and MCMC in a method that leverages the advantages of both approaches. We use annealed importance sampling (AIS), whereby we generate samples from the proposal and then move, via MCMC, through a sequence of intermediate distributions to provide samples closer to the target. AIS preserves the ability to compute importance sampling estimates, while lowering the variance of this estimate (relative to only using the proposal). Furthermore, the MCMC transitions within AIS do not have to converge for this estimate to be unbiased, and therefore can be computationally cheaper than pure MCMC. Additionally, we use a normalising flow model (the flow) for the proposal, which is a highly expressive, parameterised distribution that has the potential to capture the shape of complex targets. Together the flow-AIS combination provides a way to generate samples close to the target, overcoming the expressiveness barrier to importance sampling methods. To train the flow-AIS combination, we propose FAB (normalising **F**low **A**IS **B**ootstrap): a novel training method that allows the flow and AIS to improve each other in a bootstrapping manner. We demonstrate that FAB can be used to produce accurate approximations to complex target distributions using toy problems (including the Boltzmann distribution), where conventional methods of training importance samplers fail.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

AIS Annealed Importance Sampling

BNN Bayesian Neural Network

DLVM Deep Latent Variable Model

FAB Normalising Flow Annealed Importance Sampling Bootstrap

HMC Hamiltonian Monte Carlo

MCMC Monte Carlo

MCMC Markov Chain Monte Carlo

MD Molecular Dynamics

ML Maximum Likelihood

MoG Mixture of Gaussians

VAE Variational Autoencoder

Mathematical Notation

We note the following notation used throughout this work. Vectors are denoted by bold lowercase letters (e.g. $\mathbf{x}$). Matrices are denoted as bold uppercase letters (e.g. $\mathbf{A}$). Vectors produced in a sequence of steps (e.g. in a Markov chain) are denoted with subscript (e.g. $\mathbf{x}_t$). Vectors within a batch are indicated with superscript (e.g. $\mathbf{x}^{(n)}$). Sequences are collectively referred to using colon superscript/subscript (e.g. $\mathbf{x}^{(1:N)}$, $\mathbf{x}_{1:T}$). Finally, we occasionally need to differentiate between vectors generated by different processes in the same space. We do this with a subscript. For example, if $\mathbf{x}$ is generated by some algorithm/process f, we may denote this as $\mathbf{x}_f$. Any deviations from this notation are directly noted in the text.

Chapter 1

Introduction

Probability distributions are useful for making predictions about the world. These predictions are typically expressed in terms of expectations of a function of interest $f(\mathbf{x})$, over a certain probability distribution $p(\mathbf{x})$

$$\mathbb{E}_{p(\mathbf{x})}\left[f(\mathbf{x})\right] \;=\; \int f(\mathbf{x})\; p(\mathbf{x})\; d\mathbf{x}. \tag{1.1}$$

In machine learning a classic example of this is making predictions using the posterior distribution over a model's parameters (e.g. inference with a Bayesian Neural Network). The main motivating example we consider in this work is inference involving the Boltzmann distribution, which describes the probability distribution of a system over states, in relation to the states' energy. This distribution has a wide variety of scientific applications, for example predicting the position of an amino acid sequence in space within a protein (protein folding).

In simple cases, for example if $p(\mathbf{x})$ is Gaussian, and $f(\mathbf{x}) = \mathbf{x}$, we can compute the desired expectation analytically. However, if $p(\mathbf{x})$ or $f(\mathbf{x})$ is more complex, such that exact computation is intractable, then we have to use approximate methods. If we can sample from $p(\mathbf{x})$, we can compute a Monte Carlo approximation of Equation 1.1 in the form

$$\mathbb{E}_{p(\mathbf{x})}\left[f(\mathbf{x})\right] \;\approx\; \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}^{(n)}) \qquad \text{where } \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \ldots \mathbf{x}^{(N)} \sim p(\mathbf{x}). \tag{1.2}$$

In situations where we cannot sample from $p(\mathbf{x})$, we cannot use Equation 1.2 directly and instead have to seek other methods. This is the case for both the Boltzmann distribution and Bayesian Neural Networks.

Markov chain Monte Carlo (MCMC) allows one to obtain samples from $p(\mathbf{x})$ by generating points in a sequence, with each new point dependent on the previous (i.e. a Markov chain), with $p(\mathbf{x})$ as the equilibrium distribution. These samples can then be used in Equation 1.2 as desired. In cases where $p(\mathbf{x})$ is high dimensional and/or has isolated modes, MCMC may require very long chains to explore the target distribution $p(\mathbf{x})$, at significant computationally cost.

Another way to calculate expectations when we cannot sample from $p(\mathbf{x})$ is importance sampling. Importance sampling utilises another distribution $q(\mathbf{x})$ from which we can directly draw samples, to compute an approximation to Equation 1.1 with an appropriate weighting to account for the fact that we are not sampling from the target distribution. The benefit of importance sampling over MCMC is that it allows the estimation to be performed in a "one shot", where we generate samples directly using $q(\mathbf{x})$ instead of via the long sequence of computations required for a Markov chain. However, the accuracy of this estimation depends on how similar our proposal distribution $q(\mathbf{x})$ is to $p(\mathbf{x})$, and obtaining a good proposal distribution becomes difficult when $p(\mathbf{x})$ is complex.

In this work we investigate the use of normalising flow models, which allow us to learn highly flexible distributions for $q(\mathbf{x})$, to obtain a good approximation to $p(\mathbf{x})$ that we can use for importance sampling. We focus on the challenging case where we have to train the model without access to samples from $p(\mathbf{x})$. We derive a robust training procedure for flow[1] models that uses annealed importance sampling (AIS), which along with other benefits, allows us to combine importance sampling with MCMC to obtain advantages both approaches. We focus on Boltzmann Generators, which are normalising flow models that are trained to approximate the Boltzmann distribution, as a motivating case study.

1.1 Contributions

- A review of the common approaches for approximate inference involving unnormalised distributions which we cannot sample from, with discussion of the Boltzmann distribution and Boltzmann Generators as a case study.
- Proposing FAB (normalising **F**low **A**nnealed importance sampling **B**ootstrap), a novel training method for flow models without reliance on samples from the target. FAB combines the flow proposal with AIS in a way that allows them to work together during training.

[1] We use the terms "normalising flow" and "flow" interchangeably for convenience, as this is common practice in literature.

- Demonstration of FAB's utility on simple toy problems. We show that FAB obtains superior performance to the conventional method[2] of training the flow model using samples from itself.
- Proposing and demonstrating that AIS can be used with a trained flow model to obtain a lower variance importance weighted estimator.
- Discussion of a set of key improvements for FAB, that are left to future work.

1.2 Overview

In Chapter 2, we survey the various approaches for approximate inference involving unnormalised distributions which we cannot sample from. We begin by examining approximate inference in an ideal case: using samples from the target distribution for Monte Carlo estimation, noting its key properties of unbiasedness and variance-shrinkage. We then examine the approaches for approximate inference in the situation where we cannot sample from the target, namely importance sampling and MCMC, and see how AIS links both methods. Finally, we introduce Boltzmann Generators, which use normalising flow models to approximate the Boltzmann distribution.

In Chapter 3, we propose FAB, a novel method for training normalising flow models by combining them with AIS. Before describing FAB we begin by examining suitable objectives for training proposal distributions on problems where we cannot sample from the target distribution, and only have access to the unnormalised probability density function. We show that minimising α-divergence, with $\alpha = 2$ is a sensible choice. We note that estimates of α-divergence (with $\alpha = 2$) using samples from the proposal will be high variance, thus training with these samples is unlikely to be successful. On the other hand, we show that estimates of α-divergence (with $\alpha = 2$) over the target distribution have far lower variance. We then present the FAB method, which uses AIS seeded with samples from the proposal flow model to (1) produce samples that are more similar to the target distribution, (2) use these samples to compute an importance weighted estimate of the gradient of α-divergence (with $\alpha = 2$) with respect to the flow model parameters, written as an expectation over the target distribution and (3) use this low(er) variance gradient to train the flow model parameters. We see that FAB provides a robust method of training, where AIS helps improve the proposal distribution, which in turn helps improve AIS, resulting in recursive self improvement. We

[2] By conventional method we mean the methods used in literature specifically in cases where we do not have access to samples from the target.

also discuss how AIS can be used on trained flow models to further improve the effective sample size.

In Chapter 4, we perform experiments to test the FAB method. We test FAB on a 2-dimensional mixture of Gaussians (MoG) problem, allowing visual inspection of the training process. We give the proposal flow model a poor initialisation to make this problem relatively challenging and show that FAB performs well, while conventional methods that train the proposal exclusively off its own samples fail. We then test FAB on a high dimensional version of the "Double Well" Boltzmann distribution ((Noé et al., 2019), showing that FAB is able to scale to more difficult problems. We also show that performing AIS on the trained flow model allows us to further boost the effective sample size.

In Chapter 5, we discuss related work. We discuss similarities between FAB and current approaches to training VAEs that combine MCMC with the encoder (proposal) to improve the samples sent to the decoder, and discuss some advantages that FAB may have in this application. We also compare FAB to Stochastic Normalising Flows (SNFs) (Wu et al., 2020), and discuss how FAB could be used to improve the training of SNFs for the case where we cannot sample from the target distribution.

In Chapter 6, we discuss various extensions to FAB with the goals of improving the sample efficiency, tuning of the AIS transition operator, and exploration. We note that our target distribution and proposal distribution allow us to express exploration in a simple intuitive manner.

Finally in Chapter 7, we provide a set of conclusions and discuss future work.

Chapter 2

Background

We begin this chapter by examining the ideal case of approximate inference, where we have samples from the target distribution. We then relate this to methods used in situations where we do not have access to these samples, namely importance sampling and MCMC. We provide an overview of these methods, noting their various strengths and weaknesses. Following this, we provide further discussion on the Boltzmann distribution and Boltzmann Generators, as this is the key example problem that this work focuses on. We aim to provide definitions for all required theory from first principles, and point to useful resources to refer to in literature.

2.1 Monte Carlo Estimation

Monte Carlo approximation allows us to approximate intractable integrals over $p(\mathbf{x})$ using samples from $p(\mathbf{x})$

$$\mathbb{E}_{p(\mathbf{x})}\left[f(\mathbf{x})\right] \approx \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}^{(n)}) \qquad \text{where } \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \ldots \mathbf{x}^{(N)} \sim p(\mathbf{x}). \tag{2.1}$$

This estimate is unbiased because

$$\begin{aligned} \mathbb{E}_{\mathbf{x}^{(n)}\sim p(\mathbf{x})}\left[\frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}^{(n)})\right] &= \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{\mathbf{x}^{(n)}\sim p(\mathbf{x})}\left[f(\mathbf{x}^{(n)})\right] \\ &= \mathbb{E}_{p(\mathbf{x})}\left[f(\mathbf{x})\right]. \end{aligned} \tag{2.2}$$

The variance in the estimate depends only on the variance of $f(\mathbf{x})$ over $p(\mathbf{x})$ and the number of samples in the estimator

$$\begin{aligned} \mathrm{Var}_{\mathbf{x}^{(n)}\sim p(\mathbf{x})}\left[\frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}^{(n)})\right] &= \frac{1}{N^2}\sum_{n=1}^{N}\mathrm{Var}_{\mathbf{x}^{(n)}\sim p(\mathbf{x})}\left[f(\mathbf{x}^{(n)})\right] \\ &= \frac{1}{N}\mathrm{Var}_{\mathbf{x}\sim p(\mathbf{x})}\left[f(\mathbf{x})\right]. \end{aligned} \tag{2.3}$$

Thus the variance in our estimator decreases at a rate of $\frac{1}{N}$ (i.e. the error in our estimate shrinks at a rate of $\frac{1}{\sqrt{N}}$). Notably, this is independent to the dimensionality of $p(\mathbf{x})$.

For complicated and high dimensional $p(\mathbf{x})$ it is often the case that only a few samples from $p(\mathbf{x})$ are required to obtain an accurate estimate using Equation 2.1 (Bishop, 2006). However, given that we are interested in problems where we cannot sample from $p(\mathbf{x})$, we cannot use the aforementioned method directly.

MCMC allows us to generate samples from $p(\mathbf{x})$, however the computational cost of generating these samples typically increases as the dimensionality and complexity of $p(\mathbf{x})$ increases. On the other hand, importance sampling allows us to use points generated with a proposal distribution $q(\mathbf{x})$ directly to perform estimation, however obtaining a good proposal distribution increases in difficulty as $p(\mathbf{x})$ increases in dimension and complexity. These key challenges are a central theme of this work.

2.2 Importance Sampling

Importance sampling allows us to approximate expectations over one (potentially unnormalised) distribution which we cannot sample from, using samples from another distribution that does allow us to generate samples. Importance sampling achieves this by expressing $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$ as an expectation over some proposal distribution $q(\mathbf{x})$ that we can sample from, and then computing a Monte Carlo estimate of this using samples from $q(\mathbf{x})$. We provide a brief explanation of how this is done below.

Let $\tilde{p}(\mathbf{x})$ be the unnormalised probability density function of the target distribution such that $\tilde{p}(\mathbf{x}) = Z_p p(\mathbf{x})$, where $Z_p = \int \tilde{p}(\mathbf{x})\, d\mathbf{x}$ is the normalisation constant/partition function. Let $q(\mathbf{x})$ define our proposal distribution such that $q(\mathbf{x}) \neq 0$ wherever $p(\mathbf{x}) \neq 0$. We can rewrite

$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$ to be over $q(\mathbf{x})$

$$\begin{aligned}\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] &= \frac{1}{\int \tilde{p}(\mathbf{x})\, d\mathbf{x}} \int f(\mathbf{x})\tilde{p}(\mathbf{x})\, d\mathbf{x} \\ &= \frac{1}{\int \frac{\tilde{p}(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x})\, d\mathbf{x}} \int \frac{\tilde{p}(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) f(\mathbf{x})\, d\mathbf{x} \\ &= \frac{1}{\mathbb{E}_{q(\mathbf{x})}\left[\frac{\tilde{p}(\mathbf{x})}{q(\mathbf{x})}\right]} \mathbb{E}_{q(\mathbf{x})}\left[f(\mathbf{x})\frac{\tilde{p}(\mathbf{x})}{q(\mathbf{x})}\right].\end{aligned} \tag{2.4}$$

Now that this is written in terms of expectations over $q(\mathbf{x})$ which we can sample from, we can compute a Monte Carlo estimate given by

$$\frac{1}{\mathbb{E}_{q(\mathbf{x})}\left[\frac{\tilde{p}(\mathbf{x})}{q(\mathbf{x})}\right]} \mathbb{E}_{q(\mathbf{x})}\left[f(\mathbf{x})\frac{\tilde{p}(\mathbf{x})}{q(\mathbf{x})}\right] \approx \frac{1}{\frac{1}{N}\sum_{n=1}^{N}\frac{\tilde{p}(\mathbf{x}^{(n)})}{q(\mathbf{x}^{(n)})}} \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x})\frac{\tilde{p}(\mathbf{x}^{(n)})}{q(\mathbf{x}^{(n)})} \qquad \mathbf{x}^{(1:N)} \sim q(\mathbf{x}). \tag{2.5}$$

To simplify notation we define *importance weights* $w(\mathbf{x}) = \tilde{p}(\mathbf{x})/q(\mathbf{x})$, so that we can write the result in Equation 2.5 compactly

$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] \approx \frac{\sum_{n=1}^{N} f(\mathbf{x}^{(n)})\, w(\mathbf{x}^{(n)})}{\sum_{n=1}^{N} w(\mathbf{x}^{(n)})} \qquad \mathbf{x}^{(n)} \sim q(\mathbf{x}). \tag{2.6}$$

Besides being sensitive to how $p(\mathbf{x})f(\mathbf{x})$ varies across space, the accuracy of this estimate depends on how well $q(\mathbf{x})$ approximates $p(\mathbf{x})$. Specifically if there are regions with significant probability density under $p(\mathbf{x})$ with low probability density under $q(\mathbf{x})$, then the variance in the importance weights (and therefore in the importance weighted estimate) will be high, with many samples required for an accurate estimate. Importance sampling typically has poor scaling to complex high dimensional distributions, as obtaining a sufficiently good proposal distribution becomes exceedingly difficult (Bishop, 2006). In Section 2.3 below, we consider how we can use normalising flow models, which are highly flexible to obtain good approximations for $p(\mathbf{x})$.

2.3 Normalising Flows

If our proposal distribution is governed by a set of tuneable parameters ϕ, then we can optimise ϕ to ensure our proposal distribution is as similar as possible to $p(\mathbf{x})$. If $p(\mathbf{x})$ is a complicated distribution, we require $q_\phi(\mathbf{x})$ to be highly expressive to imitate it. Otherwise, if our proposal is not sufficiently expressive, it will fail to capture the shape of the $p(\mathbf{x})$. For

example if $p(\mathbf{x})$ is multi-modal and we utilise a Gaussian proposal distribution with trained mean and covariance, then the expressiveness limitations of the Gaussian (i.e. uni-modal) will prevent $q_\phi(\mathbf{x})$ from being very similar to $p(\mathbf{x})$. Normalising flow models are a good candidate for $q_\phi(\mathbf{x})$ as these models are highly flexible, giving the potential to learn good approximations of complex target distributions.

Normalising flow models work through sampling from a simple distribution $\mathbf{z_0} \sim q_\phi(\mathbf{z}_0)$ (e.g. Multivariate Gaussian), and transforming it through a sequence of invertible, parameterised transformations $\mathbf{z}_t = g_t(\mathbf{z}_{t-1})$ for $t = 1, 2, \ldots, T$ to give an expressive probability distribution $q(\mathbf{x}) = q(\mathbf{z}_T)$ that we can use to approximate $p(\mathbf{x})$.

Normalising flows rely on the result that for an invertible, smooth transformation $\mathbf{z}_t = g(\mathbf{z}_{t-1})$, the resultant probability distribution is given by the change of variables formula

$$\begin{aligned} q(\mathbf{z}_t) &= q(\mathbf{z}_{t-1}) \left| \det \frac{d\mathbf{z}_{t-1}}{d\mathbf{z}_t} \right| \\ &= q(\mathbf{z}_{t-1}) \left| \det \frac{d\mathbf{z}_t}{d\mathbf{z}_{t-1}} \right|^{-1}, \end{aligned} \tag{2.7}$$

where the second line is a property of an invertible function's Jacobian. Thus, as long as the determinant of the Jacobian can be calculated, we can generate samples from our flow model, as well as evaluate probability densities given by

$$\begin{aligned} q(\mathbf{x}) &= q(\mathbf{z_T}) \\ &= \log q(\mathbf{z_0}) - \sum_{t=1}^{T} \log \left| \det \frac{d\mathbf{z}_t}{d\mathbf{z}_{t-1}} \right|, \end{aligned} \tag{2.8}$$

where we have stacked T flow layers sequentially.

We consider the RealNVP (Dinh et al., 2017) normalising flow transform as a simple illustrative example. Let $\mathbf{x} \in R^D$, $s : R^d \mapsto R^{D-d}$, $t : R^d \mapsto R^{D-d}$ where $d \geq 1$, then RealNVP is defined by the following simple *affine coupling* transformation

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp\left(s\left(\mathbf{x}_{1:d}\right)\right) + t\left(\mathbf{x}_{1:d}\right), \end{aligned} \tag{2.9}$$

where the subscript refers to elements of each vector and $\odot$ is the element-wise product. Typically we use a neural network for both of the functions s and t, allowing us to train the

RealNVP layer. The Jacobian for the RealNVP transform is given by

$$\frac{d\mathbf{y}}{d\mathbf{x}^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{d\mathbf{y}_{d+1:D}}{d\mathbf{x}_{1:d}^T} & \operatorname{diag}\left(\exp\left[s\left(\mathbf{x}_{1:d}\right)\right]\right) \end{bmatrix}. \tag{2.10}$$

Since the Jacobian is triangular, its determinant is given by the product of its diagonal entries. The inversion of the transformation simply given by

$$\begin{aligned} \mathbf{x}_{1:d} &= \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} &= \left(\mathbf{y}_{d+1:D} - \mathbf{t}\left(\mathbf{y}_{1:d}\right)\right) \odot \exp\left(-\mathbf{s}\left(\mathbf{y}_{1:d}\right)\right). \end{aligned} \tag{2.11}$$

So we see that RealNVP is a simple invertible function for which we can easily calculate the Jacobian determinant. This means that we can repeatedly stack this transformation, and obtain the probability density of points/samples using Equation 2.8. With each additional layer we obtain a more expressive probability distribution, with trainable parameters (the layer specific parameters of the functions s and t). In Figure 2.1 below we provide a visualisation of how a trained flow model (using RealNVP) produces increasingly more expressive probability distributions layer by layer.

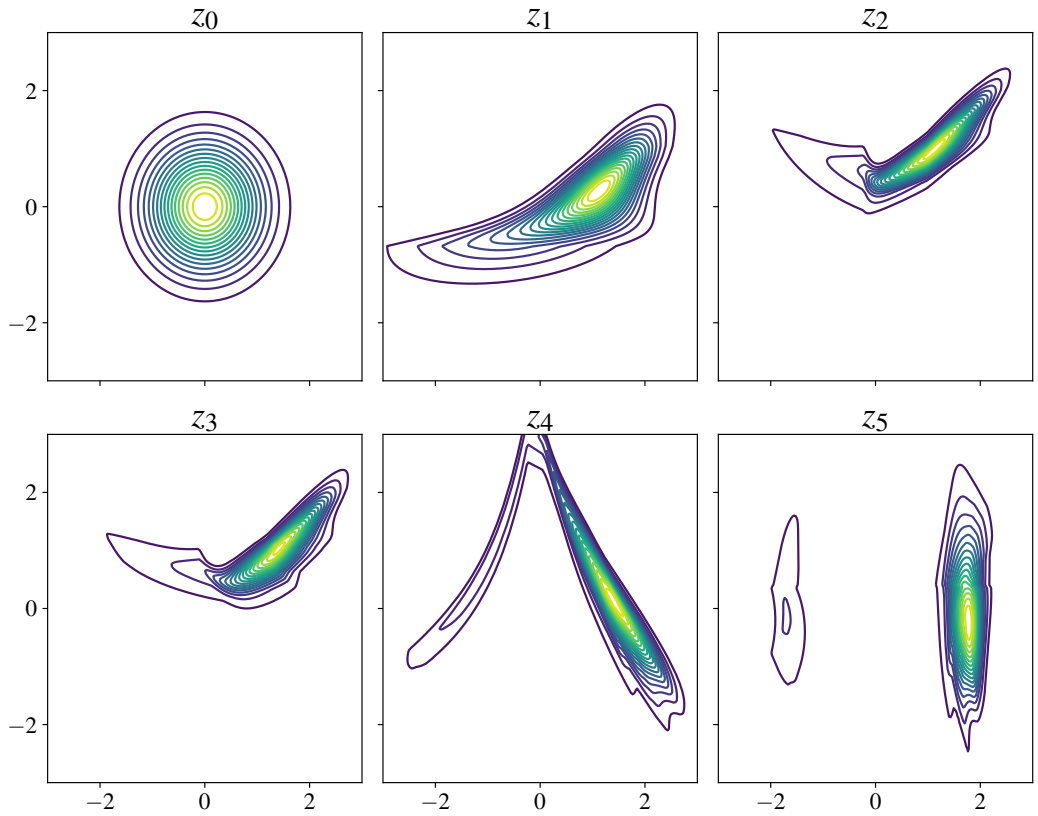


Fig. 2.1 Visualisation of a trained (RealNVP) normalising flow model layer by layer. On the top-left subplot we show the Gaussian base distribution probability contours. Each subsequent plot shows the probability distribution of each successive flow layer. We see that the probability distribution increases in complexity for each transform, such that the final (5^{th}) layer captures a multi modal shape. We flip the RealNVP coupling relationship (which element is copied vs transformed) in each layer. The target distribution is the Double Well Boltzmann distribution shown in Figure 4.5.

The most common training objective for flow models is to minimise the forward KL divergence between the target distribution and the flow model

$$\begin{aligned} L(\phi) &= \mathrm{KL}\left(p(\mathbf{x}) \parallel q_\phi(\mathbf{x})\right) \\ &= \mathbb{E}_{p(\mathbf{x})}\left[\log p(\mathbf{x}) - \log q_\phi(\mathbf{x})\right] \\ &= \mathbb{E}_{p(\mathbf{x})}\left[-\log q_\phi(\mathbf{x})\right] + \text{const}, \end{aligned} \tag{2.12}$$

which is computed with a Monte Carlo estimate using samples from $p(\mathbf{x})$. If we have a dataset of samples from $p(\mathbf{x})$ then this is equivalent to maximising the likelihood of the dataset under the flow model. In this work we are interested in the case where we do not

have access to such samples, and therefore do not use this loss function. We return to this in the next chapter.

Normalising flow models were first introduced by Tabak and Vanden-Eijnden (2010) and Tabak and Turner (2013). Rippel and Adams (2013) first parameterised flows with neural networks, which was improved upon by Dinh et al. (2014) who provided an efficient flow parameterised by neural networks. Normalising flows gained popularity after application to learning expressive distributions for variational inference (Rezende and Mohamed, 2015). RealNVP (as above) is based off an affine autoregressive transform, other noteworthy transforms for autoregressive flows include combination based transforms (De Cao et al., 2020; Ho et al., 2019; Huang et al., 2018), integration based transforms (Wehenkel and Louppe, 2019) and spline based transforms (Dolatabadi et al., 2020; Durkan et al., 2019a,b; Müller et al., 2019). Another family of flows (i.e. not autoregressive flow) are residual flows (Behrmann et al., 2019; Berg et al., 2018; Chen et al., 2019; Rezende and Mohamed, 2015; Tabak and Turner, 2013). We refer to Papamakarios et al. (2019) for an informative review of normalising flows, with discussion of their various families, expressive power and computational cost. There as been a recently increased interest in the use of flow models specifically for importance sampling, which falls within the category of "Neural Importance Sampling" (Müller et al., 2019).

2.4 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) is the standard method for generating samples from distributions that we cannot directly sample from. In this section we provide a description of how MCMC works, and then provide further discussion on Hamiltonian Monte Carlo, a version of MCMC that scales well to complex, high dimensional distributions.

In MCMC we sample a point $\mathbf{x}_0 \sim p(\mathbf{x}_0)$ from an initial sampling distribution, and then generate a sequence of points (Markov chain) $\mathbf{x}_1, \ldots, \mathbf{x}_N$, with each point only dependent on the previous, according to some transition operator $T(\mathbf{x}_{t+1} | \mathbf{x}_t) = p(\mathbf{x}_{t+1} | \mathbf{x}_t)$.

In order to produce samples from our target distribution, the Markov chain has to have two key properties: invariance and ergodicity. On a high level, invariance ensures that once we start generating samples from the target distribution, our Markov chain continues to generate samples from the same target distribution, while ergodicity ensures that when we start our Markov chain in some distribution different to our target, that we approach our target distribution (i.e. generating samples in proportion to their probability), if we run our chain for a sufficiently long period of time.

More technically, a distribution, $p^*(\mathbf{x})$ is invariant with respect to a Markov chain if applying the transition operator to samples from $p^*(\mathbf{x})$ returns samples from the same distribution

$$p^*(\mathbf{x}_{t+1}) \;=\; \int T\left(\mathbf{x}_{t+1} \,|\mathbf{x}_t\right) p^*(\mathbf{x}_t)\, d\mathbf{x}_t \tag{2.13}$$

A Markov chain is ergodic if the samples from the chain approach samples from $p^*(\mathbf{x})$ and no other distributions, as the length of the chain approaches infinity, regardless of the initial distribution. If both the conditions of invariance and ergodicity are met then $p^*(\mathbf{x})$ is referred to as the equilibrium distribution. Thus if we can construct a Markov chain with our desired target distribution as the equilibrium distribution, then we can obtain samples from our target distribution.

In practice it is often difficult to take big steps in transitions between states in the Markov chain, which means that (1) nearby samples may be highly correlated (autocorrelation), so we have to discard many samples to obtain uncorrelated points and (2) we have to run the Markov chain for very long before it returns samples from the equilibrium distribution, where assessment of convergence may be difficult. These issues are largely dependent on the choice of transition operator, where schemes that allow for big steps between points obtain far greater efficiency on complex target distributions. Below we examine one of the common, state-of-the-art algorithms for efficient MCMC: Hamiltonian Monte Carlo.

Bishop (2006) provides a clear introduction to MCMC and sampling methods in general. Brooks et al. (2011) provides a comprehensive and rigorous analysis of MCMC.

2.4.1 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) is a MCMC method inspired by Hamiltonian dynamics in physics, that allows for efficient exploration of complex high dimensional target distributions through the use of gradient information to inform both step direction and magnitude (Duane et al., 1987; Neal, 1995). Thus, we utilise HMC whenever we require MCMC in this work. Below we provide a description of HMC and discuss some key considerations in its use.

One can gain intuition for how Hamiltonian dynamics are useful for exploring a probability distribution by examining the following analogy: We consider a ball of unit mass rolling around a frictionless 3-D landscape with mass m, position $\mathbf{x}$ (in 2D not including height), momentum $\mathbf{v}$ and height h. The ball has kinetic energy related to its momentum, $K(\mathbf{v}) = \frac{|\mathbf{v}^2|}{2m}$, and potential energy relating to its position in space $U(\mathbf{x}) \propto h$. As the ball moves around the landscape it speeds up when it moves downhill (increasing kinetic energy, decreasing potential energy) and slows down when it moves uphill (decreasing kinetic energy and

increasing potential energy), where the total energy of the system (equal to the sum of the potential and kinetic energy) is conserved due to the law of the conservation of energy. HMC sets the the state $\mathbf{x}$ in a target distribution $p(\mathbf{x})$ to be defined as the a position variable, and momentum $\mathbf{v}$ to be an auxiliary variable, the mass to be equal to a unit, and potential energy to be equal to the negative log (potentially unnormalised) target

$$U(\mathbf{x}) \;=\; -\log \tilde{p}(\mathbf{x}). \tag{2.14}$$

We find that if we simulate the balls motion using Hamiltonian dynamics, this provides an efficient way of exploring our target distribution in a Markov chain that satisfies the necessary conditions required to have our target as its equilibrium distribution.

The momentum of a unit mass is given by the rate of change of position

$$\mathbf{v} \;=\; \frac{d\mathbf{x}}{dt} \tag{2.15}$$

where t is time. The acceleration (rate of change of momentum) is then given by the force exerted on the ball, which is equal to the negative gradient of the potential energy with respect to position

$$\frac{d\mathbf{v}}{dt} = -\frac{dU(\mathbf{x})}{d\mathbf{x}} \tag{2.16}$$

We define the Hamiltonian of the system as equal its total energy

$$\mathrm{H}(\mathbf{x},\mathbf{v}) = U(\mathbf{x}) + K(\mathbf{v}). \tag{2.17}$$

Using Equations 2.15 and 2.16, we can express the rate of change of position and momentum with respect to the Hamiltonian as

$$\frac{d\mathbf{x}}{dt} = \frac{dH}{d\mathbf{v}} \tag{2.18}$$

$$\frac{d\mathbf{v}}{dt} = -\frac{dH}{d\mathbf{x}} \tag{2.19}$$

Thus, we can simulate changes to the position and momentum variables according to Hamiltonian dynamics by integrating the above equations over time. It can be shown that these dynamics leave the joint distribution $p(\mathbf{x},\mathbf{v})$, defined by

$$p(\mathbf{x},\mathbf{v}) \;\propto\; \exp(-\mathrm{H}(\mathbf{x},\mathbf{v})) \tag{2.20}$$

invariant. To see why this is the case we consider the divergence of the flow field of Hamiltonian dynamics acting upon the joint space. The flow field (rate of change in position

in the joint space) is given by

$$\mathbf{V} = \left(\frac{d\mathbf{x}}{dt}, \frac{d\mathbf{v}}{dt}\right). \tag{2.21}$$

The divergence of this flow field is equal to 0

$$\begin{aligned} \operatorname{div} \mathbf{V} &= \sum_i \left[\frac{\partial}{\partial \mathbf{x}_i}\frac{\partial \mathbf{x}_i}{\partial t} + \frac{\partial}{\partial \mathbf{v}_i}\frac{\partial \mathbf{v}_i}{\partial t}\right] \\ &= \sum_i \left[-\frac{\partial}{\partial \mathbf{x}_i}\frac{\partial \mathrm{H}}{\partial \mathbf{v}_i} + \frac{\partial}{\partial \mathbf{v}_i}\frac{\partial \mathrm{H}}{\partial \mathbf{x}_i}\right] = 0, \end{aligned} \tag{2.22}$$

where subscripts indicate different dimensions of the vectors, and we plug in Equations 2.18 and 2.19 to obtain the second line. This implies that Hamiltonian dynamics preserve volume in the joint space, which implies that Hamiltonian dynamics leave the joint distribution invariant.

However, since H is constant throughout the evolution of the Hamiltonian dynamics, we see the joint distribution, which is a function of H, will have a constant value, and thus only a single probability contour of the joint distribution will be explored. Thus, running Hamiltonian dynamics is not ergodic. To ensure ergodicity, we simply define an additional transition that changes the value of H while maintaining invariance with respect to $p(\mathbf{x}, \mathbf{v})$. This can be done by introducing a Gibbs sampling step, where we sample momentum conditional on position. As momentum and position are independent, the conditional distribution for momentum is the same as its marginal distribution, which is a unit Gaussian. Thus if we follow Hamiltonian dynamics for fixed periods of time interspersed with the Gibbs update to momentum, we obtain a Markov chain with $p(\mathbf{x}, \mathbf{v})$ as its equilibrium distribution. We can then simply record the samples' position (discarding momentum) to obtain samples from the target distribution[1] $\mathbf{x} \sim p(\mathbf{x})$.

To implement this on a computer we have to approximate the integration of Hamiltonian dynamics. This is typically done using the "Leapfrog" integrator, with an additional Metropolis-hastings accept/reject step to correct for numerical errors (where we confirm whether a new point should be added to the Markov chain). The Leapfrog algorithm introduces two hyperparameter choices, namely an appropriate step size to compute changes to the system over, and the number of steps between Metropolis-hastings accept/reject steps. The performance of HMC has significant sensitivity with respect to these hyperparameters, and there

[1] Discarding samples from certain dimensions of a joint distribution provides samples from the marginal distribution of the non-discarded variables

are sophisticated methods for tuning them such as the "No-U Turn" sampler (Hoffman and Gelman, 2011).

An informative summary of HMC is given in Bishop (2006), while Neal (2011) and Betancourt (2017) provide a more in depth analysis of its properties, strengths and weaknesses.

2.5 Annealed Importance Sampling

Two downsides of standard MCMC methods is that they (1) can take long to explore target distributions with isolated modes, and (2) samples nearby in the Markov chain exhibit autocorrelation and therefore require discarding which can be hard to asses. Annealed importance sampling (AIS) deals with both of these issues in a method where we generate samples from a proposal distribution, and then move via MCMC through a sequence of intermediate distributions towards the target distribution (Neal, 2001). AIS conveniently returns both samples and importance weights which we can then use for importance sampling. The variance in the importance weights returned from AIS is lower than those from standard importance sampling. Therefore, AIS is a great candidate for application to the complex target distributions which we are interested in and it forms an integral part of the method which we propose in this work. Below we provide discussion of how AIS works.

For convenience within this section we refer to the target distribution as $p_N(\mathbf{x}) = p(\mathbf{x})$, and the proposal distribution as $p_0(\mathbf{x}) = q(\mathbf{x})$. To perform AIS we require a sequence of intermediate distributions, between the proposal and target, given by $p_1(\mathbf{x})$ to $p_{N-1}(\mathbf{x})$. AIS begins by sampling from the proposal distribution $p_0(\mathbf{x})$. We then use these samples to initialise a run of a Markov chain that leaves the intermediate distribution $p_1(\mathbf{x})$ invariant. We then repeat this process, each time using the previous samples to initialise the next Markov chain, for intermediate distribution $p_2(\mathbf{x}), p_3(\mathbf{x}), \ldots, p_{N-1}(\mathbf{x})$, where each intermediate distribution is more similar to the target distribution. The key to AIS is that movement in space during the intermediate transitions should be easier than if we ran a Markov chain straight from our proposal samples to the target distribution, where in the latter we expect to be more easily trapped in modes. This idea is based on Simulated Annealing, which seeks to find minima within a space by beginning with a high energy particle that moves freely around, exploring the space, after which the particle's energy is gradually decreased, such that its movement is more drawn to the nearest minima (Kirkpatrick et al., 1983).

To perform AIS we require that $\{p_i(\mathbf{x})\}_{i=0}^{N-1}$ satisfy $p_j(\mathbf{x}) \neq 0$ wherever $p_{j+1}(\mathbf{x}) \neq 0$, and that for each distribution we can calculate the unnormalised probability density $\tilde{p}_j(\mathbf{x}) \propto p_j(\mathbf{x})$.

There is flexibility in the choice of intermediate distributions to suite a specific problem, but Neal (2001) suggests

$$\tilde{p}_j(\mathbf{x}) = \tilde{p}_N(\mathbf{x})^{\beta_j}\ p_0(\mathbf{x})^{1-\beta_j} \tag{2.23}$$

where $0 = \beta_0 < \beta_1 < \ldots < \beta_N = 1$. We require that the Markov chain transitions T_j must leave the corresponding p_j invariant. Thus standard methods like the Metropolis algorithm or HMC may be used for these transitions. We do not require T_j satisfies the typical MCMC requirement for ergodicity, however this is preferred (Neal, 2001). Given that these conditions have been met, we can run the AIS sampling algorithm defined below

---

Generate $\mathbf{x}_0$ from p_0
Generate $\mathbf{x}_1$ from $\mathbf{x}_0$ using T_1
...
Generate $\mathbf{x}_{N-2}$ from $\mathbf{x}_{N-3}$ using T_{N-2}
Generate $\mathbf{x}_{N-1}$ from $\mathbf{x}_{N-2}$ using T_{N-1}

Listing 1 Sample generating steps for annealed importance sampling

The importance sampling weights for the final point $\mathbf{x}^{(i)} = \mathbf{x}_{N-1}$ can then be calculated using

$$w(\mathbf{x}^{(i)}) = \frac{\tilde{p}_1(\mathbf{x}_0)}{\tilde{p}_0(\mathbf{x}_0)} \frac{\tilde{p}_2(\mathbf{x}_1)}{\tilde{p}_1(\mathbf{x}_1)} \cdots \frac{\tilde{p}_{N-1}(\mathbf{x}_{N-2})}{\tilde{p}_{N-2}(\mathbf{x}_{N-2})} \frac{\tilde{p}_N(\mathbf{x}_{N-1})}{\tilde{p}_{N-1}(\mathbf{x}_{N-1})} \tag{2.24}$$

We provide a proof for why these importance weights are valid in Appendix A.1. These importance weights can then be easily used to compute approximations of expectations that are the form $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$. The annealing process ensures that the importance weights will have lower variance than standard importance sampling. Neal (2001) shows that in the case where we perform AIS with linear interpolation between the proposal and target according to Equation 2.23 and we assume that the transition operator produces independent states[2], then the variance in the log importance weights will asymptotically decrease according to σ_N^2/N as the number of interpolating distributions N increases.

Following the generation of point $\mathbf{x}_{N-1}$ using the steps from Listing 1 we can continue to run MCMC with $\mathbf{x}_{N-1}$ as a seed and $p_N(\mathbf{x})$ as the equilibrium distribution without any changes

[2]This is to simplify the proof, but is not required or recommended for AIS, Neal (2001) notes that in practice one should rather use a computationally cheap transition operator that does not come close to producing independent states, and rather expend compute on more intermediate distributions.

to the importance weights. This is the case because

$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] = \mathbb{E}_{p(\mathbf{x})}\left[\frac{1}{T}\sum_t f(\mathbf{x}_t)\right] \tag{2.25}$$

if $\mathbf{x}_t$ is generated by a process that leaves $p(\mathbf{x})$ invariant (Neal, 2001).

On a high level we can see AIS as providing a new higher quality proposal distribution by transforming samples from our proposal distribution with MCMC to look more like the target distribution[3], in a way that produces lower variance in the importance weights. Thus we see that AIS unites the two previously mentioned major methods of approximate inference (importance sampling and MCMC) into an approach that obtains benefits from both. It is also worth noting that the ease at which we can use our samples from AIS for expectation estimation is a significant advantage relative to standard MCMC, where assessing convergence and dealing with auto-correlation between samples is required before expectation computation.

2.6 α-divergence

Both standard importance sampling and AIS are strongly affected by the proposal distribution. Specifically, the variance in their importance weighted estimates is related to the degree of similarity between the proposal and target distributions. α-divergence defined by

$$D_\alpha(p\|q) = \frac{\int_x \alpha p(\mathbf{x}) + (1-\alpha)q_\phi(\mathbf{x}) - p(\mathbf{x})^\alpha q_\phi(\mathbf{x})^{1-\alpha} dx}{\alpha(1-\alpha)}. \tag{2.26}$$

is a general measure of similarity between distributions (Zhu and Rohwer, 1995). Two special cases are:

$$\lim_{\alpha\to 0} D_\alpha(p\|q) = \mathrm{KL}(q\|p) \tag{2.27}$$

$$\lim_{\alpha\to 1} D_\alpha(p\|q) = \mathrm{KL}(p\|q), \tag{2.28}$$

which we refer to as the reverse and forward KL-divergence respectively. This is useful for evaluating the quality of proposal distributions, which is of direct interest to this work. Further discussion of this is provided in the next chapter, for now we note the definition of α-divergence and some key behavior the measure exhibits. Different values of α control the degree of emphasis that the metric places on mode-seeking (small α) vs mode-covering (large

[3] AIS works well due to the freer movement under the intermediate distributions

α). This is illustrated in Figure 2.2 below. For all values of α, α-divergence is minimised if $p = q$. Further discussion of α-divergence can be found in Minka et al. (2005).

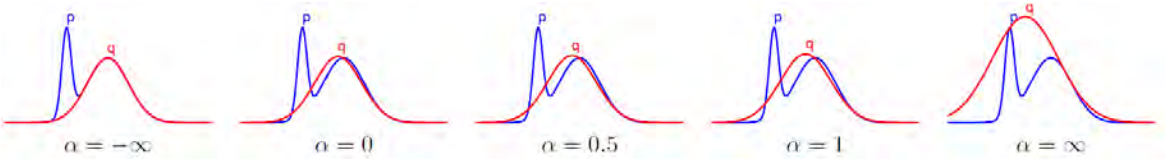


Fig. 2.2 Effect of different settings for α, when minimising α-divergence between Gaussian mixture p and Gaussian q, with respect to the mean and variance of q. High values of α encourage mode-covering, while small values of α encourage mode seeking. Figure taken from Minka et al. (2005).

2.7 The Boltzmann Distribution and Boltzmann Generators

The cornerstone example problem used in this thesis is the Boltzmann distribution. More specifically, we build upon the work by Noé et al. (2019) on Boltzmann Generators, which are normalising flow models that approximate the Boltzmann distribution. We provide a discussion of both the Boltzmann distribution and Boltzmann Generators below.

The Boltzmann distribution, defined by

$$p(\mathbf{x}) \propto e^{-u(\mathbf{x})}, \tag{2.29}$$

states that the probability that a system is in state $\mathbf{x}$ is inversely proportional to the exponential of the state's dimensionless energy $u(\mathbf{x})$. Thus, low energy states are more probable than high energy states. This is applicable to a wide variety of systems, such as configurations of atoms within a molecule, or the position of an amino acid sequence in space within a protein (i.e. protein folding). The Boltzmann distribution is useful because (1) knowing that a given system's state is likely is directly useful (e.g. knowing the most likely folded protein state is very useful) and (2) because it allows us to calculate properties (e.g. free energy differences) of a given system that are in the form

$$P = \mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]. \tag{2.30}$$

However, for non-trivial systems, there is no method to sample directly in "one shot" from the Boltzmann distribution, and enumeration of all possibilities is computationally intractable.

This means we cannot directly use the Boltzmann distribution for either of the above use-cases. Currently the most common approaches to this are MCMC (see Section 2.4) and Molecular Dynamics (MD), where the system starts in a given state and evolves in many small steps according to physical laws. These methods return samples approximately from $p(\mathbf{x})$ that we can then (1) directly inspect (e.g. look at the most probable sample), and (2) compute Monte Carlo approximations with (see Equation 2.1). However, given that the Boltzmann distribution typically has isolated modes ("meta-stable" states), these simulations often have to run for very long and are computationally expensive.

Through training a normalising flow model proposal distribution $q_\phi(\mathbf{x})$ to approximate the Boltzmann distribution, Boltzmann Generators offer an approach that allows the generation of samples in "one shot". These samples can then be re-weighted using importance sampling to calculate unbiased approximations to Equation 2.30.

To train the normalising flow model, Noé et al. (2019) utilises a training scheme that draws random samples from $\mathbf{x} \sim q(\mathbf{x})$ and minimises the reverse KL divergence between the target Boltzmann distribution and the Boltzmann generator, given by

$$\begin{aligned} \text{Loss}_{KL} &= \text{KL}(q_\phi || p) \\ &= \mathbb{E}_{q_\phi(\mathbf{x})} \left[\log q_\phi(\mathbf{x}) - \log p(\mathbf{x}) \right] \end{aligned} \tag{2.31}$$

Noé et al. (2019) notes that training on this alone focuses the Boltzmann generator on the most stable metastable state (i.e. mode clinging). This mode clinging behavior is a property of reverse KL divergence, and can be seen in Figure 2.2. To alleviate this Noé et al. (2019) includes an additional term in the loss function

$$\text{Loss} = \text{Loss}_{ML} + \text{Loss}_{KL}, \tag{2.32}$$

where Loss_{ML} utilises data $\mathbf{x}_{ML}$ generated from MD simulations, and is given by

$$\text{Loss}_{ML} = \mathbb{E}_{\mathbf{x}_{ML}} \left[\log q_\phi(\mathbf{x}) \right]. \tag{2.33}$$

Loss_{ML} encourages the flow model to place probability density on $\mathbf{x}_{ML}$, alleviating mode clinging. If we assume $\mathbf{x}_{ML} \sim p(\mathbf{x})$, then this is equivalent to minimising $\text{KL}(p||q_\phi)$. Training with Loss_{KL} is referred to as "training by energy", while the training using the term Loss_{ML} is referred to as "training by example". We see that this is the same as the previously introduced training objective commonly used for normalising flows (Equation 2.12). More recent work on Boltzmann Generators has continued to use a combination of both "training by energy" and

"training by example" (Dibak et al., 2021; Köhler et al., 2020; Wu et al., 2020). We note that the addition of the "training by example" term to some degree goes against the initial purpose of Boltzmann Generators, which was to shift from reliance on expensive MD/MCMC to a relatively computationally cheap flow model. In this case the flow model can still be useful to quickly augment a set samples from MD/MCMC by smoothing across the space. Noé et al. (2019) also notes two additional uses for Boltzmann Generators; (1) interpolating in the latent space of the Boltzmann generator can correspond to low energy pathways for transitions between different states and (2) by performing MCMC in the latent space of Boltzmann Generators, new states can be discovered. Our work focuses on improving the "training by energy" component of the work by Noé et al. (2019). We focus on the case where we do not have access to samples from the target distribution and therefore do not use Loss_{ML}. We seek to derive robust methods for training Boltzmann Generators in this context. Removing reliance on Loss_{ML} has a large benefit, as it requires expensive data from simulations which (partially) goes against the motivation behind Boltzmann Generators.

Chapter 3

Normalising Flow Annealed Importance Sampling Bootstrap

In this chapter, we introduce a new method that can be used to obtain accurate importance sampling estimates of expectations over unnormalised target distributions, without any reliance on samples from the target distribution. We propose FAB (normalising **F**low **A**nnealed importance sampling **B**ootstrap), which blends a flow model proposal distribution with AIS in a way that allows each of these components to improve the other.

Before describing the FAB method we first examine why α-divergence with $\alpha = 2$ provides a sensible choice of objective function for training importance samplers. We then discuss some important considerations in the practical estimation of this objective function. We note that the standard approach in literature for problems where we do not have access to samples from the target[1] is to estimate the proposals' training objective with samples from the proposal. This is the case for both neural importance samplers (Müller et al., 2019), and Boltzmann Generators (Noé et al., 2019).[2] We discuss why this approach of estimation of the proposals' training of objective using samples from the proposal is a poor choice for challenging problems. Following this ground work we present the FAB method, which uses AIS to obtain samples closer to the target distribution, which we use to obtain an improved estimate of the proposal training objective.

[1] We assume that we cannot directly sample from the target distribution, and do not have a dataset of samples from the target generated by some other mechanism (e.g. long MCMC simulation).

[2] For the scenario where we do not have access to samples from the target, Boltzmann Generators can only do "training by energy".

3.1 Choosing an Objective

We begin by re-iterating the problem that we focus upon in this work; we suppose that we are interested in computing expectations over $p(\mathbf{x})$ in the form $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$, where $f(\mathbf{x})$ is some function of interest. We are interested in cases where we cannot sample directly from $p(\mathbf{x})$ but can evaluate the un-normalised probability density function. To compute an approximation of the desired expectation, we can use importance sampling with some proposal distribution $q(\mathbf{x})$ that we can sample from

$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] \approx \frac{\sum_{n=1}^{N} f(\mathbf{x}^{(n)})\, w(\mathbf{x}^{(n)})}{\sum_{n=1}^{N} w(\mathbf{x}^{(n)})} \qquad \mathbf{x}^{(1:N)} \sim q(\mathbf{x}), \tag{3.1}$$

where $w(\mathbf{x}) = p(\mathbf{x})/q(\mathbf{x})$ are the importance weights. The accuracy of this estimate depends on the variance in the importance weights, which in turn depends on the similarity of the proposal distribution to the target distribution.

If our proposal distribution is governed by a set of tuneable parameters ϕ, then we can optimise ϕ to ensure our proposal distribution is as similar as possible to $p(\mathbf{x})$. The optimal proposal distribution is one that minimises the variance in the importance weighted estimate. Calculating this variance will require integration over $q_\phi(\mathbf{x})$ (which is typically intractable) and the result is dependent on the function $f(\mathbf{x})$. However, we are interested in finding good approximations for $p(\mathbf{x})$ that are independent to the choice of $f(\mathbf{x})$ - as it is more useful to have a approximation for $p(\mathbf{x})$ that we can re-use for a variety of different expectation functions, and we do not want an overly specific method. In this case we can consider an optimal proposal distribution to be one that has minimal variance in the importance weights.[3] In Appendix A.2, similarly to Minka et al. (2005), we provide a proof that minimising α-divergence (with $\alpha = 2$) is equivalent to minimising the variance in the importance weights. Thus, it is useful to select α-divergence as a measure of similarity measure between our proposal and target distributions.

[3] We note that this is also the optimal proposal distribution for calculating the normalising constant Z_p using importance sampling, which Minka et al. (2005) shows is equivalent to minimising $D_{\alpha=2}(p\|q)$.

We note that the first two terms in the numerator of the formula for $D_\alpha(p\|q)$ (Equation 2.26) integrate to constant values independent to ϕ, therefore

$$\begin{aligned} D_\alpha(p\|q) &\propto -\frac{\int p(\mathbf{x})^\alpha q_\phi(\mathbf{x})^{1-\alpha} d\mathbf{x}}{\alpha(1-\alpha)} \\ &\propto -\operatorname{sign}(\alpha(1-\alpha)) \int \frac{p(\mathbf{x})^\alpha}{q_\phi(\mathbf{x})^{\alpha-1}} d\mathbf{x} \\ &= \int \frac{p(\mathbf{x})^2}{q_\phi(\mathbf{x})} d\mathbf{x} \qquad \text{if } \alpha = 2. \end{aligned} \tag{3.2}$$

To train $q_\phi(\mathbf{x})$ we need access to a good estimate of a loss function proportional to $D_{\alpha=2}(p\|q)$ for the last line of the above equation. In the next section we provide some key considerations regarding this.

3.2 Estimating the Objective over the Proposal versus over the Target

$D_{\alpha=2}(p\|q)$ can be written as an expectation over either $p(\mathbf{x})$ or $q_\phi(\mathbf{x})$

$$D_{\alpha=2}(p\|q) \propto \int \frac{p(\mathbf{x})^2}{q_\phi(\mathbf{x})} d\mathbf{x} \tag{3.3}$$

$$= \mathrm{E}_{q_\phi(\mathbf{x})}\left[\frac{p(\mathbf{x})^2}{q_\phi(\mathbf{x})^2}\right] \tag{3.4}$$

$$= \mathrm{E}_{p(\mathbf{x})}\left[\frac{p(\mathbf{x})}{q_\phi(\mathbf{x})}\right]. \tag{3.5}$$

Therefore, if we have samples from either $p(\mathbf{x})$ or $q_\phi(\mathbf{x})$, we can compute a corresponding Monte Carlo estimate for our loss function. In the class of problems that we are interested in we only have samples from $q_\phi(\mathbf{x})$, however it is illuminating to consider both forms of this estimation.

If we look at the term inside the integral in Equation 3.3, we see that the most relevant samples for minimising the loss are those within regions of relatively high $p(\mathbf{x})$ and low $q_\phi(\mathbf{x})$. This means that samples from $p(\mathbf{x})$ provide a more accurate estimate of $D_{\alpha=2}(p\|q)$ than samples from $q_\phi(\mathbf{x})$, as they are more likely to contribute significantly to the expectation.

We illustrate this with a simple example, where we set $p(x)$ and $q(x)$ to be 1D unit Gaussians with means -1 and 1 respectively, and compare MC estimates of $D_{\alpha=2}(p\|q) \propto \int \frac{p(x)^2}{q_\phi(x)} dx$

using samples from $p(x)$ versus $q(x)$. The results of this are shown in Figure 3.1, where in the right most subplot we see that the estimate over $p(x)$ is exceedingly more accurate than the estimate over $q(x)$. For example, using 10^3 samples under $p(x)$ is better than using 10^6 samples from $q(x)$.

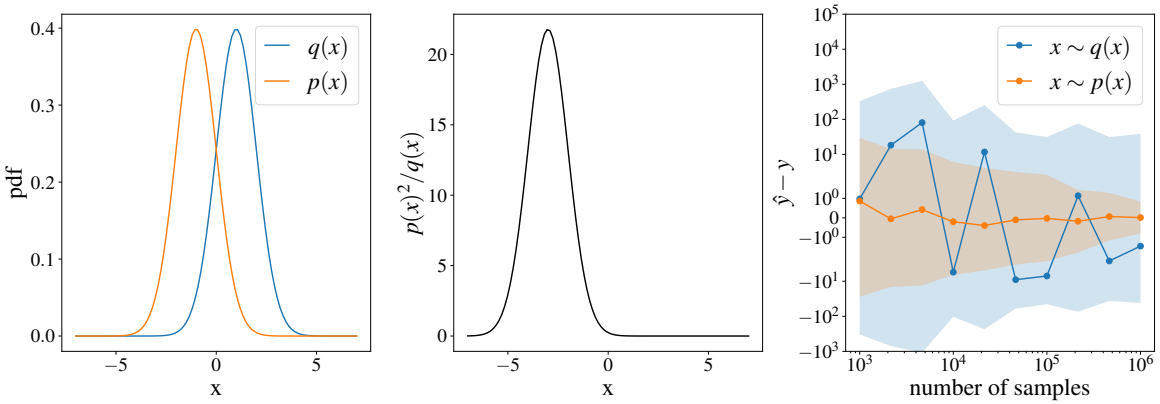


Fig. 3.1 Estimating $\int p(x)^2/q_\phi(x)\, dx \propto D_{\alpha=2}(p\|q)$ over the target vs proposal distribution. **LHS**: Plot of target and proposal distribution probability density functions; **Mid**: Plot of $\frac{p(\mathbf{x})^2}{q_\phi(\mathbf{x})}$, we wish to approximate the integral of this function using an MC estimate over p or q; **RHS**: The integral is tractable (for Gaussian p and q), so we can compute the exact value of the loss (y), and compare this to estimates ($\hat{y}$). The plot shows the average and 95% confidence interval (across 100 runs) for the difference between the y and $\hat{y}$ using p vs q. The MC estimate over p is far more accurate (tighter interval) than over q.

Not only is $q_\phi(\mathbf{x})$ worse than $p(\mathbf{x})$ for estimating the loss, it is a poor candidate in an absolute sense for problem scenarios where our target distribution is high dimensional/complex. Because $q_\phi(\mathbf{x})$ is initially dissimilar to $p(\mathbf{x})$, if we compute an MC estimate for our loss proportional to $D_{\alpha=2}(p\|q)$, we see that most samples will have high $q_\phi(\mathbf{x})$ and low $p(\mathbf{x})$, and most terms in the MC estimate will be close to zero. Furthermore, there may be points that are important contributors to the true expectation of $D_{\alpha=2}(p\|q)$, that have have a negligible chance of ever appearing in the MC estimate of the loss. In a pathological case, we can imagine a region with significant probability density under $p(\mathbf{x})$ but with extremely low probability density under $q_\phi(\mathbf{x})$, such that there are unlikely to ever be samples under $\mathbf{x} \sim q_\phi(\mathbf{x})$ within this region. In this case, if we were to train using an MC-estimate of Equation 3.3, because we never see samples in this region, ϕ will never be updated for $q_\phi(\mathbf{x})$ to place more density in this region. Having a proposal distribution that does sample from this region would be crucial for obtaining accurate importance sampling estimates, and thus the training scheme would catastrophically fail.

In literature we typically see that expressive approximating distributions are trained to mimic complex target distributions by training on samples from the target distribution in a supervised learning fashion. For example, a common method of training deep generative models is to maximise the likelihood of samples from $p(\mathbf{x})$ under $q_\phi(\mathbf{x})$, which is equivalent to minimising the forward KL divergence between the target distribution and the model (Papamakarios et al., 2019). In the case where $p(\mathbf{x})$ is the Boltzmann distribution (i.e. $q_\phi(\mathbf{x})$ is a Boltzmann Generator), literature typically uses a combination of samples from $p(\mathbf{x})$ and $q_\phi(\mathbf{x})$ for training[4], where samples from $p(\mathbf{x})$ are obtained from computationally expensive methods (e.g. MD simulations) (Noé et al., 2019). As we are focusing on the case where we do not have access to samples from $p(\mathbf{x})$, we have to grapple with the difficulty of obtaining a good training procedure without reliance on these samples.

FAB, the training method proposed in this thesis, aims to provide a robust method for training $q_\phi(\mathbf{x})$ in this context.

3.3 FAB Method

FAB introduces AIS into the training loop of the proposal distribution $q_\phi(\mathbf{x})$, improving the gradient estimator for minimising α-divergence with respect to ϕ by writing the loss function to train our flow as an expectation over $p(\mathbf{x})$, and estimating it with the samples and importance weights generated by AIS. We show how this can be done below.

First, we note that the α-divergence (see Equation 3.2) is proportional to the following expectation over $p(\mathbf{x})$

$$\begin{aligned} D_\alpha(p\|q_\phi) &\propto \operatorname{sign}(\alpha(\alpha-1)) \int p(\mathbf{x})^\alpha q_\phi(\mathbf{x})^{1-\alpha}\, d\mathbf{x} \\ &\propto \operatorname{sign}(\alpha(\alpha-1))\ \mathbb{E}_{p(\mathbf{x})}\left[\frac{p(\mathbf{x})^{\alpha-1}}{q_\phi(\mathbf{x})^{\alpha-1}}\right]. \end{aligned} \tag{3.6}$$

[4]The previous reasoning in this section makes us sceptical of the utility of the "training by energy" component of Boltzmann generators, where we expect that early in training, most of the heavy lifting is done by the "training by example loss". The "training by energy" term can help push the Boltzmann generators probability mass down in areas where the target has low probability mass (as reverse KL has the property of "zero-forcing" (Minka et al., 2005)), however this is much less important for high quality importance samplers than having probability mass in regions with high mass under the target. We note that if p and q are similar, then estimates of KL/α-divergence over q are reasonably accurate, however we are interested in difficult problems where this will not be true early in training.

To maximise our objective (and minimise $D_\alpha(p\|q_\phi)$) we obtain a gradient estimator by differentiating with respect to the parameters ϕ of the flow model

$$\nabla_\phi \left[c \, \mathrm{E}_{p(\mathbf{x})} \left[\frac{p(\mathbf{x})^{\alpha-1}}{q_\phi(\mathbf{x})^{\alpha-1}} \right] \right] \; = \; c \, \mathrm{E}_{p(\mathbf{x})} \left[\nabla_\phi \frac{p(\mathbf{x})^{\alpha-1}}{q_\phi(\mathbf{x})^{\alpha-1}} \right], \tag{3.7}$$

where we have set[5] $c = -\operatorname{sign}(\alpha(\alpha-1))$ and have used the fact that since ϕ is independent to samples from $p(\mathbf{x})$, we can move ∇_ϕ inside the expectation. If we set $f(\mathbf{x}) = \nabla_\phi \frac{p(\mathbf{x})^{\alpha-1}}{q_\phi(\mathbf{x})^{\alpha-1}}$, we see that Equation 3.7 is in the form $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$, so we can estimate it with AIS.

With this goal in mind, during the training loop, we generate a batch of importance weights $w_{\mathrm{AIS}}^{(1:L)}$, and samples $\mathbf{x}_{\mathrm{AIS}}^{(1:L)}$ using AIS, with $p(\mathbf{x})$ as the target distribution and $q_\phi(\mathbf{x})$ as the proposal distribution. We can then obtain an importance weighted estimate of the above gradient operator

$$c\mathrm{E}_{p(\mathbf{x})} \left[\nabla_\phi \frac{p(\mathbf{x})^{\alpha-1}}{q_\phi(\mathbf{x})^{\alpha-1}} \right] \; \approx \; c \sum_{l=1}^{L} \bar{w}_{\mathrm{AIS}}^{(l)} \left[\nabla_\phi \frac{p(\bar{\mathbf{x}}_{\mathrm{AIS}}^{(l)})^{\alpha-1}}{q_\phi(\bar{\mathbf{x}}_{\mathrm{AIS}}^{(l)})^{\alpha-1}} \right]. \tag{3.8}$$

We note that in Equation 3.7, $q_\phi(\mathbf{x})$ is only differentiated through ϕ's contribution to the probability density function, and not[6] via $\nabla_\phi x$. Therefore, in Equation 3.8 we take care to block the gradient of $x_{\mathrm{AIS}}^{(1:L)}$ with respect to ϕ. We denote the blocked gradients with $\bar{x}_{AIS}^{(l)}$. Equation 3.8 provides an unbiased estimate of the gradient operator, because AIS provides unbiased estimates over the target distribution. As we noted in Section 3.2, estimation of the training objective (minimising $D_{\alpha=2}(p\|q)$) has far lower variance when using samples from $p(\mathbf{x})$ instead of $q_\phi(\mathbf{x})$. Therefore, Equation 3.8 provides a lower variance gradient operator than an MC estimate of the gradient operator using samples from $q_\phi(\mathbf{x})$. Thus, we can train the proposal by maximising the surrogate "objective function"

$$O(\phi) = c \sum_{l=1}^{L} \bar{w}_{\mathrm{AIS}}^{(l)} \left[\frac{p(\bar{\mathbf{x}}_{\mathrm{AIS}}^{(l)})^{\alpha-1}}{q_\phi(\bar{\mathbf{x}}_{\mathrm{AIS}}^{(l)})^{\alpha-1}} \right], \tag{3.9}$$

taking care[7] to block the gradient of $\bar{w}_{\mathrm{AIS}}^{(1:L)}$ and $\bar{\mathbf{x}}_{\mathrm{AIS}}^{(1:L)}$ with respect to ϕ.

[5] We keep track of the sign of $\operatorname{sign}(\alpha(\alpha-1))$, using c as a coefficient for all objective functions to make sure we are correctly maximising/minimising.

[6] In Equation 3.7, ∇_ϕ is inside the expectation, with $x \sim p(\mathbf{x})$ independent to ϕ.

[7] In Equation 3.8, $w_{\mathrm{AIS}}^{(1:L)}$ is not differentiated with respect to ϕ, so we must block the gradient of $w_{\mathrm{AIS}}^{(1:L)}$ with respect to ϕ (e.g. with `torch.no_grad`($w_{\mathrm{AIS}}^{(1:L)}$)), as otherwise automatic differentiation will result in an incorrect estimate of the gradient. This is because the flow model parameters ϕ participate in the calculation of $w_{\mathrm{AIS}}^{(1:L)}$ and $x_{\mathrm{AIS}}^{(1:L)}$.

To obtain a good objective function for training it is beneficial to instead seek to write the surrogate objective (Equation 3.9) in terms of log probabilities and log importance weights, because inside the expectation the importance weights and fractions of probabilities will have high variance. To do this we can re-write the surrogate objective as

$$\begin{aligned} O(\phi) &= c \exp \log \sum_{l=1}^{L} \bar{w}_{\mathrm{AIS}}^{(l)} \left[\frac{p(\bar{\mathbf{x}}_{\mathrm{AIS}}^{(l)})^{\alpha-1}}{q_\phi(\bar{\mathbf{x}}_{\mathrm{AIS}}^{(l)})^{\alpha-1}} \right] \\ &= c \exp \log \sum_{l=1}^{L} \exp \Bigg(\log \bar{w}_{\mathrm{AIS}}^{(l)} + \\ &\qquad (\alpha-1) \left(\log p(\bar{\mathbf{x}}_{\mathrm{AIS}}^{(l)}) - \log q_\phi(\bar{\mathbf{x}}_{\mathrm{AIS}}^{(l)}) \right) \Bigg). \end{aligned} \tag{3.10}$$

We then instead maximise the $\log O(\phi)$, which by Jensen's inequality[8] gives us an lower bound of the previous objective.

$$\begin{aligned} \mathscr{L}(\phi) &= c \log \sum_{l=1}^{L} \exp \Bigg(\log \bar{w}_{\mathrm{AIS}}^{(l)} + \\ &\qquad (\alpha-1) \left(\log p(x_{\mathrm{AIS}}^{(l)}) - \log q_\phi(x_{\mathrm{AIS}}^{(l)}) \right) \Bigg) \end{aligned} \tag{3.11}$$

We can now work with log probabilities and log importance weights, and use the "logsumexp" trick to obtain a numerically stable estimate. Equation 3.11 is the exact surrogate objective[9] implemented in practice for training, setting $\alpha = 2$ for the aforementioned reasons.

[8] $\mathbb{E}_{p(\mathbf{x})}[\log f(\mathbf{x})] \leq \log \mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$

[9] We note that the following gradient estimators from Müller et al. (2019) can also be used with FAB (where we use AIS instead of IS): (1) Minimising the forward KL-divergence using $\nabla_\phi \mathrm{KL}(p \mid\mid q) \eqsim \sum_{l=1}^{L} \bar{w}_{\mathrm{AIS}}^{(l)} \nabla_\phi \log q_\phi(\mathbf{x})$ and (2) minimising $D_{\alpha=2}(p\|q)$ using $\nabla_\phi \, D_{\alpha=2}(p\|q) \eqsim \sum_{l=1}^{L} (\bar{w}_{\mathrm{AIS}}^{(l)})^2 \, \nabla_\phi \log q_\phi(\mathbf{x})$. In practice we found these estimators to be more unstable than Equation 3.11.

This leads to the following algorithm for training the flow model proposal distribution:

Algorithm 1: FAB for minimisation of $D_{\alpha=2}(p \parallel q_\phi)$

Set target p
Initialise proposal q_ϕ
for *iteration = 1, M* **do**
  Sample batch $\mathbf{x}_q^{(1:L)}$, $\log q(\mathbf{x}_q^{(1:L)})$ from q_ϕ
  Generate batch $\mathbf{x}_{AIS}^{(1:L)}$, $\log w_{AIS}^{(1:L)}$ from AIS seeded with $\mathbf{x}_q^{(1:L)}$, $\log q(\mathbf{x}_q^{(1:L)})$
  Calculate FAB objective:
  $\mathscr{L}(\phi) = \log \sum_{l=1}^{L} \exp\left(\log \bar{w}_{\text{AIS}}^{(l)} + \left(\log p(\mathbf{x}_{\text{AIS}}^{(l)}) - \log q_\phi(\mathbf{x}_{\text{AIS}}^{(l)})\right)\right)$
  Perform gradient descent on $\mathscr{L}(\phi)$
end

This method obtains the benefit of bootstrapping, where AIS is used to improve the proposal flow model (by improved estimation of the gradient of the loss function, which is used to update ϕ) which then improves AIS (by improving its initial distribution). So the AIS and flow model work together, where improving one helps improve the other and visa versa.

3.4 Further Remarks

In addition to updating the parameters of $q_\phi(\mathbf{x})$, we may also tune parameters of AIS during training. Specifically, if we utilise Hamiltonian Monte Carlo as the transition operator for AIS, then we can tune the step size parameters ε. We have the option of using a different step sizes for different dimensions of $\mathbf{x}$, and for each intermediate distribution used in AIS. The performance of HMC is highly sensitive to its hyperparameters, and thus this tuning is useful during training, as it allows us to improve the effectiveness of the AIS.

AIS is also useful **after training**; if the flow model has not converged to a very high effective sample size, then rather than directly sampling from the trained flow model, we can use AIS in conjunction with the trained flow model to reduce the variance in the importance weights, improving the effective sample size. In this case, AIS after training can utilise the tuned AIS parameters from training, either directly, or as an initialisation for further tuning.

Chapter 4

Experiments

In this chapter, we demonstrate FAB's effectiveness through experiments on a set of toy problems. We also show that AIS is useful after training, to further improve the accuracy of estimates over a trained flow proposal distribution.

We begin with a simple 2D mixture of Gaussian problem that allows us to easily visualise the progress of the model throughout training. On this problem we show that FAB outperforms the typical method used to train Boltzmann Generators/neural importance samplers without access to samples from the target (i.e. training exclusively off samples from the proposal). We then test FAB on the high dimensional, multi-modal "Many Well" Boltzmann distribution to demonstrate that it is robust on more challenging problems. Finally, we test the use of AIS on an already-trained flow model, and confirm a strong improvement in effective sample size. Before diving into experiments, we begin by considering how best to measure performance, and note some general hyperparameter/sub-method choices that we make.

4.1 Measuring Performance

Measuring performance is very difficult for the class of problems that we focus on. In this section we define effective sample size, the main measure we use for performance, and discuss the importance of taking precautions if we desire to use it accurately.

Effective sample size is defined by

$$\text{ESS} = \frac{1}{N \sum_{n=1}^{N} \hat{w}(\mathbf{x}^{(n)})^2} \tag{4.1}$$

where $\hat{w}(\mathbf{x})$ is the normalised importance weight. This statistic tells us how many samples we effectively have from $p(\mathbf{x})$. For example, if our effective sample size is 0.5 then generating 2 samples using our method, should be equivalent to obtaining 1 sample from $p(\mathbf{x})$. We see that in the case where $q = p$, all of the normalised importance weights are equal to $1/N$, which if we plug into Equation 4.1, gives an effective sample size of 1. This is the standard choice used in literature for both MCMC and importance sampling, and is therefore well suited to the methods of interest in this work (Martino et al., 2017). It is however easy for the effective sample size estimate to be spurious. Specifically, if we have zones with significant probability mass under $p(\mathbf{x})$ with negligible mass under $q(\mathbf{x})$, then the true ESS is very low. However, since we are unlikely to see samples from this region when we sample from $q(\mathbf{x})$, the effective sample size will appear artificially high. To guard against this, within each problem we perform further checks, including both visual checks as well as additional metrics, to ensure that we can "trust" the effective sample size statistic.

4.2 Choices for Sub-methods and Hyperparameters

Below we provide a brief discussion of choices for hyperparameters and sub-methods within FAB that are used across all experiments within this chapter. The effect of the quality of the transition operator in AIS has a big effect (see Section 4.5), and therefore the selection of transition operator, and tuning method for the parameters of the transition operator is an importance consideration. Because AIS is contained inside the training loop, if the transition operator (or tuning of transition operator parameters) is computationally expensive then this will slow down training significantly. It is therefore a key required property for the transition operator to be efficient, and the tuning method to have low computational overhead. HMC is used as the transition operator in all experiments, as it is generally able to explore complex high dimensional distributions in a relatively efficient manner. Neal (2001) recommends that for AIS, on the margin compute is better allocated to more interpolating distributions than to better transitions between distributions, such that one should typically use relatively lower quality transitions that do not produce true samples from the intermediate target. Thus we utilise only one outer-loop, and five inner (leap-frog) steps of HMC, and in cases where we need better performance from AIS we simply increase the number of intermediate distributions. This choice of a relatively short inner loop and a single outer-loop is common in literature when using HMC with AIS (for example Wu et al. (2017)).

To adjust HMC's step size we use the simple heuristic of aiming for an average Metropolis acceptance probability near 65%. This is commonly used and is supported to be optimal by theory in literature (Beskos et al., 2010; Neal, 2011). To tune the step size to achieve

this we use the simple heuristic of multiplying the step size by 1.1 when the average (per batch) acceptance probability is below 65%, and divide the step size by 1.1 when the average acceptance probability is above 65%. For the target distributions used in this chapter, the length scales for different dimensions are relatively similar - so having a shared HMC step size across dimensions is justified. However, we do tune a separate step size for each intermediate AIS distribution. This method of tuning a shared step size is robust, and has very low computational overhead.

We utilise linear spacing for intermediate distributions, interpolating between the proposal and target distribution. Neal (2001) recommends geometric spacing, however in this section we generally use a relatively small number of intermediate distributions, for which linear spacing is simpler to use[1]. In AIS, after travelling through the intermediate distributions, one may run a Markov chain that leaves the target distribution invariant, without any changes to the importance weights, however we do not include this.

We did not do extensive hyperparameter searches, and hyperparameter selection is performed using the heuristic of picking a hyperparameter that on face value seems like it should lead to fast training. For example, when picking the number of intermediate AIS distributions we typically try to pick the smallest number of distributions that allowed for progress to be made during training (i.e. for the FAB bootstrap to lead to short term improvements in the effective sample size during the start of training). We leave study of the effects of these hyperparameters to future work. For each set of experiments we provide descriptions of key high level design choices, and provide further details in the Appendix C, as well as providing our code at https://github.com/lollcat/FAB-MPHIL-2021.

4.3 Demonstration on Simple Mixture of Gaussian Problem

To investigate FAB, we begin with a simple two dimensional mixture of 4 diagonal-covariance Gaussians as the target distribution $p(\mathbf{x})$. Figure 4.1 shows the initial problem setting, where we have given the proposal distribution a poor initialisation, such that it places a very low probability mass on 3 of the modes. This is in accordance with the example of the pathological case discussed at the end of Section 3.2, in which we expect methods which directly use

[1] If we have a small number of distributions, we would have to put more thought into how to ensure our geometric spacing is good (e.g. where to place the first intermediate distribution). Linear spacing on the other hand is automatic and easy.

samples from the proposal distribution to train the proposal distribution to perform poorly. The main point of this section is to provide a visual illustration of how FAB works.

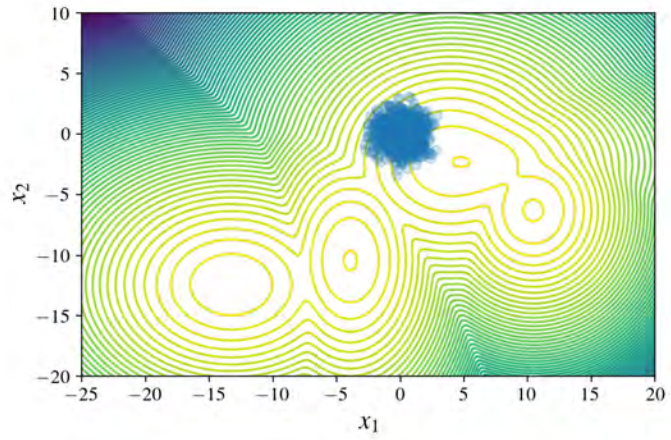


Fig. 4.1 Samples (in blue) from the initialised proposal distribution (flow model) vs log probability density function contours of the target MoG distribution. The proposal distribution has a poor initialisation placing low probability mass on the modes of the target distribution.

To estimate expectations $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$, with our proposal distribution, we set $f(\mathbf{x})$ to be the following toy quadratic function

$$f(\mathbf{x}) \;=\; \mathbf{a}^T(\mathbf{x}-2\mathbf{b}) + 2(\mathbf{x}-2\mathbf{b})^T \mathbf{C}(\mathbf{x}-2\mathbf{b}), \tag{4.2}$$

where the elements of $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{C}$ are sampled from a unit Gaussian and then fixed for the problem. This allows us to inspect the bias and variance of estimates of the expectation of this toy function, as a further test of a given method performance.

The proposal distribution is chosen to be a flow model containing 30 RealNVP (Dinh et al., 2017) layers each followed by an ActNorm layer (Kingma and Dhariwal, 2018). We begin by looking at the performance of FAB on this problem, and then compare it to the case where we train only on samples from $q(\mathbf{x})$.

In Figure 4.2, we provide a visualisation of the progression of the FAB model over a training period of 1000 iterations (see Algorithm 2 for what an "iteration" entails). We see that the FAB method works well; at each iteration the samples from AIS look more similar to the target distribution than the samples from the flow, and these samples effectively "guide" the flow towards towards undiscovered modes. Figure 4.2 also demonstrates the "bootstrapping" where AIS improves the flow (by guiding it), and improving the flow improves AIS (by providing it with a better initialisation). We note that the samples after the AIS step do not

need to be from the true distribution in order for the bootstrap to work. Rather, the samples from the AIS step just need to be "good enough" to improve the current flow proposal distribution by a bit. In this MoG problem, 2 intermediate AIS steps results in a relatively minor improvement in the samples - however this is more than sufficient to help guide the flow at each step. The most challenging part of training is at the beginning, when the target and flow proposal are the least similar, and the estimate of the loss is the most noisy. We find that if AIS is good enough to help improve the flow by a little bit at the start of training, then this is enough for the "bootstrap" to kick in, after which the training becomes progressively easier. Thus in future work it may be useful to consider decreasing the number of AIS steps throughout training. Table 4.1 shows that after training samples from both the flow and AIS have a high effective sample size, with the latter significantly higher as expected. Furthermore, we see that the trained flow can be used (both directly and with AIS) to give importance weighted estimates of $\mathbb{E}_{p(\mathbf{x})}\left[f(\mathbf{x})\right]$ with very low bias.

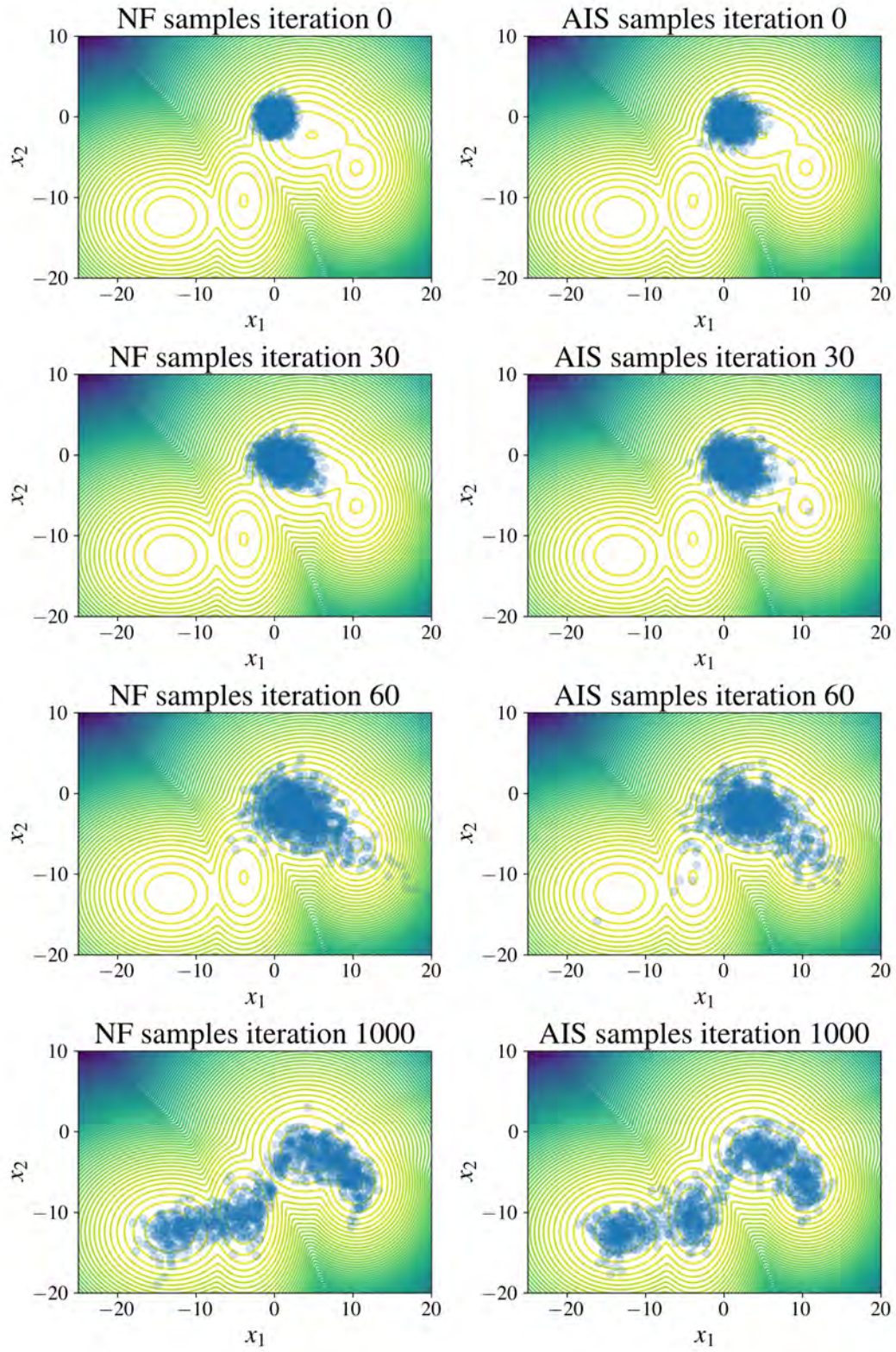


Fig. 4.2 We plot the progression of samples from the flow (NF) proposal and AIS throughout training, with the target MoG log probability contours in the background. At each iteration the AIS samples are closer to the target distribution, helping guide the proposal. The improved flow proposal then helps improve the samples from AIS by improving its initial distribution.

Next we look at the performance of two alternatives, namely (1) training off samples from $q(\mathbf{x})$ and (2) training using FAB but setting $\alpha = 0.1$ to give a loss which will be roughly equivalent to minimising reverse KL-divergence, $\mathrm{KL}(q\|p) = D_{\alpha=0}(p\|q_\phi)$. We see that both of these methods have far worse performance, resulting in importance weighted estimates of $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$ with very large bias (see Table 4.1).

In comparison (1) we use $\mathrm{KL}(q\|p)$ as a loss function - this is equivalent to the "training by energy" loss term used in Boltzmann Generators (see Section 2.7). The resultant proposal distribution after training (Figure 4.3) is concentrated on a subset of the modes nearest to the proposal's initialisation. The reasons for this are that (A) reverse KL-divergence encourages mode seeking and (B) if we train exclusively off samples from $q(\mathbf{x})$ then the training data never includes modes far from the initialisation, and thus the proposal is never updated to include them. We consider this as a benchmark of the FAB method relative to "typical" Boltzmann Generator performance, as the model architecture is the same with only the training method differing. However, we further compare FAB to "typical" Boltzmann Generators by running the code provided by Wu et al. (2020) on the MoG problem[2], using only the "training by energy" objective. We find that both standard Boltzmann Generator and Stochastic Normalising Flow models[3] in Wu et al. (2020) fail on the MoG problem, clinging a subset of the modes, similarly to our implementation (Figure 4.4).

Training using FAB with $\alpha = 0.1$ also results in the proposal fitting a sub-set of the modes, and therefore lower performance (i.e. high bias in expectation estimation). This is due to the mode seeking nature of reverse KL-divergence. Thus we see that both the use of α-divergence (with $\alpha = 2$) as well as AIS in the inner loop are important to achieve good performance with FAB. This stands in contrast to literature on Boltzmann Generators, where minimising reverse KL-divergence is the standard choice (Noé et al., 2019) when we do not have samples from the target. Although both $\mathrm{KL}(q\|p)$ and $D_{\alpha=2}(p\|q)$ are minimised with $p = q$ we can see the loss landscape of $D_{\alpha=2}(p\|q)$ as far more "friendly" than $\mathrm{KL}(q\|p)$. $D_{\alpha=2}(p\|q)$ encourages the proposal to place probability mass on all the modes of the target, after which the proposal can be refined to sequentially look more like the target. On the other hand, $\mathrm{KL}(q\|p)$ first encourages the proposal to not place mass in any of the areas where the target doesn't have mass ("zero-forcing"), often resulting in the proposal fitting a subset of the modes. Once the proposal has fitted a subset of the modes this often results in the proposal getting stuck in a local minima where it is difficult to add probability mass to new modes. This is because (1) samples from the proposal are likely to be far from undiscovered modes,

[2]We use the model configuration that Wu et al. (2020) use for the Double Well Boltzmann distribution problem, simply changing the weighting of the "training by example" loss to 0.

[3]see discussion in Section 5.2 of Stochastic Normalising Flows

making it less likely[4] that the MC estimate of the loss will include samples from these regions and (2) even if the MC estimate of the loss includes samples in the undiscovered regions, because the proposal is typically a smooth function, placing probability mass on a new region often entails placing probability mass between the current region and the new region, which often **increases** $\mathrm{KL}(q\|p)$ which penalises this via "zero-forcing", thus discouraging this behavior.

In Table 4.1, we see that estimates of ESS can very easily be spurious if $q(\mathbf{x})$ places no probability density on some of the target distribution's modes. In this case the plots as well as the bias in the expectation estimate make it easy to detect that the ESS statistic cannot be trusted.

Overall, this problem confirms the pathology of training off samples from $q(\mathbf{x})$, and demonstrates that FAB (with only 2 intermediate distributions) is able to escape this pathological behavior and achieve robust performance.

Table 4.1 Performance of (1) standard FAB, (2) the proposal distribution trained exclusively on its own samples, and (3) FAB trained with $\alpha = 0.1$ to give a loss approximately equivalent to minimising reverse KL-divergence. Estimates of bias and variance are normalised by the true expectation ($\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] = 205.5$). We see that the standard FAB model is far superior in estimation of the toy quadratic function, as the alternatives have extremely high bias. The values for ESS for the second two methods is spurious, as they place very low mass on some modes. Statistics are calculated using 20 runs each with 10000 samples.

	FAB		$q(\mathbf{x})$ KL	FAB ($\approx$KL)	
	IS over $q(\mathbf{x})$	AIS	IS over $q(\mathbf{x})$	IS over $q(\mathbf{x})$	AIS
ESS (%)	64.1 ± 4.1	75.1 ± 2.0	93.0 ± 1.3	50 ± 19.6	65.4 ± 7.2
bias (%)	0.04	0.08	107.1	98.2	103.8
std (%)	14.1	18.1	13.5	18.6	5.4

[4] the AIS step of FAB may be able to discover these modes, however this is on the margin more difficult if the proposal has fitted a subset of the modes.

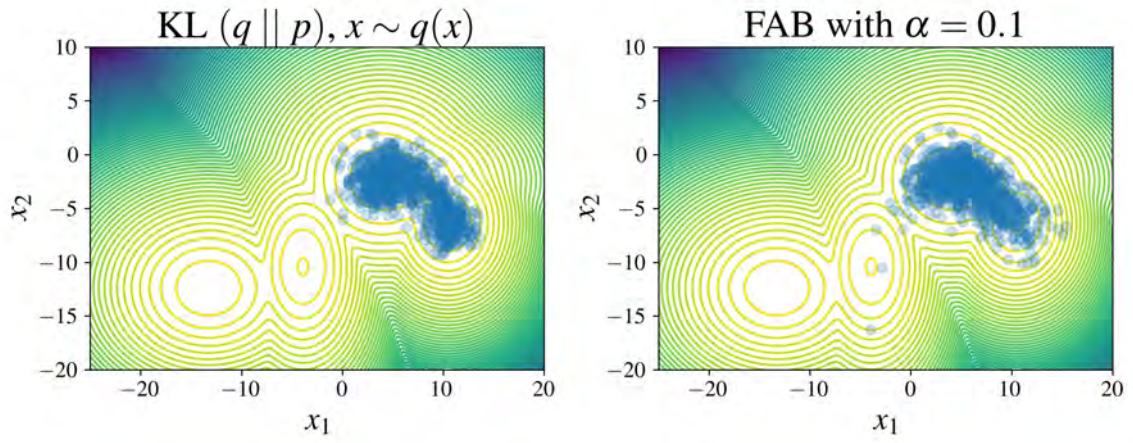


Fig. 4.3 Samples from trained models (1000 iterations) versus the target distribution probability density contours. LHS plot: we train the proposal exclusively on samples from $q(\mathbf{x})$ minimising reverse KL-divergence, which results in fitting a sub-set of the modes. This is the conventional approach for Boltzmann Generators when we do not have access to samples from the target. RHS plot: We train using FAB setting $\alpha = 0.1$ as this gives us a similar α-divergence to reverse KL-divergence ($\alpha = 0$). We see that this objective results in worse performance than with $\alpha = 2$, with more mode clinging.

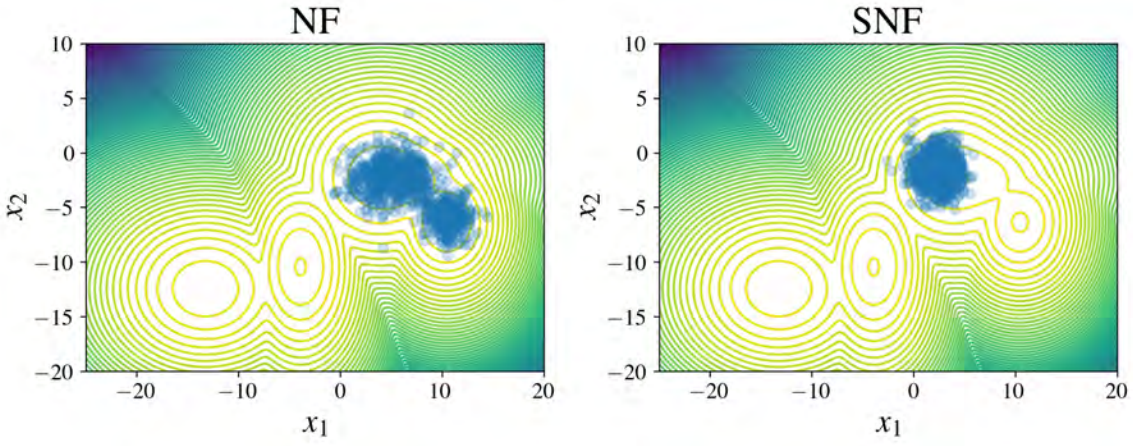


Fig. 4.4 We train Boltzmann Generators on the MoG problem using the code provided by Wu et al. (2020), we see that their methods cling to a subset of the modes. LHS: Vanilla Boltzmann Generator. RHS: Stochastic Normalising Flow model. We noted the SNF model was slower to train so trained for 10000 iterations instead of 1000 iterations. We see that the SNF model is the most reliant on the "training by example" loss, and performs badly without it.

4.4 The Many Well Problem

To test our method on a more challenging problem we utilise the Double Well Boltzmann distribution (Figure 4.5), which has been used in Boltzmann Generator literature (Noé et al., 2019; Wu et al., 2020). We augment the Double Well problem by repeating the two dimensions of the Double Well problem 8 times. To evaluate the log probability density across all dimensions, we simply sum the log probability density functions that correspond to each of the 8 repeated pairs of dimensions. The resultant distribution has $2^{\frac{D}{2}} = 2^{\frac{16}{2}} = 256$ modes, where D is the number of dimensions. This approach of repeating dimensions is standard in literature - for example see Neal (2011), who use this approach to test how HMC's performance scales to higher dimensions. The Boltzmann distribution for large molecules (e.g. proteins) is often high dimensional with many isolated modes. Thus, it is important for approaches that learn approximations to the Boltzmann distribution to be able to scale to more challenging problems like the Many Well problem.

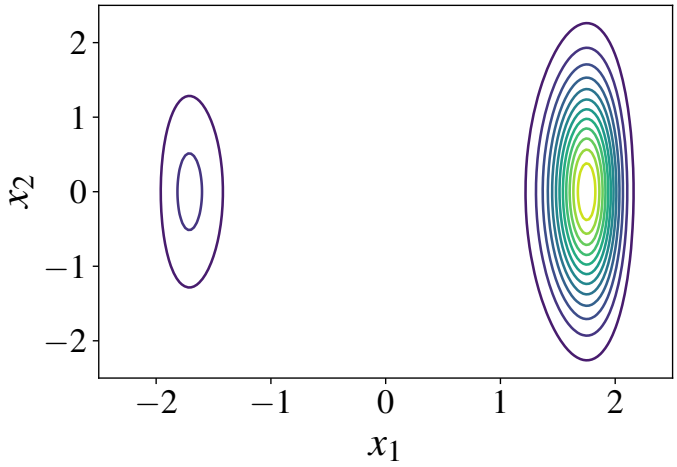


Fig. 4.5 The Double Well problem probability contours. We see that there are two modes, with the one on the right taking up more mass.

We note that Wu et al. (2020) choose a metric to test their SNF model on a Double Well problem which is outperformed using an untrained (but well placed) uniform proposal distribution (we show this in Appendix A.3). This is entirely because their choice of metric is poor - their SNF model is a far greater approximation for the Double Well problem than a uniform distribution. The fact that this high quality paper makes this mistake highlights the importance of taking great care in assessment on these problems! We take careful precautions to ensure we can trust the ESS statistic, which is the main metric we use to judge our model. To do this we create a test set of data, where we manually place points near each mode in the

Many Well problem, and then ensure that our proposal distribution increases the probability of this unseen dataset during training. Furthermore, we inspect plots of each pair of marginal distributions to ensure that they look reasonable.

We utilise a flow model with 20 inverse autoregressive flow[5] (IAF) layers (Kingma et al., 2016), interspersed with ActNorm layers, and use 2 intermediate distribution for AIS. We train the FAB model for 100,000 iterations, with a batch size of 1000, to achieve an effective sample size[6] > 70% (after AIS), and > 60% (over the proposal). The steady increase in effective samples size throughout training is shown in Figure 4.6. The effective sample size improvement follows an S-curve which can be explained as follows: (1) Initially the effective sample size is very low, and AIS is poorly tuned, thus estimates of the loss function are quite noisy and improvement is slow. (2) As the proposal improves and the AIS transition operator becomes well tuned - the "bootstrap" of FAB starts working better, with the estimate of the loss function becoming increasingly more accurate resulting in progressively faster learning. (3) Eventually we start to be limited by the expressiveness of the flow, and the small number of AIS steps used, resulting in a plateau in performance[7]. In Figure 4.6 we see that early in training there is a big drop in the average probability the flow places on the test-set. This corresponds to the proposal momentarily missing a sub-set of the modes, resulting in an extremely low probability for one of the test-set points. However, the average test-set log probability under the proposal recovers, and is significantly higher after training than during initialisation. This is because the AIS step would have discovered this mode during training (even when the proposal no longer placed probability density here), encouraging the proposal to re-allocate its probability density to this area. This type of behavior would be impossible if we only learnt off samples generated by the proposal and demonstrates that the FAB model is robust against mode-clinging. Furthermore in Figure 4.7, we see in plots of the marginal distribution of points generated by AIS, that the samples fit the marginal contours extremely well, with all marginal modes containing samples. Together this confirms that we can trust the effective sample size statistic.

One way to benchmark the efficiency of the FAB model against current, more standard methods is to look at the number of target evaluations used by the algorithm. We note that this does not give us as complete a picture as running both algorithms on the same machine,

[5]IAF is fast in one direction, and slow in another (inversion is slow). In FAB we evaluate the log-prob function much more than generate samples (in each step of HMC we evaluate the log-prob function, but we only generate samples during the start of AIS). IAF is much more expressive than RealNVP, so is therefore a good choice.

[6]Estimated with 10^5 samples

[7]These are not hard limits, and are rather a function of this specific model - as we can add more layers to the proposal distribution to increase its expressiveness, and increase the number of intermediate distribution in AIS.

but still gives us a good idea of efficiency. Noé et al. (2019) states that the Double Well problem requires 4×10^8 steps of "Molecular Dynamics" (it is actually MCMC **not** MD[8]) to obtain a sufficient number of "return trips" between the two states. To estimate how this would scale to higher dimensions we obtain an extremely conservative lower bound (i.e. actual scaling would be much worse) by assuming that the "return trip" takes the perfect path, only visiting each mode once on the way out and back. As we now have 256 modes, we have a path of 255 transitions between modes (instead of 1 for the double well) so this implies that the number of steps required is lower bounded by $4 \times 10^8 \times 255 \approx 10^{11}$ steps.

The training of the FAB model costs approximately 10^9 target evaluations (see Appendix C.2). Thus our method is at least 100 times more efficient (in terms of number of target evaluations) than a conservative lower bound estimate of the performance of the MCMC algorithm described by Noé et al. (2019). This problem is much more difficult than the previous due to the higher dimension and number of modes, and thus training standard Boltzmann Generators using "training by energy" fails catastrophically, where only a small fraction of the modes get covered. We illustrate this in Appendix B.2 where we show that a standard Boltzmann Generator trained on the 8 dimensional version of the Many Well problem converges to covering a subset of the modes.

[8]Noé et al. (2019) note in their supplementary material that they in fact use Metropolis-Hastings rather than MD simulation (in the main paper they simply state that they benchmark against MD simulations, which is misleading). Well tuned Metropolis-Hastings should obtain reasonable performance in 2D so this is still a useful estimate, however it is important to note that this number could potentially be reduced with a more efficient MCMC method, and thus this statistic is quite rough.

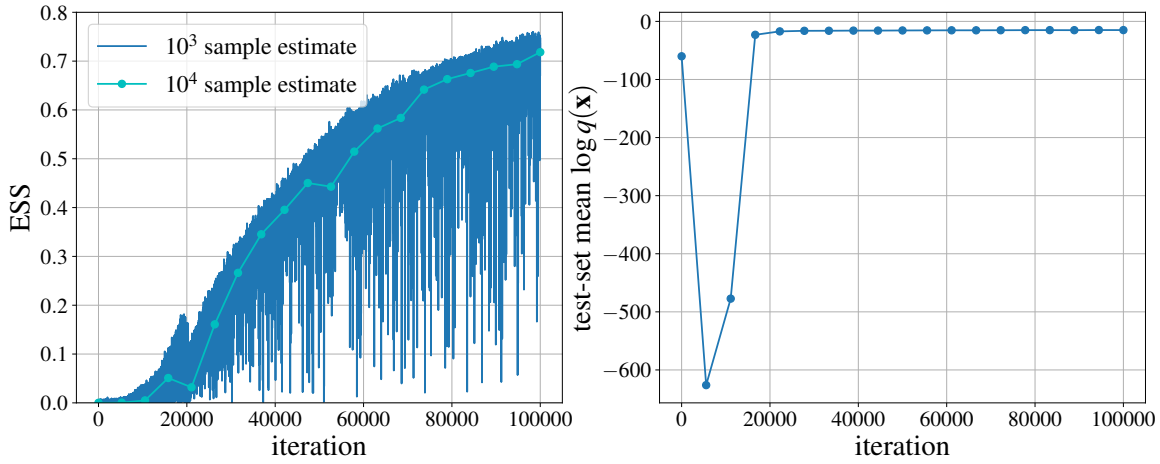


Fig. 4.6 Performance of FAB on Many Well problem during training. We see strong improvement in effective sample size (ESS) of the points generated by AIS, as well as in the mean probability under the proposal distribution of handcrafted test set which contains points on all 256 modes. There is a large dip in the mean probability of the test-set under the proposal, which corresponds to it momentarily missing a mode. However, this is corrected, as samples from AIS still reach this mode, and then encourage the proposal to place probability mass here.

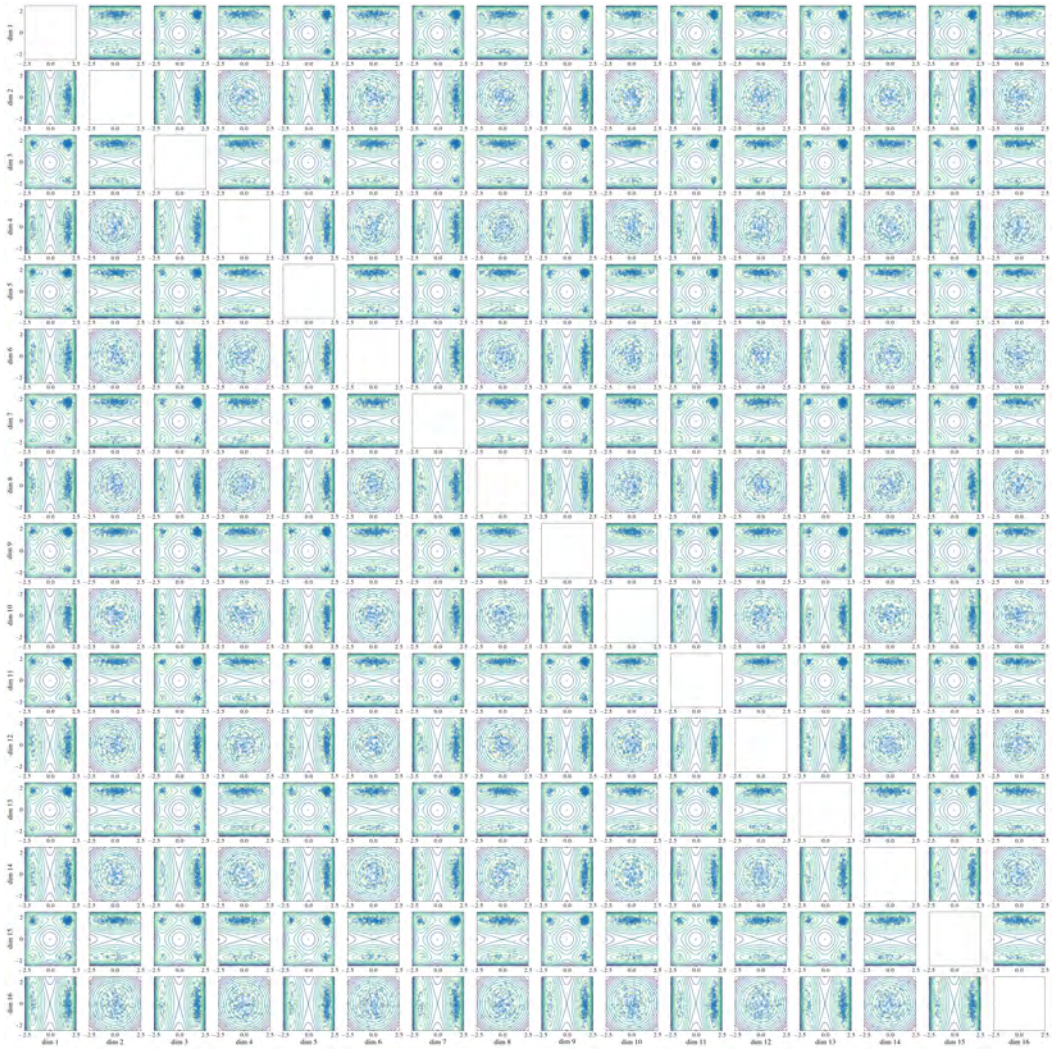


Fig. 4.7 Marginal distribution for each pair of dimensions for samples from AIS for FAB on the Many Well problem, plotted over the target log probability density contours. Zoom to see each plot. We see that each marginals' mode contains samples. There are 3 types of plots; (1) pairs of bi-modal dimensions giving a marginal distribution with 4 modes. (2) one mono-modal and a bi-modal dimension giving a marginal distribution with 2 modes. (3) two uni-modal dimensions giving a uni modal marginal distribution. Note that in dimensions containing a bi-modal distribution, one mode contains more probability mass than the other.

4.5 Annealed Importance Sampling after Training

Besides its utility during training, AIS can be used in conjunction with the trained flow model to improve the effective sampling size. To demonstrate this, using the 32-dimensional Many Well Problem, we partially train a FAB model with 2 intermediate AIS distributions, up until it has an effective sample size of roughly 7% (over the flow) and 14% (after AIS).

The 32 dimensional Many Well problem has $2^{16} = 65536$ modes, so is substantially more difficult than the 16 dimensional version. We confirm that the ESS statistic is trustworthy using the same tests as the previous section, where we make sure that the handcrafted test-set's probability under the proposal is increased throughout training, and that the marginal distribution for each combination of pairs of dimensions does not miss any modes (see Appendix B.2).

We then perform AIS using the flow model as the proposal distribution, varying the number of intermediate distributions. Furthermore, we compare using the tuned HMC step size from the trained FAB model[9] ($\varepsilon = 0.14$) to using the original step size ($\varepsilon = 1.0$), as well as a intermediate step size ($\varepsilon = 0.28$). This results of this are illustrated in Figure 4.8 where we we see a strong improvement (with the tuned step size) in the effective sample size with an increasing number of intermediate AIS distributions, with 64 distributions leading to an ESS $> 50\%$. We also see a strong effect of the "quality" of the transition operator on AIS. The tuned step size obtains the strongest improvement with more AIS distributions, while when we use the initial step size setting ($\varepsilon = 1.0$), this results in failed AIS, due to the vast majority of HMC steps being rejected due to the overly large steps.

Thus, we see that AIS can be used to boost the effective sample size after training. There are important considerations here with regards to the optimal number of intermediate distributions, and their spacing - where increasing the effective sample size has to be balanced with the increasing computational cost. We leave this to future work.

[9]This is the step size of the final AIS transition after training.

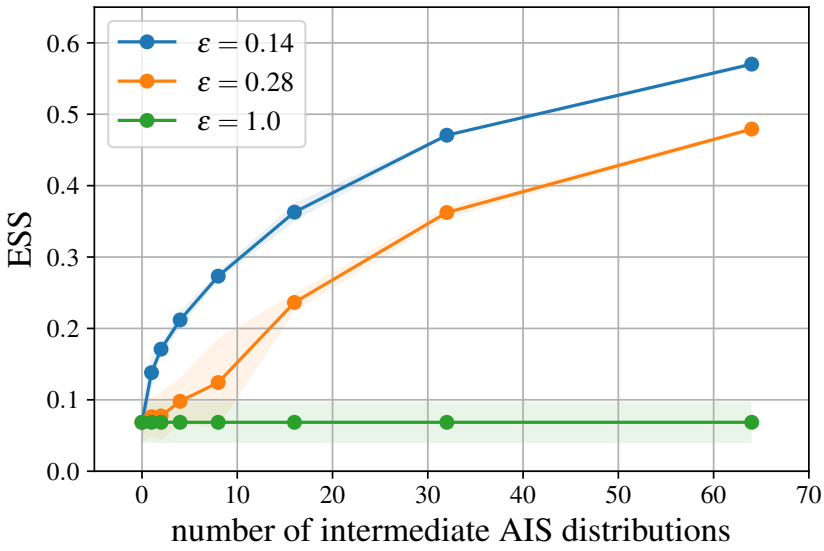


Fig. 4.8 We perform AIS on the (partially) trained flow model to improve the effective sample size. Plotting mean performance across 5 different random seeds, with 1.96σ confidence interval. We see a strong improvement in the effective sample size with an increasing number of intermediate distributions. Furthermore we see that the quality of the transition operator has a significant effect, with the tuned step size from training ($\varepsilon = 0.14$) resulting in greatest improvement in ESS with increasing numbers of intermediate distributions.

Chapter 5

Related Works

In this chapter, we describe relevant pieces of literature to our work. First, we describe how the training of variational autoencoders (VAEs) also includes the problem of learning a good approximation for a distribution from which we cannot sample. We highlight some approaches from literature that incorporate MCMC to obtain good approximations for estimating expectations over this distribution. We then discuss how FAB could be applied to VAEs and compare it to the aforementioned approaches, noting some advantages that it may have.

Following this we discuss Stochastic Normalising Flows (SNFs) by Wu et al. (2020), which we briefly saw when we benchmarked FAB against them on the Mixture of Gaussians problem, where they had far worse performance. SNFs are the most relevant part of a set of recent work that combines normalising flow transitions, with MCMC transitions. We provide a description of SNFs and compare them to FAB.

5.1 MCMC with Variational Autoencoders

Deep Latent Variable Models (DLVM) model some variable $\mathbf{x}$ by assuming a directed graphical model structure with $\mathbf{x}$ conditionally dependant on a relatively simpler set of latent variables $\mathbf{z}$. In this case, the probability observing $\mathbf{x}$ is given by

$$\begin{aligned} p_\theta(\mathbf{x}) &= \int p_\theta(\mathbf{x}, \mathbf{z})\, d\mathbf{z} \\ &= \int p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z})\, d\mathbf{z}, \end{aligned} \tag{5.1}$$

where θ denotes the parameters of the generative model. To train this generative model we would like to estimate the gradient of the data log-likelihood

$$\mathscr{L}(\theta) \;=\; \log p_\theta(\mathbf{x}) = \log \int p_\theta(\mathbf{x},\mathbf{z})\, d\mathbf{z}. \tag{5.2}$$

This is commonly expressed as an expectation over the posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$

$$\nabla_\theta \mathscr{L}(\theta) \;=\; \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\left[\nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z})\right]. \tag{5.3}$$

However because the generative model can only compute the posterior on $p_\theta(\mathbf{x} \mid \mathbf{z})$ and not $p_\theta(\mathbf{z}|\mathbf{x})$, we have no way of obtaining samples from $p_\theta(\mathbf{z} \mid \mathbf{x})$ and thus cannot derive an MC estimate of the above gradient. We see that this is the exact problem that we have focused on in this work; estimating expectations over a target distribution from which we cannot sample. VAEs are the most common method of dealing with this, where we learn an approximation for $p_\theta(\mathbf{z} \mid \mathbf{x})$ (i.e. a encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$) that we can sample from, that is trained in conjunction with $p_\theta(\mathbf{x} \mid \mathbf{z})$ on a variational lower bound of the data likelihood (ELBO). We refer to Kingma and Welling (2019) for a great overview of VAEs.

One way to improve upon this is to perform MCMC with a Markov chain initialised with samples from the decoder to obtain samples closer to $p_\theta(\mathbf{x} \mid \mathbf{z})$. Salimans et al. (2015) and Hoffman (2017) did exactly this using HMC, showing improved training of VAEs. More recently, Ruiz et al. (2020) derived a low variance, unbiased estimate of the log-likelihood gradient using coupled MCMC, and show that this is able to improve VAE training. Unlike standard MCMC, coupled MCMC is able to produce unbiased estimates of expectations over a target function in finite (but random) time (Jacob et al., 2020). Both the work by Hoffman (2017) and Ruiz et al. (2020) focuses exclusively on training the decoder, and train the encoder using the standard VAE ELBO objective. While, Salimans et al. (2015) backpropogate through MCMC to train the encoder.

5.1.1 FAB Variational Autoencoders

We note that FAB can be applied to VAEs which would allow improvement of **both** the encoder and decoder. We define how FAB can be used to train VAEs below;

Similarly to Ding and Freedman (2019)[1], we can train the decoder $p_\theta(\mathbf{z} \mid \mathbf{x})$ by sampling from the proposal encoder distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$, and then using this as a seed for AIS with the unnormalised target $\tilde{p}_\theta(\mathbf{z} \mid \mathbf{x}) = p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z})$. Using the AIS importance weights $\bar{w}_{AIS}^{(1:L)}$,

[1] who use AIS to train the decoder only

we can estimate the gradient of the data likelihood

$$\begin{aligned}\nabla_\theta \mathscr{L}(\theta) &= \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\left[\nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z})\right] \\ &\approx \frac{1}{L}\sum_{l=1}^{L} \bar{w}_{AIS}^{(l)}\, \nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z}^{(l)}).\end{aligned} \tag{5.4}$$

The encoder can be trained using the standard FAB objective given in Equation 3.11, in order to minimise $D_{\alpha=2}(p_\theta(\mathbf{x} \mid \mathbf{z}) \mid\mid q_\phi(\mathbf{z} \mid \mathbf{x}))$, using the samples and weights returned by AIS. We provide the FAB algorithm for application to VAEs below:

Algorithm 2: FAB for training VAE encoder and decoder

Set target p
Initialise proposal q_ϕ
for *iteration = 1, M* **do**
  Sample batch $\mathbf{x}_q^{(1:L)}$, $\log q(\mathbf{x}_q^{(1:L)})$ from q_ϕ
  Generate batch $\mathbf{x}_{AIS}^{(1:L)}$, $\log w_{AIS}^{(1:L)}$ from AIS seeded with $\mathbf{x}_q^{(1:L)}$, $\log q(\mathbf{x}_q^{(1:L)})$
  Calculate decoder FAB objective:
  $\mathscr{L}(\theta) = \frac{1}{L}\sum_{l=1}^{L} \bar{w}_{AIS}^{(l)} \log p_\theta(\mathbf{x},\mathbf{z}^{(l)})$
  Calculate encoder FAB objective:
  $\mathscr{L}(\phi) = \log \sum_{l=1}^{L} \exp\left(\log \bar{w}_{\text{AIS}}^{(l)} + \left(\log p(\mathbf{x}_{\text{AIS}}^{(l)}) - \log q_\phi(\mathbf{x}_{\text{AIS}}^{(l)})\right)\right)$
  Perform gradient descent on $\mathscr{L}(\theta)$ and $\mathscr{L}(\phi)$
end

The samples from AIS will be closer to $p_\theta(\mathbf{z} \mid \mathbf{x})$ than those directly from the encoder, and therefore provide a better set of points for training the decoder. Furthermore, the training objective in 5.4 is an unbiased estimator of the data log likelihood, while ELBO is biased (as it is a lower bound). As we saw in previously in this chapter, training a proposal exclusively off samples generated by itself can lead to it missing important modes. Thus, by using FAB to train the encoder, we could potentially improve its similarity to $p_\theta(\mathbf{z} \mid \mathbf{x})$. When the encoder outputs simple distributions (e.g. factorised Gaussians), the standard ELBO objective is probably sufficient. However for more expressive encoder distributions, the FAB loss could help encourage the encoder to better capture $p_\theta(\mathbf{x} \mid \mathbf{z})$. Interestingly Salimans et al. (2015) noted that AIS had strong potential for MCMC with VAEs, but stated that a disadvantage was that having a transition operator that needs to meet the detailed balance makes this hard to train the encoder, as it does not allow backpropogation through MCMC. Ding and Freedman (2019) use AIS exclusively to train the decoder. FAB bridges this gap and provides a way of making use of AIS to also train the encoder!

Even in the case that one can backpropogate through MCMC to train the decoder, as in Salimans et al. (2015), FAB is preferred. FAB allows a gradient estimator that improves as the quality of the samples produced by AIS improve. On the other hand, backpropogating through MCMC provides a increasingly poor signal to the proposal distribution as the quality of MCMC improves. This can be seen by imagining the extreme case where MCMC is perfect and returns samples from the true posterior independent to the proposal distribution for the start of the chain. In this case there is no signal to the proposal if we backpropogate through MCMC, as the proposal now has no effect. While for FAB the training objective simplifies to the case where we are estimating the gradient of the objective (for minimising $D_{\alpha=2}(p\|q_\phi)$) over the target distribution, which is the best case scenario (becomes supervised learning).

It is also worth noting the coupled MCMC initialised with the proposal, as used in Ruiz et al. (2020) presents a potential alternative variation[2] on FAB where instead of using AIS to obtain an estimate of a gradient operator that is an expectation over $p(\mathbf{x})$, we could instead use coupled MCMC initialised with samples from the proposal. We therefore note that the work by Ruiz et al. (2020) could be simply extended by not only using the samples produced by MCMC to improve the decoder training, but also to improve the encoder by using them to compute an MC estimate $\propto D_{\alpha=2}(p_\theta(\mathbf{x} \mid \mathbf{z}) \mid\mid q_\phi(\mathbf{z} \mid \mathbf{x}))$ for the encoder training objective similarly to FAB. As AIS is well suited to importance sampling, with both theoretical and empirical support of good performance for making low variance estimates, it is a good candidate for our method. However it is important to note that we can easily introduce other methods which are able to transform samples from a proposal distribution to obtain a lower variance estimates of expectations over $p(\mathbf{x})$. This family of algorithms (replacing AIS with alternatives) would obtain the same benefits of FAB - i.e. bootstrapping with the proposal and the additional method, with each working together.

5.2 Stochastic Normalising Flows

SNFs combine normalising flows with MCMC in a way that retains the ability to perform importance sampling, allowing them to be used as Boltzmann Generators. The rationale behind SNFs is that they help improve the expressiveness of the flow model, where the flow model can be limited by the invertibility constraint.

SNFs work by defining a "forward-path" from a simple prior distribution to a complex target distribution, where each step is made up of interwoven deterministic normalising flow

[2]This has a downside in that coupled MCMC takes random time, while AIS takes deterministic time.

transforms, and stochastic MCMC transforms. The stochastic steps mean that the probability density of SNFs is intractable. However, by considering a backwards path from the target distribution to the prior, this allows Wu et al. (2020) to derive importance weights in a similar manner to the AIS (see Appendix A.1). Using these importance weights, Wu et al. (2020) train SNF based Boltzmann Generators, with the "training by energy" and "training by example" loss functions. Wu et al. (2020) show that SNFs have more fine grained densities than flow models. For example, SNFs do not exhibit the narrow band of probability density that flow models often place connecting different modes. This fine-grained expressiveness is a different goal to our work, where FAB aims to (1) improve the **training** in situations where we don't have access to samples from $p(\mathbf{x})$, and (2) minimise the variance in importance sampling estimates, which is more course grained. One can roughly think of the work on SNFs as focusing on improving from an effective sample size of 90% to 100%, where the difference comes from limits in the expressiveness of the normalising flow. While FAB's focus is on getting from an effective sample size of 0.001% to 50%, where the limits are in the difficulty of training without access to samples from $p(\mathbf{x})$.

Because SNF's are able to obtain importance weights of the samples generated in the "forward" direction, the FAB bootstrapping objective could be easily incorporated into its training. In Wu et al. (2020), the Boltzmann distribution problems in which the SNF is applied to include samples from the target distribution, and thus they do not have to worry about the pathologies of training exclusively off samples from $q(\mathbf{x})$. We saw previously on the Mixture of Gaussian's problem, that SNFs achieved low performance when trained exclusively on its own samples. This is because current SNFs are heavily reliant on the "training by example" loss term, requiring samples from the target distribution. Thus, introducing ideas from FAB into the training of SNF's could be useful for extending SNF's application to more challenging problems where we don't have access to samples from $p(\mathbf{x})$.

Chapter 6

Improving FAB

The FAB algorithm used in the previous chapters is still mainly a proof of concept, with many possible improvements that could be made. In this chapter we discuss some major improvements to the FAB algorithm. We think the last of these, which focuses on improving exploration is the most interesting, as we note that the proposal distribution allows us to express exploration in a clear manner.

6.1 Hyperparameter Optimisation

In this work we have performed very little hyperparameter optimisation. However, there are many hyperparameter choices (e.g. number of intermediate distributions, number of MCMC steps per transition between intermediate distributions) which are worth theoretical consideration to find good trade-offs for efficient learning. Performing a thorough analysis of these would be important for future work. This would most likely improve performance, and more importantly it would allow us to gain a more clear understanding of the FAB algorithm. There are also more banal hyperparameters (e.g. learning rate, number of flow layers) for which it would be useful to perform standard hyperparameter optimisation (e.g. Bayesian optimisation), as this could result in a significant performance increases. We can consider the lack of hyperparameter optimisation performed in this report to be partially good, as it shows that the method is quite robust.

6.2 Gradient based HMC tuning

In the toy problems in the previous section, the length scale for various dimensions was roughly the same, meaning that a shared HMC step size across dimensions was reasonable, and the simple tuning method (p-accept=0.65) was robust. However in problems with different length scales, more sophisticated tuning methods for HMC would be necessary. As the tuning method has to run inside an inner training loop, it is important for the additional overhead required for tuning to be small. Hoffman and Gelman (2011) and Levy et al. (2017) provides a suitable choice of loss function which involves maximising the expected squared distance on the steps. The methods in Hoffman and Gelman (2011) and Levy et al. (2017) requires relatively high computational overhead during tuning. We propose that the following method for tuning would be very computationally cheap: using the squared distance loss but only differentiating with respect to the final inner step of HMC. This would work because the final step of HMC tells us if the squared distance travelled is increasing or decreasing and the step size of each dimension is shared[1] across the inner loop. In future work, testing this method amongst others would be interesting and important for problems with varying length scales.

6.3 Replay Memory

In the previous chapter we only use samples generated from AIS once, after which they were discarded. A simple way to improve the efficiency of FAB is to instead store samples in a replay memory buffer similar to that used by Mnih et al. (2013). As the samples and importance weights returned by the AIS step are considered independent to the current FAB model in the estimation of the gradient operator, they can be simply re-used. As we expect the quality of the samples to improve steadily throughout training it makes sense to periodically discard old sam-

[1] so if the step size of the final inner step is too big (we have performed a U-turn/rejection probability is high) then this implies that the shared step size is too big.

ples[2]. Thus we can rewrite the FAB algorithm with the inclusion of replay memory as follows:

Algorithm 3: FAB for minimisation of $D_{\alpha=2}(p \parallel q_\phi)$ with Replay Memory

Set target p
Initialise proposal q_ϕ
Initialise replay memory D to max-length N
for *iteration = 1, M* **do**
  Sample batch $\mathbf{x}_q^{(1:K)}$, $\log q(\mathbf{x}_q^{(1:K)})$ from q_ϕ
  Generate batch $\mathbf{x}_{AIS}^{(1:K)}$, $\log w_{AIS}^{(1:K)}$ from AIS seeded with $\mathbf{x}_q^{(1:K)}, \log q(\mathbf{x}_q^{(1:K)})$
  Store $(\mathbf{x}_{AIS}^{(1:K)}, \log w_{AIS}^{(1:K)})$ in D
  Sample random minibatch $(\mathbf{x}_j^{(1:L)}, \log w_j^{(1:L)})$ from D
  Calculate FAB objective:
  $\mathscr{L}(\phi) = \log \sum_{l=1}^{L} \exp\left(\log \bar{w}_j^{(l)} + \left(\log p(\mathbf{x}_j^{(l)}) - \log q_\phi(\mathbf{x}_j^{(l)})\right)\right)$
  Perform gradient descent on $\mathscr{L}(\phi)$
end

where K is the batch size for the generation of new points, while $L \geq K$ is the batch size used for sampling from the replay memory.

6.4 Mixture Models

Instead of training a single proposal distribution, we could instead train many proposal distributions in parallel, and then use them together in a mixture model defined by

$$q(\mathbf{x}) = \sum_i^N \pi_i q_i(\mathbf{x}). \tag{6.1}$$

We can set $\pi_i = \frac{1}{N}$ where N is the number of proposal distributions. To sample from this mixture we simply sample i from the uniform categorical variable over each mixture component, and then sample from the relevant proposal q_i. As each $q_i(\mathbf{x})$ can be computed independently, this mixture model can make use of parallel computation. This mixture model should obtain an improved accuracy (each model likely to make independent errors which can cancel). We note that this greatly reduces the chance of a mode being missed, as if the probability of a certain mode being missed by a single proposal distribution is p_{miss}, then the probability that the mixture model misses this is p_{miss}^N. As FAB is quite robust,

[2]It would also worth considering storing a separate catalogue of representative points across the space in different modes

we expect p_{miss} will be low, and thus p_{miss}^N quickly becomes negligible. As some training hyperparameters (e.g. learning rate) increase the speed of training, while also increasing the probability of missing a mode, the mixture of proposals may allow for a faster training scheme, as it is less important that the proposal does not miss a single mode.

The simplest way to train the mixture of proposals would be to just run train each model independently in parallel. Alternatively, it may be worth considering sharing a portion of samples in the Replay Buffer across different models during training. Backpropogating through mixture models can be tricky, but in both of the above schemes the generation of samples and log-weights by the mixture model is not differentiated with respect to the proposal's parameters.

6.5 Exploration

In the FAB model proposed in Chapter 3, the MCMC within AIS is responsible for exploration: the discovery of new points outside the current areas with probability density of the proposal distribution. This is especially clear in the visualisation of samples from the proposal vs AIS in 2D space for the mixture of Gaussians problem (see Section 4.3). However we think it would be better to treat the exploration component part of the problem more explicitly, by including an additional algorithm that focuses purely on exploration. Our proposal distribution describes the current "best guess" of the target distribution, and it is therefore useful to express exploration in terms of both the target and proposal distribution. We think the following two "exploration" objective functions would be interesting to test;

$$O(\mathbf{x}) \;=\; \log p(\mathbf{x}) \;-\; \log q_\phi(\mathbf{x}) \tag{6.2}$$

$$O(\mathbf{x}) \;=\; 2\log p(\mathbf{x}) \;-\; \log q_\phi(\mathbf{x}). \tag{6.3}$$

The first exploration objective function effectively subtracts the current modes in the log probability landscape of q from p. Thus if our proposal distribution has captured a certain mode perfectly, then this would effectively squash the mode flat in the exploration landscape. The second exploration objective function is based off inspection of $D_{\alpha=2}(p\|q) \propto \int \frac{p^2(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x}$, and encourages finding the regions that make the biggest contribution to $D_{\alpha=2}(p\|q)$.

We can discover new useful points in space by optimising these objectives using standard gradient ascent, using many different seeds in parallel. These points can then be incorporated

into the training of FAB by including an additional term in the objective

$$O(\phi) \;=\; \log q_\phi(\mathbf{x}_{explore}), \tag{6.4}$$

which encourages our proposal distribution to place probability mass on the $\mathbf{x}_{explore}$ points. The above equation is agnostic about the method for exploration, so it would be easy to test out a variety of methods and see what works best. As exploration is more important early in training, we could allocate more computational resources on this at the beginning of training. However as the exploration objective incorporates the current proposal distribution, it would most likely be useful to include this throughout training (as later on in training the exploration objective could be used to simply find points that more neglected on the margin.).

In literature HMC is often tasked with the responsibility of exploration. For example the standard method for obtaining samples from the posterior of a BNN is to run many long chains of HMC in parallel (Izmailov et al., 2021). However as HMC has strong additional constraints, it is not the perfect explorer. The above method allows us to express exploration clearly in terms of our current map of the landscape, and then easily incorporate newly discovered points into our probabilistic model. Thus it would be interesting to add the exploration term to FAB and benchmark it against current state of the art methods for learning complex probability distributions (e.g. BNNs).

Chapter 7

Conclusions and Future Work

In this work we have proposed FAB, a method which combines normalising flow models with AIS in a method that allows us to learn good approximations for complex target distributions that we cannot sample from. In the FAB method, we use samples from the flow model as the starting point for AIS, which then generates samples closer to the target distribution that have lower variance importance weights and a higher effective sample size. The key "trick" behind FAB is that we can write our flow model's training objective (minimising α-divergence between the flow and the target, with $\alpha = 2$), as an expectation over the target distribution, and then use AIS to estimate the gradient of this objective with respect to the flow model parameters. This allows us to obtain a robust training procedure for the flow model without any reliance on samples from the target distribution. The FAB method obtains the benefit of bootstrapping, where AIS is used to improve the flow model (by improved estimation of the flow model gradient operator) which then improves AIS (by improving its initial distribution). We also show that AIS can be used on a trained flow model to improve its effective sample size.

In our experiments, we show that FAB is superior to training a flow model exclusively off its own samples. This approach of training a flow model exclusively off its own samples is the typical approach used in literature for cases where we cannot sample from the target, and is used both for Boltzmann Generators ("training by energy") (Noé et al., 2019) and neural importance samplers (Müller et al., 2019). We also show that FAB can successfully be applied to high dimensional (16 and 32 dimensional) Boltzmann distributions with many modes (256 modes for 16 dimensions, 65536 modes for 32 dimensions).

Our work has application to current literature on Boltzmann Generators. Firstly, we can take a given trained Boltzmann Generator and instead of sampling directly from it when

computing expectations, instead use the Boltzmann Generator as the proposal distribution for AIS, which will significantly increase the effective sample size. Secondly, for SNF based Boltzmann Generators, we can obtain a better training objective for the flow layers by utilising samples from the stochastic layers to estimate the FAB loss function, which then replaces the flow training objective originally used for SNFs.

More generally, this work unites the areas of neural importance samplers and neural MCMC. Previous literature has showed that neural network based methods can be used to improve the transition kernel of MCMC (Levy et al., 2017; Song et al., 2018). This work has shown how that we can also use a neural network for the proposal distribution, and that AIS is a suitable choice in such a context (where MCMC is used between intermediate distributions). Crucially, we have derived robust methods of training in this context.

The limitations to this work are: (1) the FAB algorithm is still in a relatively rudimentary form, with many possible improvements that could be made (e.g. to improve sample efficiency) that were not tested, (2) this work does not provide a clear analysis of the various trade-offs made in selection of hyperparameters, the most important of which being the trade-off between obtaining a more accurate estimate of the FAB objective function by using more AIS steps and the increased compute required and (3) although we have provided clear comparison to the alternative approach of training flow models using samples from the flow, we have not performed benchmarking against more general methods (e.g. state of the art MCMC). In the final section of this work below, we discuss how to improve upon these limitations and other interesting avenues of further work.

7.1 Future Work

In Chapter 6, we provided a list of improvements to FAB that can be made in future work; hyper-parameter optimisation, gradient based tuning of HMC, re-use of samples with replay memory, and improving exploration. The most interesting these was adding an algorithm which focus purely on exploration of the space. We noted that the proposal distribution allows us to express our current estimate of the distribution landscape, and therefore can be combined with the target distribution in ways that encourage finding new modes that our model is unaware of. With these additions we hope that FAB could provide a way for end-to-end training on extremely difficult problems, without any access to samples from the target.

This work has focused on showing that the FAB method works well, and is far superior to training a proposal distribution using samples from itself. However there is still further

benchmarking which should be done to analyse FAB in a broader context. This benchmarking should include comparisons to (1) high quality MCMC and (2) generating samples from MCMC and then training a flow model on these in a supervised fashion. The former would be useful because this is the current state of the art method for calculating expectations over target distributions (e.g. this is what is used for BNNs). The latter would be useful because it would determine if the "bootstrap" component of the FAB method is necessary. BNNs are a good problem for this benchmarking, as there is a vast amount of literature on this to compare against (Izmailov et al., 2021).

Another interesting avenue for future work on FAB is VAEs. In Section 5.1.1 we discussed how FAB could be used to **both** improve the quality of the samples sent to the decoder and improve training of the encoder. We noted that methods which use MCMC to improve VAE training focus exclusively on helping train the decoder. However, some of these methods (specifically Ruiz et al. (2020)), could utilise a FAB style training objective to improve their encoder without having to alter the overall algorithm.

We see our work as placing importance sampling and MCMC on a spectrum rather than a dichotomy. On one extreme is "one-shot" generation of samples with the proposal. On the other is running MCMC for long enough that the initial proposal is irrelevant. Generally "one-shot" is preferred **if** it can generate sufficiently high quality samples, because it is computationally cheap. In the case that we can obtain a sufficiently flexible proposal distribution such that it is hypothetically able to accurately approximate the target, then FAB presents a useful way of training such a model. In the case that our proposal distribution is insufficiently expressive such that it can only roughly fit the target shape (e.g. maximum proposal effective sample size of 5% even with global optimum of flow model parameter setting), then FAB presents a good way of training a reasonably good proposal, after which AIS can be used to further improve the quality of the samples. Besides placing importance sampling and MCMC on a spectrum, we hope that our work can contribute towards moving **on the margin** more towards computationally cheap importance sampling over MCMC. Overall we think that a combination of the two into a single method that leverages the benefits of both will be best. Therefore, we think that performing a thorough investigation of this spectrum presents a promising avenue for further work. Specifically, it would be important to perform careful analysis of: (1) the computational trade-offs between compute requirements and improving the quality of the samples with more MCMC/AIS steps both during and after training and (2) what the practical limits are to the expressiveness of normalising flow models.

References

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. (2019). Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR.

Berg, R. v. d., Hasenclever, L., Tomczak, J. M., and Welling, M. (2018). Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*.

Beskos, A., Pillai, N. S., Roberts, G. O., Sanz-Serna, J. M., and Stuart, A. M. (2010). Optimal tuning of the hybrid monte-carlo algorithm. *arXiv eprint arXiv:1001.4460*.

Betancourt, M. (2017). A conceptual introduction to hamiltonian monte carlo. *arXiv eprint arXiv:1701.02434*.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. (2011). *Handbook of markov chain monte carlo*. CRC press.

Chen, R. T., Behrmann, J., Duvenaud, D., and Jacobsen, J.-H. (2019). Residual flows for invertible generative modeling. *arXiv preprint arXiv:1906.02735*.

De Cao, N., Aziz, W., and Titov, I. (2020). Block neural autoregressive flow. In *Uncertainty in Artificial Intelligence*, pages 1263–1273. PMLR.

Dibak, M., Klein, L., and Noé, F. (2021). Temperature steerable flows and boltzmann generators. *arXiv preprint arXiv:2108.01590*.

Ding, X. and Freedman, D. J. (2019). Learning deep generative models with annealed importance sampling. *arXiv preprint arXiv:1906.04904*.

Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv eprint arXiv:1410.8516*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using real nvp. *arXiv eprint arXiv:1605.08803*.

Dolatabadi, H. M., Erfani, S., and Leckie, C. (2020). Invertible generative modeling using linear rational splines. In *International Conference on Artificial Intelligence and Statistics*, pages 4236–4246. PMLR.

Duane, S., Kennedy, A., Pendleton, B. J., and Roweth, D. (1987). Hybrid monte carlo. *Physics Letters B*, 195(2):216–222.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019a). Cubic-spline flows. *arXiv eprint arXiv:1906.02145*.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019b). Neural spline flows. *Advances in Neural Information Processing Systems*, 32:7511–7522.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. (2019). Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pages 2722–2730. PMLR.

Hoffman, M. D. (2017). Learning deep latent gaussian models with markov chain monte carlo. In *International conference on machine learning*, pages 1510–1519. PMLR.

Hoffman, M. D. and Gelman, A. (2011). The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *arXiv eprint arXiv:1111.4246*.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. (2018). Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2078–2087. PMLR.

Izmailov, P., Vikram, S., Hoffman, M. D., and Wilson, A. G. (2021). What are bayesian neural network posteriors really like? *arXiv eprint arXiv:2104.14421*.

Jacob, P. E., O'Leary, J., and Atchadé, Y. F. (2020). Unbiased markov chain monte carlo methods with couplings. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(3):543–600.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *arXiv eprint arXiv:1807.03039*.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29:4743–4751.

Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Köhler, J., Klein, L., and Noé, F. (2020). Equivariant flows: Exact likelihood generative learning for symmetric densities. *arXiv eprint arXiv:2006.02425*.

Levy, D., Hoffman, M. D., and Sohl-Dickstein, J. (2017). Generalizing hamiltonian monte carlo with neural networks. *arXiv eprint arXiv:1711.09268*.

Martino, L., Elvira, V., and Louzada, F. (2017). Effective sample size for importance sampling based on discrepancy measures. *Signal Processing*, 131:386–401.

Minka, T. et al. (2005). Divergence measures and message passing. Technical report, Citeseer.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv eprint arXiv:1312.5602*.

Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. (2019). Neural importance sampling. *ACM Transactions on Graphics (TOG)*, 38(5):1–19.

Neal, R. (1995). Bayesian learning for neural networks [phd thesis]. *Toronto, Ontario, Canada: Department of Computer Science, University of Toronto*.

Neal, R. M. (2001). Annealed importance sampling. *Statistics and computing*, 11(2):125–139.

Neal, R. M. (2011). *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC.

Noé, F., Olsson, S., Köhler, J., and Wu, H. (2019). Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457).

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2019). Normalizing flows for probabilistic modeling and inference. *arXiv eprint arXiv:1912.02762*.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR.

Rippel, O. and Adams, R. P. (2013). High-dimensional probability estimation with deep density models. *arXiv eprint arXiv:1302.5125*.

Ruiz, F. J., Titsias, M. K., Cemgil, T., and Doucet, A. (2020). Unbiased gradient estimation for variational auto-encoders using coupled markov chains. *arXiv preprint arXiv:2010.01845*.

Salimans, T., Kingma, D. P., and Welling, M. (2015). Markov chain monte carlo and variational inference: Bridging the gap. *arXiv eprint arXiv:1410.6460*.

Song, J., Zhao, S., and Ermon, S. (2018). A-nice-mc: Adversarial training for mcmc. *arXiv eprint arXiv:1706.07561*.

Tabak, E. G. and Turner, C. V. (2013). A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164.

Tabak, E. G. and Vanden-Eijnden, E. (2010). Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217 – 233.

Wehenkel, A. and Louppe, G. (2019). Unconstrained monotonic neural networks. *Advances in Neural Information Processing Systems*, 32:1545–1555.

Wu, H., Köhler, J., and Noé, F. (2020). Stochastic normalizing flows. *arXiv eprint arXiv:2002.06707*.

Wu, Y., Burda, Y., Salakhutdinov, R., and Grosse, R. (2017). On the quantitative analysis of decoder-based generative models. *arXiv eprint arXiv:1611.04273*.

Zhu, H. and Rohwer, R. (1995). Information geometric measurements of generalisation. Technical report, Aston University.

Appendix A

Further Notes and Proofs

A.1 Proof of Annealed Importance Sampling Weights Formula

Neal (2001) proves a brief discussion of how to derive the annealed importance weights formula. We provide a more fleshed out proof below.

The importance sampling weights for the final point $\mathbf{x}^{(i)} = \mathbf{x}_{N-1}$ returned from the annealed importance sampling algorithm (see Section 2.5) can be calculated using

$$w(\mathbf{x}^{(i)}) = \frac{\tilde{p}_1(\mathbf{x}_0)}{\tilde{p}_0(\mathbf{x}_0)} \frac{\tilde{p}_2(\mathbf{x}_1)}{\tilde{p}_1(\mathbf{x}_1)} \cdots \frac{\tilde{p}_{N-1}(\mathbf{x}_{N-2})}{\tilde{p}_{N-2}(\mathbf{x}_{N-2})} \frac{\tilde{p}_N(\mathbf{x}_{N-1})}{\tilde{p}_{N-1}(\mathbf{x}_{N-1})}. \tag{A.1}$$

To see why this is the case we first note that the joint distribution of the points generated according to the steps described in Listing 1 are given by

$$p(\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{N-2}, \mathbf{x}_{N-1}) = p(\mathbf{x}_0) \prod_{n=1}^{N-1} T_n(\mathbf{x}_{n-1}, \mathbf{x}_n). \tag{A.2}$$

We refer to this as the "forward-joint" sequence below. Next, we construct a "backwards-joint" sequence of samples, which we can imagine as starting in the target distribution and following the reverse of the sequence of transitions defined in the AIS generation process. This is defined by the joint distribution

$$\hat{p}(\mathbf{x}_{N-1}, \mathbf{x}_{N-2}, ..., \mathbf{x}_1, \mathbf{x}_0) = \hat{p}_N(\mathbf{x}_{N-1}) \prod_{i=1}^{N-1} \hat{T}_{N-i}(\mathbf{x}_{N-i}, \mathbf{x}_{N-i-1}), \tag{A.3}$$

where we sample $\mathbf{x}_{N-1}$ from our target distribution, and then transition using $\hat{T}_{N-1:1}$, where each transition $\hat{T}_j$ is the inverse of the AIS transition operator T_j, which satisfies the following relation

$$\begin{aligned} p_j(\mathbf{x}', \mathbf{x}) &= p_j(\mathbf{x}) T_j(\mathbf{x}, \mathbf{x}') \\ &= p_j(\mathbf{x}') \hat{T}_j(\mathbf{x}', \mathbf{x}) \\ \implies \hat{T}_j(\mathbf{x}', \mathbf{x}) &= \frac{p_j(\mathbf{x})}{p_j(\mathbf{x}')} T_j(\mathbf{x}, \mathbf{x}'). \end{aligned} \tag{A.4}$$

We see that we can write the expectation we desire to estimate over our target distribution (i.e. the hypothetical sampling distribution in the above "reverse-joint" sequence) to instead be over this joint distribution using the following identity

$$\begin{aligned} \mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x}] &= \int p(\mathbf{x}) f(\mathbf{x}) \, d\mathbf{x} \\ &= \int p(\mathbf{x}, \mathbf{y}) \, d\mathbf{y} \, f(\mathbf{x}) \, d\mathbf{x} \\ &= \mathbb{E}_{p(\mathbf{x}, \mathbf{y})}[f(\mathbf{x}]. \end{aligned} \tag{A.5}$$

which gives us

$$\mathbb{E}_{\hat{p}_N(\mathbf{x}_{N-1})}[f(\mathbf{x}_{N-1})] = \mathbb{E}_{\hat{p}(\mathbf{x}_{N-1}, \mathbf{x}_{N-2}, \ldots, \mathbf{x}_1, \mathbf{x}_0)}[f(\mathbf{x}_{N-1})]. \tag{A.6}$$

We can now compute an importance sampling estimate of the above expectation over the "reverse-joint" distribution, using our "forward-joint" distribution as the proposal. The importance weights for this are then given by

$$w(x^{(i)}) = \frac{\hat{p}(\mathbf{x}_{N-1}, \mathbf{x}_{N-2}, \ldots, \mathbf{x}_1, \mathbf{x}_0)}{p(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N-2}, \mathbf{x}_{N-1})}. \tag{A.7}$$

where $\mathbf{x}^{(i)} = \mathbf{x}_{N-1}$. Plugging in Equation A.2 and A.3 we get

$$w(x^{(i)}) = \frac{\hat{p}_N(\mathbf{x}_{N-1}) \prod_{i=1}^{N-1} \hat{T}_{N-i}(\mathbf{x}_{N-i}, \mathbf{x}_{N-i-1})}{p(\mathbf{x}_0) \prod_{n=1}^{N-1} T_n(\mathbf{x}_{n-1}, \mathbf{x}_n)}. \tag{A.8}$$

Plugging in Equation A.4, and noting that our the following refers to our target distribution $p(\mathbf{x}) = \hat{p}_N(\mathbf{x})$ we get

$$w(x^{(i)}) = \frac{p_N(\mathbf{x}_{N-1}) \prod_{i=1}^{N-1} T_{N-i}(\mathbf{x}_{N-i-1}, \mathbf{x}_{N-i}) \frac{p_{N-1}(\mathbf{x}_{N-1})}{p_{N-i}(\mathbf{x}_{N-i-1})}}{p(\mathbf{x}_0) \prod_{n=1}^{N-1} T_n(\mathbf{x}_{n-1}, \mathbf{x}_n)}. \tag{A.9}$$

the transition operators on the top and bottom now cancel to give

$$w(x^{(i)}) \quad = \quad \frac{p_N(\mathbf{x}_{N-1}) \prod_{i=1}^{N-1} \frac{p_{N-1}(\mathbf{x}_{N-1})}{p_{N-i}(\mathbf{x}_{N-i-1})}}{p(\mathbf{x}_0)}. \tag{A.10}$$

which if set replace $p_{1:N}(\mathbf{x})$ with their un-normalised probability density function $\tilde{p}_{1:N}(\mathbf{x})$, and expand we obtain the original equation for the importance weights (which we now denote $\tilde{w}$ to differentiate from the equation above)

$$\tilde{w}(\mathbf{x}^{(i)}) = \frac{\tilde{p}_1(\mathbf{x}_0)}{\tilde{p}_0(\mathbf{x}_0)} \frac{\tilde{p}_2(\mathbf{x}_1)}{\tilde{p}_1(\mathbf{x}_1)} \cdots \frac{\tilde{p}_{N-1}(\mathbf{x}_{N-2})}{\tilde{p}_{N-2}(\mathbf{x}_{N-2})} \frac{\tilde{p}_N(\mathbf{x}_{N-1})}{\tilde{p}_{N-1}(\mathbf{x}_{N-1})}, \tag{A.11}$$

where now the average of the importance weights converges to the normalisation constant for our target probability density function, $\int p_N(\mathbf{x}) \approx \frac{1}{N} \sum \tilde{w}$. Noting that the normalisation constants cancel for all terms except the target $\tilde{p}_N(\mathbf{x}_{N-1})$ in the nominator, and the proposal $\tilde{p}_N(\mathbf{x})$ in the denominator, where the latter is already normalised.

A.2 Proof that minimising variance in importance weights is equivalent to minimising α-divergence with $\alpha = 2$

If we inspect the variance in the importance weights

$$\begin{aligned} \text{var}(w) &= \mathbb{E}_q\left[w^2\right] - \mathbb{E}_q\left[w\right]^2 \\ &= \mathbb{E}_q\left[w^2\right] - \left[\int q(\mathbf{x})\frac{\tilde{p}(\mathbf{x})}{q(\mathbf{x})}d\mathbf{x}\right]^2 \\ &= \mathbb{E}_q\left[w^2\right] - Z_p^2 \\ &\propto \mathbb{E}_q\left[w^2\right], \end{aligned} \tag{A.12}$$

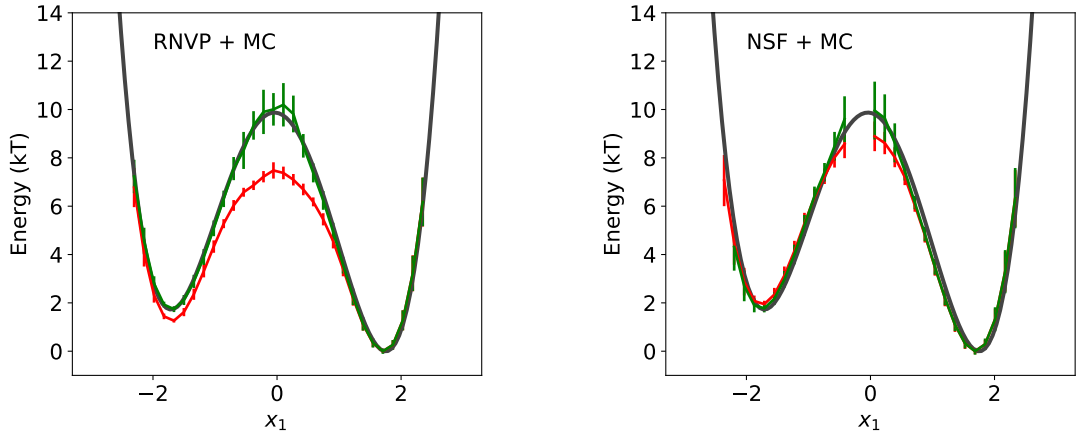
where Z_p is the normalisation constant for $\tilde{p}$. Through comparison to Equation 3.2 that the final line of the above equation is proportional to α-divergence with $\alpha = 2$. This proof is similar to the proof given by Minka et al. (2005) - where he proves that minimising α-divergence minimises the variance in the estimate of Z_p.

A.3 Critique of metric used to asses performance of the Double Well problem in literature

Wu et al. (2020) measure the performance of their Boltzmann generators on the Double Well problem by dividing the space into a set of bins, and then comparing importance sampled weighted estimated of free energy differences between the lowest energy state and these various bins. However this is a poor metric, as since to compute an overall score Wu et al. (2020) average across bins, giving the accuracy of the estimate of each bin equal weight, meaning that the optimal proposal distribution does not have to look similar to the Double Well distribution to obtain a low bias, low variance estimate. To demonstrate this we show that if we place a uniform distribution that within the zone of interest [-2.5, 2.5] in the first dimension, and then use the same in the other distribution which is marginalised, that we are able to outperform all of their models using their test (see Table A.1, Figure A.1), even though a uniform distribution is a poor approximation for the Double Well problem. We have simply plugged in a uniform distribution into their plotting and scoring code to show this.

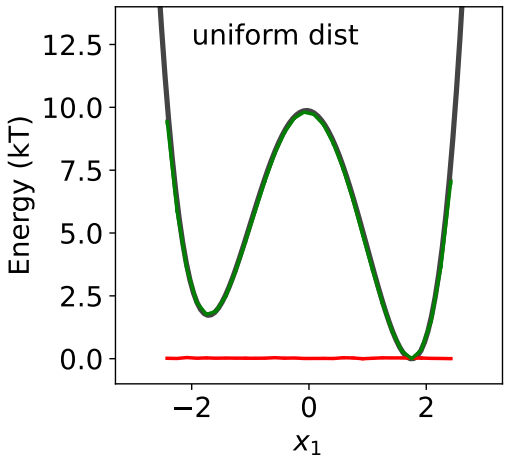
Table A.1 Comparison of the top performing models in Wu et al. (2020) to a uniform distribution using their measurement of performance based on estimating energy differences in bins over the first dimension. We see that a uniform distribution has lower bias and variance than their models, even though it is a poor approximation of the target density, illustrating the poor choice of performance metric.

	bias	std	$\sqrt{bias^2 + var}$
RNVP + MC	0.2 ± 0.1	0.6 ± 0.1	0.6 ± 0.1
NSF + MC	0.1 ± 0.1	0.6 ± 0.2	0.6 ± 0.2
Uniform	0.08 ± 0.01	0.06 ± 0.002	0.1 ± 0.001



(a) RealNVP Boltzmann generator + Metropolis Monte Carlo

(b) SNF Boltzman generator + Metropolis Monte Carlo



(c) Uniform distribution

Fig. A.1 In this plot we estimate free energy differences using importance sampling. The green line shows the importance weighted estimate of the free energy, the red shows the estimate of free energy difference if we simply bin samples without importance sampling. Thus the red line shows the shape of the proposal distribution. The top two plots are taken directly from Wu et al. (2020) for their best performing models. In the 3rd plot we see that a uniform distribution achieves greater accuracy, where the bounds are so tight that they are barely visible. To create this plot we simply plugged in a uniform distribution to the code provided by Wu et al. (2020).

Appendix B

Further results

B.1 Conventional Boltzmann Generators on the Many Well Problem

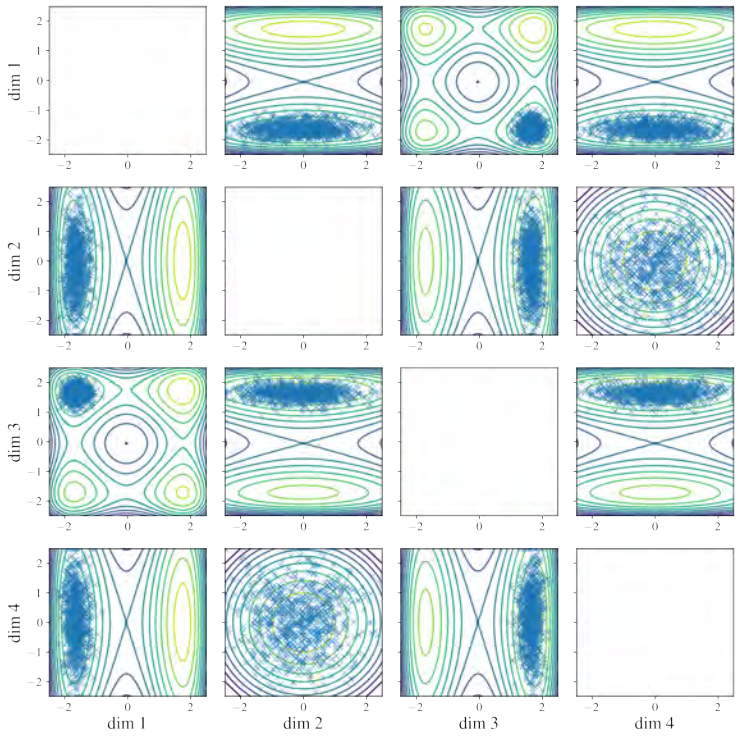


Fig. B.1 Standard Boltzmann Generators are not able to train successfully on the higher dimensional Many Well Boltzmann distribution. We illustrate this in this plot by training a Boltzmann Generator using the code given in Wu et al. (2020) on an 8 dimensional Many Well problem, and show that it fits a subset of the modes. In this plot we show samples (little blue crosses) from pairs of marginal distributions for 4 of the 8 dimensional, where the huge hugging behavior is clearly visible. Counters in the background are from the target Many-Well Boltzmann distribution.

B.2 32 dimensional Many Well Problem

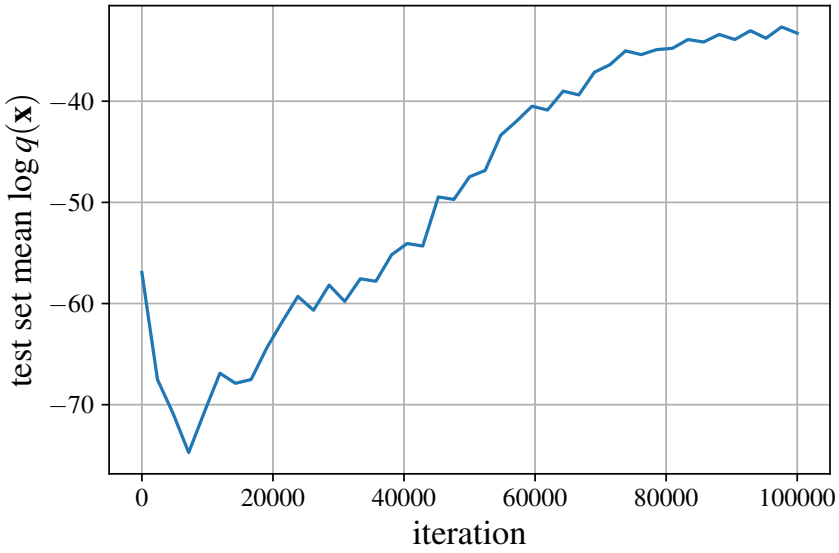


Fig. B.2 Test-set performance during training on the 32 Double Well Problem. We see a steady increase in the probability the proposal assigns to the test-set throughout training, ensuring that no mode is missed, meaning that we can trust the effective sample size statistic.

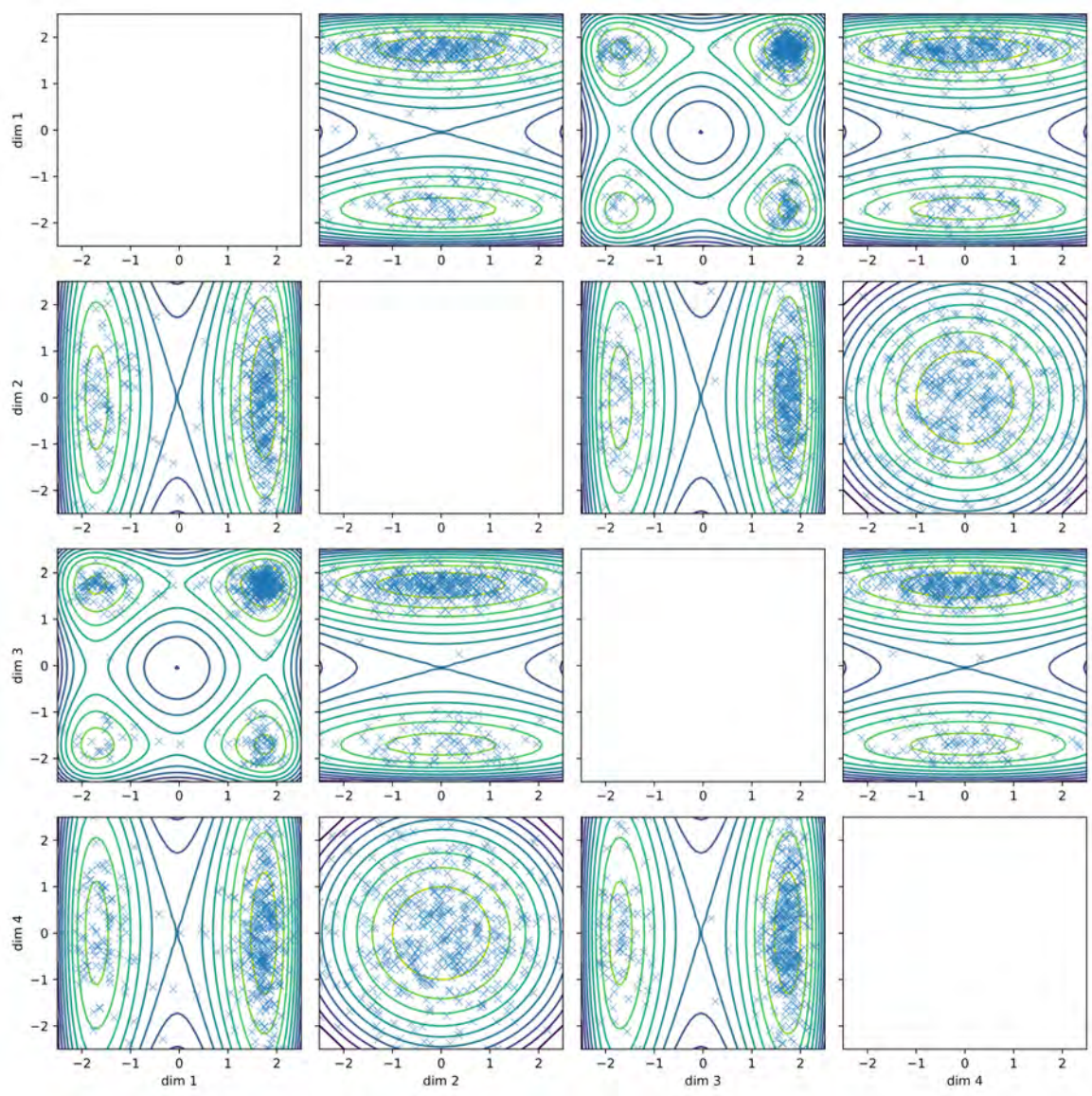


Fig. B.3 Marginal distribution for pairs of dimensions on the 32 Double Well Problem. Contours show the target probability density, and points show samples from the FAB model after AIS. We show a sub-set of the marginal pairs instead of the 32-by-32 plot (for simplicity). However, the 32-by-32 plot shows the same general pattern with all modes containing samples.

Appendix C

Model Details

C.1 Mixture of Gaussians Problem

Parameter	Value
Flow type	RealNVP + ActNorm
Number of flow layers	30
Batch size	100
Iterations	1000
Optimizer	AdamW
Learning rate	0.0005

Table C.1 Model parameters for mixture of Gaussians Problem. This is used for all models within Section 4.3 except for the SNF and Boltzmann Generator which used the code from (Wu et al., 2020)

C.2 Many Well Problem

Number of target function evaluations is given by,

$$\begin{aligned} \text{n-eval} &= \text{iterations} \times \text{batch-size} \times \text{HMC-steps} \times \text{n-intermediate-distributions} \\ &= 10^5 \times 10^3 \times 5 \times 2 \\ &= 10^9 \end{aligned} \tag{C.1}$$

Parameter	Value
Flow type	IAF + ActNorm
Number of flow layers	20
Batch size	1000
Iterations	100,000
Optimizer	AdamW
Learning rate	0.0005

Table C.2 Model parameters for mixture of Many Well Problem. This is used for all models within Section 4.4 and Section 4.4.