Amortized Inference and Model Learning for Probabilistic Programming

Tuan Anh Le

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2019

Abstract

Probabilistic modeling lets us infer, predict and make decisions based on incomplete or noisy data. The goal of *probabilistic programming* is to automate inference in probabilistic models that are expressed as probabilistic programs—programs that can draw random values and condition the resulting stochastic execution on data. The ability to define models using programming constructs such as recursion, stochastic branching, higher-order functions, and highly-developed simulation libraries allows us to more easily express and perform inference in models that have simulators, a dynamic number of latent variables, highly structured latent variables or nested probabilistic models. The key to success of probabilistic programming is efficient inference and model learning in such models. The most powerful black-box approximate inference algorithms for probabilistic programs are either sampling-based (Monte Carlo simulations) or optimization-based (variational methods). In both cases, one must re-run the inference algorithm for new data. Hence, our first goal is developing algorithms for *amortized inference* which, here, refers to reducing the run-time cost of inference by pre-training a neural network that maps from observed data to efficient parameters for performing fast, repeated inference. Our second goal is developing algorithms for *model learning* that are advantageous for two classes of models that one might typically want to write as probabilistic programs: models of sequential data and models that contain discrete latent variables. In this thesis, we discuss the general topic of Bayesian machine learning and probabilistic programming before moving on to our methodological contributions in amortized inference and model learning.

Acknowledgements

I would like to thank my supervisor, Frank Wood, for introducing me to the fascinating world of probabilistic programming and machine learning research, for guiding me through it, for generously discussing ideas with me, and for being a mentor I can always turn to for advice.

I would like to thank Yee Whye Teh, for agreeing to be my second supervisor towards the end of my DPhil. Thank you for allowing me to interact with your lab, for your generous time and for your career advice.

I would like to thank my collaborators. Atılım Güneş Baydin: thank you for your friendship and for working together early in my DPhil. I have learned so much from you. Maximilian Igl, Tom Rainforth, Tom Jin: thank you for all the hard work that went into auto-encoding sequential Monte Carlo. Adam Kosiorek and Siddharth N: thank you for revisiting reweighted wake-sleep with me.

I would like to thank my colleagues and friends in Oxford for all the fun inside and outside of the lab: Atılım Güneş Baydin, Rob Cornish, Piotr Czaban, Neil Dhir, Adam Goliński, Bradley Gram-Hansen, Will Harvey, Maximilian Igl, Tom Jin, Hyunjik Kim, Adam Kosiorek, Mario Lezcano, David Martínez Rubio, Siddharth N, Brooks Paige, Yura Perov, Tom Rainforth, Mike Teng, David Tolpin, Jan-Willem van de Meent, Andrew Warrington, Stefan Webb, Hongseok Yang, Yuan Zhou, and Rob Zinkov.

I would like to thank my friends for their advice and support: Matej Balog, Matej Hamaš, Alexander Ivanov, Hai Mac Duy, Trang Duong Minh, Rangarajan Ramesh, Abhay Raji Soorya, Marek Šebo, Shaan Tehal, Šimon Váry and Mohammad Abrar Wadud.

Finally and most importantly, I would like to thank my parents, Quang Le Hong and Loc Tran Thi An, my sister, Thuy Anh Le Nu, and my wife, Nguyet Anh Nguyen, for their unwavering support and love. You are my role models and I am what I am thanks to you.

Contents

1 Introduction

How do we create systems that can gain understanding from limited data and use this understanding to act intelligently? Deep learning approaches have shown great potential in the applications in many domains including computer vision (Krizhevsky et al., 2012), speech recognition (Graves et al., 2013), machine translation (Bahdanau et al., 2014) and game-playing (Mnih et al., 2015; Silver et al., 2017). Success in these tasks rests primarily on the availability of large datasets of inputs and corresponding ground truth labels which are used to fit the parameters of deep neural networks. Despite reaching extraordinary performance on many tasks, doing so for all of them only via labeled data is infeasible. It is unlikely that achieving human-level intelligence relies solely on collecting enough data to learn predictive models for all possible input-output relationships. Humans seem to possess an understanding of the world that allows us to learn from limited data and generalize to related tasks by reusing this knowledge without having to learn from scratch.

One proposed computational account of human cognition suggests that this understanding of the world is in the form of a probabilistic generative model where beliefs about the world, represented as probability distributions, are updated in light of evidence via the application of Bayes rule—a process known as Bayesian inference. This hypothesis assumes that humans don't learn from scratch, but instead possess so-called *core knowledge*. This is knowledge that is already present

in humans, and even many animals, at the first encounter with tasks of interest. For instance, babies understand that objects are permanent and solid (Baillargeon et al., 1985), or that agents have goals and tend to take actions to achieve these goals efficiently (Gergely et al., 1995). As a consequence of this hypothesis, we should incorporate as much as possible of the core knowledge as priors in probabilistic models in order to build intelligent systems (Tenenbaum et al., 2011).

Alternatively, one could argue that to tease out knowledge from limited data, we must somehow encode assumptions about the kind of knowledge we are to learn, whether implicitly or explicitly. Or as MacKay (2003, page 26) puts it, "you cannot do inference without making assumptions." In Bayesian statistics, we encode such assumptions as a probabilistic model, and inference, prediction and decision-making are simply the results of manipulating probability distributions and utility functions. These ideas have been widely adopted in machine learning with many important applications in unsupervised learning such as clustering (Rasmussen, 2000), time-series analysis (Durbin and Koopman, 2012) and dimensionality reduction (Tipping and Bishop, 1999).

The introduction of probabilistic graphical models (Pearl, 1988) has been key to the widespread application of Bayesian ideas in machine learning. Here, a probabilistic model is represented by a graph whose nodes correspond to random variables in the model and edges to conditional probability distributions. However, the expressiveness of graphical models is limited because they can only canonically express models with finite number of random variables. In this thesis, we focus on probabilistic programming (Gordon et al., 2014), which is a way of representing probabilistic models as programs with inference resulting in a distribution over program executions instead of just a set of variables as in the case of graphical models. Probabilistic programming aims to be a tool in the Bayesian statistics pipeline that allows modelers to focus on modeling, without having to worry about inference, and designers of inference algorithms to have their algorithms be implemented once and be widely used. Our interest in probabilistic programming stems from its ability to easily express both cognitively and statistically motivated generative

models that are difficult to express as graphical models. Such models typically have one or more of the following elements:

- **Simulator**. Generative models which contain a simulator, typically written using a programming language, are easily written as probabilistic programs. Applications include intuitive physics (Battaglia et al., 2013), vision-as-inverse-graphics (Mansinghka et al., 2013; Le et al., 2017b) as well as engineering design (Rainforth et al., 2016a) and particle physics (Baydin et al., 2018).

- **Dynamic number of variables**. We often do not know how many things there are in the world. Maintaining a dynamically changing number of random variables is easily handled in a probabilistic program, for example using a loop with a stochastic end condition. Applications include one-shot concept learning (Salakhutdinov et al., 2012), object-tracking (Neiswanger et al., 2014) as well as Bayesian nonparametrics for clustering, information retrieval and text modeling (Teh et al., 2005).

- **Structured latent variables**. Latent variables which correspond to data structures like trees, grids, directed graphs and even program text are most naturally expressed using a programming language. Applications include learning hand-written character concepts by inferring a drawing program (Lake et al., 2015), inference of interpretable samplers using a generative model of probabilistic programs (Perov and Wood, 2016), and statistical parsing (Manning et al., 1999).

- **Nested models**. Reasoning about reasoning may require nested inference (Stuhlmüller and Goodman, 2014; Rainforth, 2018). Nested inference subroutines are easier to express and reason about in a probabilistic programming framework. Applications include intuitive psychology (Baker et al., 2009; Evans et al., 2017).

Despite this representational power of probabilistic programming, we don't escape the difficulties of inference and modeling. This thesis is about using neural

networks for speeding up and automating inference and model learning in the kinds of models one usually needs to use probabilistic programming for. First, inference in complex models is typically slow and must be re-run for every new observed data. To this end, we focus on *amortized inference* (Gershman and Goodman, 2014). Given a probabilistic program, we "amortize" the cost of performing inference by training a neural network, called the *inference network* (also known as recognition network, encoder, or inference compilation artifact), which helps us in performing fast repeated inference during test time. We only need to perform the amortization step once. Once this is done, the inference network can be reused repeatedly for various new observations. Second, if the model is not well-specified, inference cannot be used for making predictions and decisions in the real world. However, most approaches to modeling involve hand-tuning. To this end, we focus on *model learning* by extending algorithmic advances in deep generative modeling (Kingma and Welling, 2014; Rezende et al., 2014) to probabilistic programming with the eventual goal of being able to write probabilistic programs in such a way that certain parts of it are unspecified—typically functions that can be approximated with neural networks—and learned from data.

Thesis structure. Chapter 2 provides background for Bayesian machine learning, probabilistic programming and several inference algorithms. Chapter 3 presents an approach to amortized inference in the context of higher-order probabilistic programming languages. Chapter 4 is about two approaches to model learning that are particularly suitable for probabilistic programming. Chapter 5 summarizes our work and outlines future research directions.

This is an integrated thesis based on the following publications:

- Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 1338–1348, 2017a,

- Tuan Anh Le, Atılım Güneş Baydin, Robert Zinkov, and Frank Wood. Using synthetic data to train neural networks is model-based reasoning. In *30th International Joint Conference on Neural Networks*, pages 3514–3521. IEEE, 2017b,

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

- Tuan Anh Le, Adam R. Kosiorek, N. Siddharth, Yee Whye Teh, and Frank Wood. Revisiting reweighted wake-sleep. *arXiv preprint 1805.10469*, 2018b.

Contents of Section 3.2 are based on the first two publications. Contents of Sections 4.2 and 4.3 are based on the third and fourth publication respectively. The contributions of each paper's authors are given at the end of the corresponding sections. All other text and figures in this thesis are mine.

2

Bayesian Machine Learning and Probabilistic Programming

In this chapter, we provide background for Bayesian machine learning and probabilistic programing. The ideas in this chapter form the basis of the remaining parts of the thesis on amortized inference and model learning.

Bayesian machine learning is a simple but powerful framework for formalizing unsupervised learning. We cannot perform inference from unlabeled data without making assumptions, implicit or explicit. Bayesian machine learning lets us make these assumptions explicitly, by placing a joint probability distribution on unobserved quantities—which we will call latent variables—that we want to infer and observed data. This joint probability distribution consists of a prior distribution and a likelihood. The prior distribution represents our initial belief about the latent variables, before seeing data and the likelihood links the latent variables to observed data through a conditional probability distribution. Inference in this framework corresponds to updating our prior belief. Our belief about the latent variables, after seeing data, is represented by the conditional distribution of latent variables given observed data, known as the posterior distribution. That subjective beliefs should be represented as probability distributions is often justified through Cox

axioms (Cox, 1946) or the "Dutch book argument" (de Finetti, 1931, 1937). For a good introduction to the philosophy of Bayesian inference, see (Jaynes, 2003).

Probabilistic programming advocates representing probabilistic models as stochastic programs. Inference in such models corresponds to probabilistically conditioning the program executions on observed data. The posterior distribution is then a probability distribution on program execution traces. Probabilistic programs can express and automate inference in a rich class of probabilistic models, including ones not expressible using Bayesian networks. Why probabilistic programming? First, probabilistic programming is a tool that can greatly increase the productivity of Bayesian modelers, allowing them to write models as programs without having to worry about the inference. Some probabilistic models—like hierarchical Bayesian models over highly-structured objects which can be used to computationally explain concept learning (Lake et al., 2015)—can be difficult to write without probabilistic programing. Second, inference algorithms can be anchored to a specific language specification, allowing inference algorithm writers to have their algorithms widely adopted.

Section 2.1 provides background for Bayesian machine learning. Section 2.2 provides background for probabilistic programming systems. Section 2.3 introduces two important general-purpose inference algorithms that are often used in probabilistic programming systems.

2.1 Bayesian Machine Learning

In this section, we provide background about Bayesian machine learning and establish the notation which will be used throughout the thesis. The central question is: Given unlabeled data $x \in \mathcal{X}$, how can we make meaningful inferences about underlying patterns $z \in \mathcal{Z}$? This is widely applicable to many domains, depending on what we choose to be $\mathcal{X}$ and $\mathcal{Z}$ and what data x is available. Applications of Bayesian inference include clustering, dimensionality reduction, speech recognition, intent recognition, topic modeling or medical diagnosis.

Section 2.1.1 introduces basic probability theory and the notation. Section 2.1.2 introduces inference and several examples of applications. Section 2.1.3 introduces learning in probabilistic models. For an in-depth textbook treatment of the subject, see e.g. (Gelman et al., 2013), (Murphy, 2012) or (Bishop, 2006). For a general review of recent progress and trends in the field, see e.g. (Ghahramani, 2015).

2.1.1 Basic Probability Theory

We use random variables to model quantities that are not fixed. Formally, we start by defining probability space $(\Omega, \mathcal{F}, \mathbb{P})$ consisting of a sample space Ω which is a set of all possible outcomes, a set of events $\mathcal{F}$ where each event $E \in \mathcal{F}$ is a subset of Ω to which we can assign a probability (i.e. it is measurable), and a probability measure $\mathbb{P} : \mathcal{F} \to [0, 1]$ which is a function whose output $\mathbb{P}(E)$ represents the probability of $E \in \mathcal{F}$. A $\mathcal{Z}$-valued random variable Z is a measurable function from Ω to the state space $\mathcal{Z}$ with an associated set of events $\mathcal{F}_Z$. That Z is a measurable function means that for all events $A \in \mathcal{F}_Z$, $\{\omega \in \Omega : Z(\omega) \in E\} \in \mathcal{F}$. The probability of $Z \in A$, denoted $\mathbb{P}(Z \in A)$, is the value of $\mathbb{P}(\{\omega \in \Omega : Z(\omega) \in E\})$. We typically assume that there exists a probability density $p_Z : \mathcal{Z} \to [0, \infty)$ such that

$$\mathbb{P}(Z \in A) = \int_A p_Z(z) \, \mathrm{d}z, \tag{2.1}$$

where $\int_A \cdot \, \mathrm{d}z$ is an integral with respect to the reference measure of the density p_Z over the set A. In case of continuous random variables with values in $\mathcal{Z} = \mathbb{R}^d$, the reference measure is typically the Lebesgue measure. In case of discrete random variables where $\mathcal{Z}$ is a countable set, the reference measure is the counting measure and the integral turns into a summation. The probability of $\{Z \in \mathcal{Z}\}$, and hence $\int_{\mathcal{Z}} p_Z(z) \, \mathrm{d}z$, is one.

For two random variables, say Z and X, we assume that there exists a joint probability density $p_{Z,X} : \mathcal{Z} \times \mathcal{X} \to [0, \infty)$. The probability of (Z, X) being in the measurable set $A \subseteq \mathcal{Z} \times \mathcal{X}$ is $\mathbb{P}((Z, X) \in A) = \int_A p_{Z,X}(z, x) \, \mathrm{d}z \, \mathrm{d}x$. The marginal probability density of Z is obtained using the sum rule which integrates out x:

$$p_Z(z) = \int p_{Z,X}(z, x) \, \mathrm{d}x. \tag{2.2}$$

The conditional probability density $p_{X|Z}(x|z)$ is a function $p_{X|Z} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$ which, for a fixed value of z, is a probability density on x. For any $z \in \mathcal{Z}$, $\mathbb{P}(X \in A|z) = \int_A p_{X|Z}(x|z)\,\mathrm{d}x$ and $\mathbb{P}(X \in \mathcal{X}|z) = 1$. The product rule relates $p_{X|Z}$ to p_Z and $p_{Z,X}$:

$$p_{Z,X}(z, x) = p_Z(z)p_{X|Z}(x|z). \tag{2.3}$$

We suppress the subscripts to avoid notational clutter when it is clear which probability density is meant from its arguments or the context. We will also use lowercase letters to denote random variables when the distinction between a random variable and a value in its range is clear.

Bayes rule is obtained from the application of the product and sum rules. The marginal distribution of X is obtained through the sum rule $p(x) = \int p(z)p(x|z)\,\mathrm{d}z$. The product rule lets us relate the marginal density $p(x)$ and the conditional density $p(z|x)$ to the joint probability density as $p(z, x) = p(x)p(z|x)$. Equating this with (2.3) results in the Bayes rule:

$$p(z|x) = \frac{p(x|z)p(z)}{\int p(z)p(x|z)\,\mathrm{d}z}. \tag{2.4}$$

Expected values allow us to summarize information in a random variable and make a decision based on it. Given a function $f : \mathcal{Z} \to \mathbb{R}^d$, the expected value of $f(Z)$ is

$$\mathbb{E}[f(Z)] = \int_{\mathcal{Z}} f(z)p(z)\,\mathrm{d}z. \tag{2.5}$$

When f is the identity function, we obtain the expected value, or the mean, of Z.

We obtain particular realizations of random variables through sampling. We denote "z is a sample of a random variable Z" as $z \sim p(z)$. We discuss in Section 2.3.1.1 how sampling can be used to estimate expected values.

We denote known probability distributions and their densities using their names. For example, a normal distribution is denoted as $\mathrm{Normal}(\mu, \sigma^2)$. Its density evaluated at z is denoted as $\mathrm{Normal}(z|\mu, \sigma^2)$.

2.1.2 Inference in Latent Variable Models

How do we make inferences about unobservable quantities given observable quantities? For instance, we want to infer the speed of a car given noisy radar measurements. The radar measurement is dependent on the speed of the car, but there is a small amount of noise that prevents us from getting the exact speed. Or we want to infer inputs to an engineering simulator that results in a particular output. Or we could infer and predict underlying patterns based on time series, like financial data. Such predictions let us make decisions that take uncertainty into consideration. These are some examples which can be formalized as latent variable models in which inference can be viewed as applying Bayes rule.

In a latent variable model, we denote the unobservable quantity by z and the observable quantity by x. We also call z the latent variable and x the observed variable, or sometimes just data. The model consists of a probability distribution over z called the *prior*, and a conditional probability distribution of x given z, called the *likelihood*. The prior distribution, $p(z)$, represents our subjective belief about z before observing x. The likelihood of z, $p(x|z)$, represents the plausibility of the z in light of observed data x. These two components define the joint probability distribution on (z, x) whose density is given by the product rule

$$p(z, x) = \underbrace{p(z)}_{\text{prior}} \underbrace{p(x|z)}_{\text{likelihood}}. \tag{2.6}$$

Given an actual value of x, we can apply Bayes rule to get the *posterior distribution*:

$$p(z|x) = \frac{p(z)p(x|z)}{p(x)}. \tag{2.7}$$

The posterior distribution represents our subjective belief about z after observing x. Bayesian inference is the process of obtaining the posterior distribution.

The denominator in (2.7) is called the *marginal likelihood* or *evidence*:

$$p(x) = \int p(z)p(x|z)\,\mathrm{d}z. \tag{2.8}$$

It is the expected value of the likelihood—or the plausibility of z—under the prior distribution $p(z)$. We discuss in Section 2.1.3 how this quantity can be used to perform model selection.

2.1.2.1 Gaussian Unknown Mean

Let's consider a latent variable model which we will refer to as the *Gaussian unknown mean*. It can be used to model a single noisy radar measurement of the car's speed. Let's model the car's speed using a latent variable z whose prior is a normal distribution with mean μ_z and standard deviation σ_z:

$$p(z) = \text{Normal}(z|\mu_z, \sigma_z^2).$$

The noisy radar measurement x is modeled as a normal distribution centered on z, with a standard deviation σ:

$$p(x|z) = \text{Normal}(x|z, \sigma^2).$$

The posterior density can be expressed in closed form as

$$p(z|x) = \text{Normal}(z|\mu_{\text{post}}, \sigma_{\text{post}}^2),$$

where $\sigma_{\text{post}}^2 = 1/(1/\sigma_z^2 + 1/\sigma^2)$ and $\mu_{\text{post}} = \sigma_{\text{post}}^2(\mu_z/\sigma_z^2 + x/\sigma^2)$. The evidence of this model is also a normal density evaluated at x:

$$p(x) = \text{Normal}(x|\mu_z, \sigma_z^2 + \sigma^2).$$

A prior-likelihood pair for which the posterior distribution is in the same family of distributions as the prior is called *conjugate*. Posterior density and evidence in conjugate models can be computed in closed form.

Inference in the Gaussian unknown mean model can be intuitively understood by parameterizing the normal distribution using the reciprocal of the variance which is called precision (Figure 2.1). Let the prior, likelihood and posterior precisions be defined as $\lambda_z := 1/\sigma_z^2$, $\lambda := 1/\sigma^2$ and $\lambda_{\text{post}} := 1/\sigma_{\text{post}}^2$. The posterior precision is a sum of the prior and likelihood precisions: $\lambda_{\text{post}} = \lambda_z + \lambda$. This intuitively means that the more data we get, the more certain our posterior. The posterior mean is a weighted sum of the prior mean and data, where the weights are proportional to the prior and likelihood precisions: $\mu_{\text{post}} = \frac{\lambda_z}{\lambda_{\text{post}}}\mu_z + \frac{\lambda}{\lambda_{\text{post}}}x$.

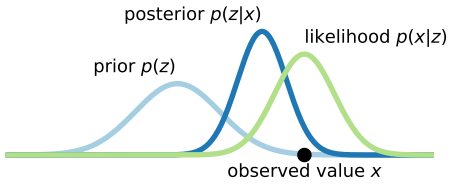


Figure 2.1: Bayesian inference in the Gaussian unknown mean model. The posterior precision is a sum of the prior and likelihood precisions. The posterior mean is a weighted sum of the prior mean and data, where the weights are proportional to the prior and likelihood precisions.

2.1.2.2 Gaussian Mixture Model

Clustering is a canonical example of unsupervised learning. Given a set of unlabeled data-points (without their corresponding cluster index), we seek to assign each one to a cluster. Each cluster is characterized by a per-cluster parameter, which can be used as a representation of data-points belonging to that cluster. Clustering lets us ask questions such as "Are these two data-points in the same cluster?", "How many points are in a particular cluster?", or given a new data-point "Which cluster does it belong to?". We present an approach to clustering based on Bayesian inference in a Gaussian mixture model (GMM).

Inference. In a GMM, we model N unlabeled D-dimensional data-points $x := x_{1:N}$ as being generated from K clusters. The latent variable $z := (\pi, z_{1:N}, \mu_{1:K}, \Sigma_{1:K})$ consists of a mixture probability vector $\pi \in \mathbb{R}^K$, a per-data cluster identity $z_n \in \{1, \ldots, K\}$, per-cluster means $\mu_k \in \mathbb{R}^D$, and per-cluster covariance matrices $\Sigma_k \in \mathbb{R}^{D \times D}$. Inference in a GMM cannot be performed in closed-form like in the Gaussian unknown mean model and we must resort to approximate inference algorithms such as Gibbs sampling (Murphy, 2012). We choose the following parameterization of a GMM since it allows computing closed-form updates for a Gibbs sampler:

$$
\begin{aligned}
p(z) &= p(\pi) \prod_{n=1}^{N} p(z_n|\pi) \prod_{k=1}^{K} p(\mu_k, \Sigma_k) \\
&= \text{Dirichlet}(\pi|\alpha) \prod_{n=1}^{N} \text{Categorical}(z_n|\pi) \prod_{k=1}^{K} \text{NIW}(\mu_k, \Sigma_k|\mu_0, \lambda, \Psi, \nu) \\
p(x|z) &= \prod_{n=1}^{N} p(x_n|z_n) = \prod_{n=1}^{N} \text{Normal}(x_n|\mu_{z_n}, \Sigma_{z_n}).
\end{aligned}
$$

This model has the following hyper-parameters. The Dirichlet distribution is parameterized by the positive concentration parameter $\alpha \in (0, \infty)^K$. "NIW" is a Normal-inverse-Wishart distribution over positive semi-definite matrices which is parameterized by the location $\mu_0 \in \mathbb{R}^D$, $\lambda > 0$, scale $\Psi \in \mathbb{R}^{D \times D}$, and degrees of freedom $\nu > D - 1$.

Queries about data. We can use the posterior expectation to answer queries about our data. "What is the probability of x_i and x_j being in the same cluster?" can be answered through $\mathbb{E}_{p(z|x)}[\mathbb{1}(z_i = z_j)]$. "How many points are in cluster k?" is answered through $\mathbb{E}_{p(z|x)}[\sum_{n=1}^{N} \mathbb{1}(z_n = k)]$. Given samples of the posterior distribution, for example through Gibbs sampling, the posterior expectations are estimated using the Monte Carlo method (see Section 2.3.1.1).

Learning. Instead of performing inference on all latent variables, we often *learn* a subset thereof through maximum likelihood while inferring the rest. In a GMM, the per-cluster parameters and cluster probabilities $(\mu_{1:K}, \Sigma_{1:K}, \pi)$ can be learned by maximizing the maximum marginal likelihood:

$$(\mu^*_{1:K}, \Sigma^*_{1:K}, \pi^*) = \underset{\mu_{1:K}, \Sigma_{1:K}, \pi}{\operatorname{argmax}} \; p(x_{1:N} | \mu_{1:K}, \Sigma_{1:K}, \pi).$$

Direct evaluation of the marginal likelihood requires an intractable summation

$$p(x_{1:N} | \mu_{1:K}, \Sigma_{1:K}, \pi) = \sum_{z_{1:N}} p(x_{1:N}, z_{1:N} | \mu_{1:K}, \Sigma_{1:K}, \pi).$$

We instead resort to the expectation maximization (EM) (Dempster et al., 1977) which is a class of algorithms for iteratively maximizing the marginal likelihood when inference in the model is tractable.

Given the learned parameters $(\mu^*_{1:K}, \Sigma^*_{1:K}, \pi^*)$, inference of remaining latent variables $z_{1:N}$ becomes tractable:

$$\begin{aligned} p(z_{1:N} | x_{1:N}, \mu^*_{1:K}, \Sigma^*_{1:K}, \pi^*) &= \prod_{n=1}^{N} p(z_n | x_n, \mu^*_{1:K}, \Sigma^*_{1:K}, \pi^*) \\ &= \prod_{n=1}^{N} \frac{\text{Categorical}(z_n | \pi^*) \text{Normal}(x_n | \mu^*_{z_n}, \Sigma^*_{z_n})}{\sum_{k=1}^{K} \text{Categorical}(k | \pi^*) \text{Normal}(x_n | \mu^*_k, \Sigma^*_k)}. \end{aligned}$$

In Figure 2.2, we show inference in two variants of a GMM. One where the per-cluster parameters and cluster probabilities are learned and we only infer the cluster assignments and one where we infer all latent variables.

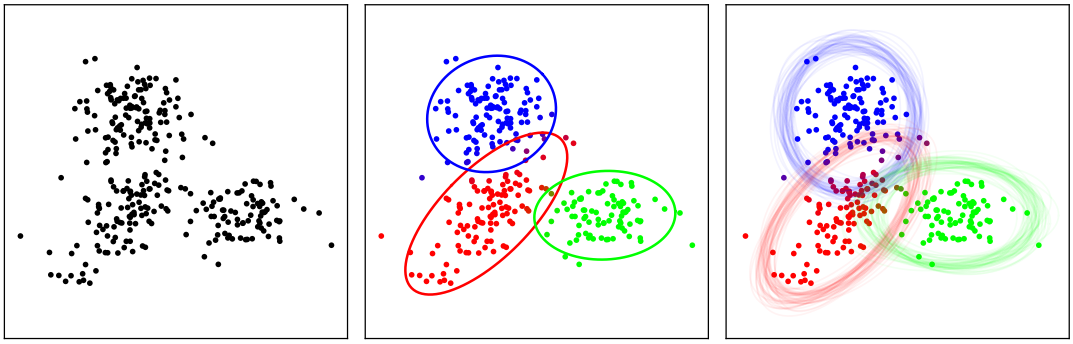


Figure 2.2: Clustering using the Gaussian mixture model. (Left) Unlabeled two-dimensional synthetic data. (Middle) Inference of cluster identities $z_{1:N}$ with $K = 3$. The rest of the latent variables $(\mu_{1:K}, \Sigma_{1:K}, \pi)$ is learned through maximum marginal likelihood. Per-cluster parameters (μ_k, Σ_k) are represented as an ellipse centered on μ_k which covers 95% of the probability mass under Normal(μ_k, Σ_k). Each ellipse is colored according to the cluster identity k. Each data-point is colored according the to the posterior distribution $p(z_n|x_n)$ which is taken to be the RGB color vector. (Right) Inference of all latent variables using Gibbs sampling. Posterior samples of μ_k, Σ_k are represented as multiple ellipses.

2.1.2.3 State Space Models

How can we make decisions based on sequential data? For example, we may have financial data such as prices of a commodity. We want to predict how the price will evolve in the future and make a decision whether to buy or sell. Or we might have noisy sensor measurements of a rocket's position. How can we estimate its true position so that we can accurately control it? We could just treat the data-points as independent and estimate the position using, for example, the Gaussian unknown mean model. This, however, ignores important information about the dynamics of the rocket and as a result we can end up with a less accurate estimate of the position.

General state-space models. We denote each data-point as x_t and observe a sequence $x := x_{1:T}$ of T data-points. State-space models (SSMs) are a class of probabilistic models used to model sequences. In an SSM there is a latent

state z_t corresponding to each data-point x_t. In general, the joint density of an SSM factorizes as

$$p(z_{1:T}, x_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t|z_{1:t-1}, x_{1:t-1}) \prod_{t=1}^{T} p(x_t|z_{1:t}, x_{1:t-1}), \tag{2.9}$$

where $p(z_1)$ is the *initial distribution*, $p(z_t|z_{1:t-1}, x_{1:t-1})$ is the *transition distribution* and $p(x_t|z_{1:t}, x_{1:t-1})$ is the *emission distribution*.

Markov state-space models. The generality of this factorization comes at a cost. Since the transition and emission distributions depend on many variables, these distributions have more parameters that must be learned. In order to make learning more tractable, we often assume a Markovian factorization for the latent variable sequence. This means that $p(z_t|z_{1:t-1}, x_{1:t-1}) = p(z_t|z_{t-1})$. We also assume that the emission distribution of x_t only conditions on the current state z_t, $p(x_t|z_{1:t}, x_{1:t-1}) = p(x_t|z_t)$. The joint density of a Markovian SSM is

$$p(z_{1:T}, x_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t|z_{t-1}) \prod_{t=1}^{T} p(x_t|z_t). \tag{2.10}$$

Linear Gaussian state space models (LGSSMs) (also known as linear dynamical systems) are a class of Markovian SSMs where each latent variable $z_t \in \mathbb{R}^{D_z}$ is normally distributed with the mean being a linear function of z_{t-1} and each data-point $x_t \in \mathbb{R}^{D_x}$ is normally distributed with the mean being a linear function of z_t:

$$p(z_1) = \text{Normal}(z_1|\mu, \Sigma) \tag{2.11}$$

$$p(z_t|z_{t-1}) = \text{Normal}(z_t|Az_{t-1} + b, S) \tag{2.12}$$

$$p(x_t|z_t) = \text{Normal}(x_t|Cz_t + d, R). \tag{2.13}$$

Here, the parameters $\theta := (A, b, S, C, d, R)$ can be learned by maximizing the marginal likelihood $p(x_{1:T}|\theta)$ using the EM algorithm (Murphy, 2012).

Higher order Markov state-space models. In a Markov SSM, z_t is conditionally independent of $z_{1:t-2}$ given z_{t-1}. This modeling assumption is often incorrect which can result in bad inferences and predictions. A compromise between a general SSM which has too many parameters to learn and a Markov SSM whose modeling assumptions are too strong is a higher order Markov SSM. Given M which satisfies $T > M > 1$, an Mth order Markov SSM consists of an initial distribution $p(z_{1:M})$, transitions $p(z_t|z_{t-M:t-1})$ and emissions $p(x_t|z_t)$ and factorizes as:

$$p(z_{1:T}, x_{1:T}) = p(z_{1:M}) \left(\prod_{t=M+1}^{T} p(z_t|z_{t-M:t-1}) \right) \left(\prod_{t=1}^{T} p(x_t|z_t) \right). \tag{2.14}$$

Inference and prediction. Inference in a SSM results in a posterior $p(z_{1:T}|x_{1:T})$. State predictions to the next L time-steps can be made through the transition:

$$\begin{aligned} p(z_{T+1:T+L}|x_{1:T}) &= \int p(z_{1:T}|x_{1:T}) p(z_{T+1:T+L}|z_{1:T}) \, \mathrm{d}z_{1:T} \\ &= \int p(z_{1:T}|x_{1:T}) \prod_{t=T+1}^{T+L} p(z_t|z_{1:t-1}, x_{1:T}) \, \mathrm{d}z_{1:T}. \end{aligned} \tag{2.15}$$

In some applications, we are interested in data prediction, in which case we extend the state prediction through the emission:

$$\begin{aligned} p(x_{T+1:T+L}|x_{1:T}) &= \int p(z_{T+1:T+L}|x_{1:T}) p(x_{T+1:T+L}|z_{T+1:T+L}) \, \mathrm{d}z_{T+1:T+L} \\ &= \int p(z_{T+1:T+L}|x_{1:T}) \prod_{t=T+1}^{T+L} p(x_t|z_{1:t}, x_{1:T}) \, \mathrm{d}z_{T+1:T+L}. \end{aligned} \tag{2.16}$$

In Figure 2.3, we show inference and prediction in a LGSSM and a higher order Markov SSM. Since the latter is a more flexible family of models, inference and prediction in the learned model can be better. However, if the family of models is too flexible, the learned model can overfit. This results in bad inferences and predictions given previously unseen data.

2.1.3 Learning in Latent Variable Models

Ideally, we would place a prior on all unobservable quantities of interest. Given data, we perform Bayesian inference to obtain the posterior distribution over these

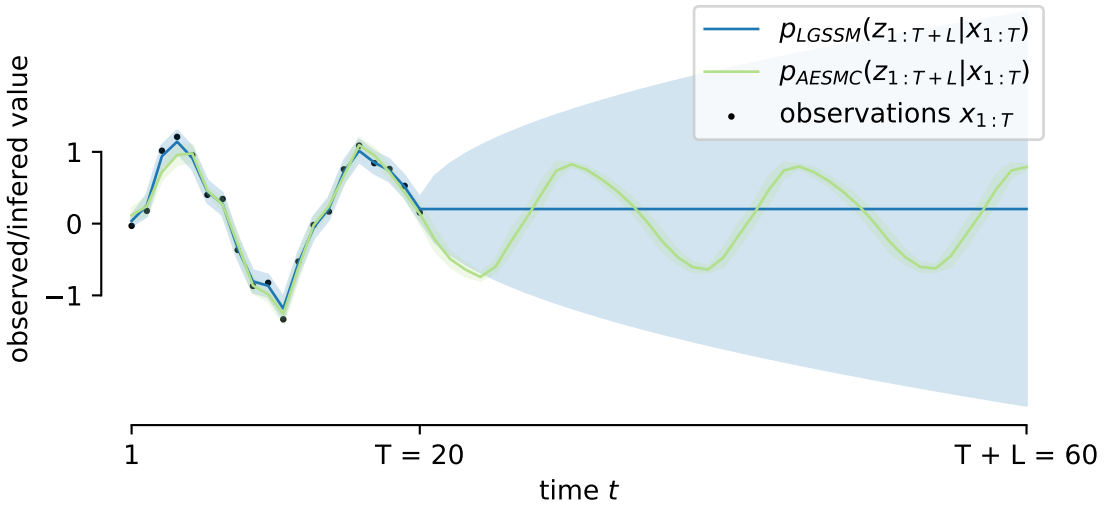


Figure 2.3: Inference and prediction for a synthetic sine-wave time-series in two models. The first model, which we denote p_{LGSSM}, is a linear Gaussian state space model whose parameters are learned using the EM algorithm. Inference and prediction are performed exactly. The second model, which we denote p_{AESMC}, is an Mth order Markov state space model ($M = 5$) where the parameter for $p(z_t|z_{t-M:t-1})$ is the output of a neural network whose input is $z_{t-M:t-1}$. The neural network weights are learned using the auto-encoding sequential Monte Carlo algorithm (Le et al., 2018a). Inference and prediction are performed approximately using the sequential Monte Carlo algorithm. We show the mean plus minus one standard deviation of $p(z_{1:T+L}|x_{1:T})$.

quantities. The posterior distribution is all we need to know if we want to predict or make a decision using our generative model.

In practice, we cannot place a prior on everything. We can place a prior on a parameter, but that prior itself has parameters that we could place a prior on, and so on. At some point, the prior distribution no longer reflects our subjective belief about the parameters. The unobservable quantity could be the model itself. We could place a prior on plausible models and perform inference in the space of models. Do we place a prior on this prior?

Instead of placing a prior on all unobservable quantities, we resort to *learning* a subset thereof. Learning refers to an optimization process that results in a fixed set of parameters. Given data, we perform Bayesian inference on the latent variables while having the parameters fixed. For example, we could want to learn the means and covariances in a GMM, the number of states in a hidden Markov model or the number of clusters in a GMM.

In this thesis, we will view learning as maximizing the marginal likelihood. Let θ denote the set of parameters to be learned. Both the prior distribution $p(z|\theta)$ and the likelihood $p(x|z,\theta)$ are conditioned upon θ. Notationally, we will write the parameters to be learned in the subscript of p. Therefore, we refer to learning as:

$$\theta^* = \operatorname*{argmax}_\theta p_\theta(x), \tag{2.17}$$

where $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)\,\mathrm{d}z$ is the marginal likelihood or evidence.

Expectation maximization (EM)—which has been previously used to learn parameters of the GMM in Section 2.1.2.2 and the SSM in Section 2.1.2.3—iteratively updates a parameter θ_i and an auxiliary probability distribution q_i over the latent variable space $\mathcal{Z}$ in, what are known, the expectation (E) and the maximization (M) steps as shown in Algorithm 1. If these steps can be done exactly, this

Algorithm 1 EM algorithm for maximum marginal likelihood.

1: Initialize θ_0
2: **for** iteration $i = 1, 2, 3, \ldots$ until convergence **do**
3: E step: $q_i(z) \leftarrow p_{\theta_{i-1}}(z|x)$
4: M step: $\theta_i \leftarrow \operatorname{argmax}_\theta \mathbb{E}_{q_i(z)}[\log p_\theta(z,x)]$
return the final θ_i.

procedure is guaranteed to always increase the log-marginal likelihood at each iteration, $p_{\theta_{i+1}}(x) \geq p_{\theta_i}(x)$. For a justification of learning or selecting models through maximization of marginal likelihood based on the Occam's razor, see e.g. (MacKay, 2003, Chapter 28).

2.2 Probabilistic Programming Languages

In the previous section, we have introduced Bayesian inference, prediction and learning in generative models with examples in clustering and time series analysis. In these examples, learning and inference was done by painstakingly deriving and implementing update equations. This can be error-prone and slow. Probabilistic programming lets modelers write models as programs while inference is automated using a general-purpose inference back-end. Modeling and inference are thus decoupled, making it possible for modelers to focus on modeling and inference

designers to focus on inference. Van de Meent et al. (2018) show that a restricted class of probabilistic programming languages which we will refer to as first-order probabilistic programming languages (FOPPLs) can express the same models as graphical models with finite number of variables. On the other hand, a higher-order probabilistic programming language (HOPPL) can express a much larger class of models since it is created by taking a Turing-complete programming language and extending it with a construct for defining random variables, which we call `sample`, and a construct for conditioning, which we call `observe`. Our interest is in HOPPLs and their ability to easily express generative models that are cumbersome (though not impossible) to express as graphical models. Such models typically have one or more of the following characteristics.

Simulator-based models. Generative models which contain a simulator are readily expressed as probabilistic programs since simulators are typically written using a programming language. Physical core knowledge may be encoded as priors in a probabilistic program containing a physics engine. Inference in such a model corresponds to inverting this engine to obtain plausible physical properties of the objects like mass and velocity using which we can simulate the program forward to predict the state of the world a ew seconds into the future. For instance, looking at wooden blocks stacked on a table, we'd like to predict whether they will fall if subject to a slight push to the table (Battaglia et al., 2013). The vision-as-inverse-graphics paradigm is also readily implemented as inference in a probabilistic program. This program places a prior on inputs to a graphics renderer and inference returns a distribution of inputs which satisfy the constraint of matching the renderer output to the observed image. We took this approach to solve Captchas in the absence of labeled data (Le et al., 2017b; Mansinghka et al., 2013). Simulators are also widely used in engineering. For instance, given a simulator of how radiators spread heat around a house, we might want to find radiator temperatures that satisfy cost constraints as well as make the house occupants comfortable (Rainforth et al., 2016a). In particle physics, Baydin et al. (2018) have used a large-scale particle

interaction simulator together with a particle detection simulator in a probabilistic program which defines a forward generative model of particle interactions and the observed particle detections. Inference in this probabilistic program results in a distribution over particle properties and interactions that could plausibly have generated the observed particle detections.

Dynamic number of variables. In modeling data with groups, we seldom know how many there are. Placing a prior on the number of groups leads to a dynamic number of group-specific variables, which is easily handled using a loop with a stochastic end condition. For instance, it is difficult to know how many objects there are in a scene and it is unrealistic to fix this number ahead of time. Moreover, objects often appear and disappear from a scene, requiring the model to maintain a dynamically changing number of variables (Neiswanger et al., 2014). Salakhutdinov et al. (2012) place a prior on the number of object categories, which allows us to learn about new categories from just one instance of an object from it. Even in the aforementioned Captcha generative model, we must place a prior on the number of letters since it is unlikely to be fixed. From a statistical point of view, Bayesian nonparametrics (e.g. Dirichlet process) is one of the leading approaches to modeling data with uncertain number of groups with applications in many other domains like information retrieval and text modeling (Teh et al., 2005).

Models with structured latent variables. Probabilistic models with structured latent variables are often used in Bayesian cognitive science to model human concept learning (Tenenbaum et al., 2011). These latent variables are data structures like trees, grids, directed graphs and even program text which are easily expressed using a programming language. For example, learning different kinds of hand-written characters can be viewed as inference in a hierarchical Bayesian model which places a prior on simple drawing programs consisting of curves, lines and their relations (Lake et al., 2015). Perov and Wood (2016) have written a probabilistic program which places a prior over probabilistic programs in order to automatically infer interpretable samplers. In natural language processing, inference in probabilistic

context free grammars is one of the key ideas in parsing syntactically ambiguous sentences (Manning et al., 1999).

Nested models. Intuitive psychology may be formalized as Bayesian inference of agent's goals given the observed effect of its actions on the environment (Baker et al., 2009; Evans et al., 2017). Similarly to vision-as-inverse-graphics, this approach to intuitive psychology can be understood as inverting an approximately rational planner. This model consists of a subroutine which takes a goal as input, and potentially the agent's observations, and outputs an appropriate action. If the agent only partially observes the environment, this subroutine may advantageously perform inference as an intermediate step prior to choosing an action. It is easier to reason about inference in models which consist of such subroutines (which may themselves perform inference) in a probabilistic programming framework than a probabilistic graphical model (Rainforth, 2018).

We first introduce HOPPLs through examples written in the Anglican probabilistic programming language (PPL) (Section 2.2.1). We then explain in more detail differences between FOPPLs and HOPPLs (Section 2.2.2). Then, we explain the concepts of addressing (Section 2.2.3) and execution traces (Section 2.2.4). Lastly, we provide a notion of joint and posterior distributions in a HOPPL (Section 2.2.5). For an in-depth tutorial treatment of the subject, see e.g. Goodman and Stuhlmüller (2014) or van de Meent et al. (2018).

2.2.1 Anglican Probabilistic Programming Language

There are many probabilistic programming languages each targeting a different class of models and coming from a different community[1]:

- programming languages: Hakaru (Narayanan et al., 2016), Augur (Tristan et al., 2014), R2 (Nori et al., 2014), Figaro (Pfeffer, 2009), IBAL (Pfeffer, 2001), PSI (Gehr et al., 2016), monad-bayes (Ścibior et al., 2015),

[1] this list is adapted from van de Meent et al. (2018, Section 1.2.1)

- machine learning: Church (Goodman et al., 2008), Anglican (Wood et al., 2014), BLOG (Milch et al., 2004), Turing.jl (Ge et al., 2018), BayesDB (Mansinghka et al., 2015), Venture (Mansinghka et al., 2014), Probabilistic C (Paige and Wood, 2014), webPPL (Goodman and Stuhlmüller, 2014), CPProb (Casado, 2017),

- statistics: Biips (Todeschini et al., 2014), LibBi (Murray, 2013), Birch (Murray and Schön, 2018), STAN (Carpenter et al., 2017), JAGS (Plummer et al., 2003), BUGS (Lunn et al., 2000), Infer.NET (Minka et al., 2018), Factorie (McCallum et al., 2009),

- deep generative modeling: Pyro (Bingham et al., 2018), Edward (Tran et al., 2016), Probtorch (Siddharth et al., 2017), pyprob (Le et al., 2017a), and

- probabilistic logic: ProbLog (Kimmig et al., 2011), PRISM (Sato and Kameya, 1997).

Here, we illustrate principles of HOPPLs using probabilistic programs written in the Anglican PPL (Wood et al., 2014). Anglican is a PPL built on top of a subset of the Clojure programming language (Hickey, 2008)[2]. In general, a HOPPL extends a Turing-complete programming language with `sample` and `observe` statements (Gordon et al., 2014). Both `sample` and `observe` are functions that specify random variables in a generative model using probability distribution objects as an argument. The `observe` statement, in addition, specifies the conditioning of a random variable upon an observed value. These observed values induce a posterior probability distribution over execution traces.

We present three Anglican programs in order to illustrate different aspects of PPLs. First, we present a simple Anglican program to illustrate the usage of `sample` and `observe` statements. Second, we present an Anglican program denoting a open-universe GMM. This example illustrates how one can write generative models with potentially unbounded number of variables in Anglican. Lastly, we present an

[2]The syntax and language reference can be found on `http://probprog.ml/anglican`.

Anglican program which denotes a stochastic Captcha renderer. This illustrates how we can use the full machinery of a programming language—in this case a rendering library—to define generative models.

A simple Anglican program. Figure 2.4 shows an Anglican program that denotes the Gaussian unknown mean model in Section 2.1. An Anglican program is defined as a `defquery` statement. This program has one argument, `x`. The `let` statement first binds `(sample (normal 0 1))` to the variable `z`. This denotes that `z` is drawn from Normal$(0, 1^2)$. The next statement in the `let` block, `(observe (normal z 1) x)`, serves two purposes. First, it defines the conditional distribution of the observed value given in-scope variables, in this case $p(x|z) =$ Normal$(x|z, 1^2)$. Second, it provides an actual observed value of x, here denoted as the variable `x`. The last statement of the `let` block, `z`, simply defines its return value. The return value denotes variables we are interested in inferring. In this case, we are interested in the distribution of `z` given that we observe `x` under the Gaussian unknown mean model.

```
1 (defquery [x]
2   (let [z (sample (normal 0 1))]
3     (observe (normal z 1) x)
4     z))
```

Figure 2.4: Gaussian unknown mean (see Section 2.1.2.1) model in Anglican.

Open-Universe Gaussian Mixture Model. Next, we consider an extension of the GMM (see Section 2.1.2.2) where the number of clusters is potentially unbounded. Being able to define generative models with potentially unbounded number of variables is one of the features distinguishing HOPPLs from FOPPLs. In Figure 2.5, we show an Anglican program that denotes a GMM where the prior distribution on number of clusters has an infinite support. The `let` binding in line 2 denotes that the number of clusters K is distributed according to $(1 + \text{Poisson}(\lambda = 3))$ whose support is all positive integers. The cluster probabilities π, whose prior is defined in

lines 3–4, are distributed according to a Dirichlet distribution whose dimensionality is dependent on K. The priors for cluster means $\mu_{1:K}$ and variances $\sigma^2_{1:K}$ are defined in lines 5–8. The number of these per-cluster parameters is dependent on K can take values in $\{1, 2, \dots\}$ (although it is finite with probability one). The rest of the probabilistic program consists of a `loop` statement which loops through all data points $x_{1:N}$ represented by the variable `data`. For each data point x_n, we sample its cluster identity z_n in line 13. The `observe` statement in lines 14–16, defines a likelihood $\text{Normal}(x_n|\mu_{z_n}, \sigma^2_{z_n})$ and fixes the observed value of x_n (here through `(first data)`). The return value of the `loop` statement and the whole probabilistic program—defined in line 12—are the cluster probabilities π, means $\mu_{1:K}$, variances $\sigma^2_{1:K}$ and cluster identities $z_{1:N}$. This probabilistic program denotes that we are interested in a posterior distribution $p(K, \pi, \mu_{1:K}, \sigma^2_{1:K}, z_{1:N}|x_{1:N})$. Its support doesn't have a fixed dimensionality.

```
1  (defquery gmm [data]
2    (let [num-clusters (inc (sample (poisson 3)))
3          cluster-probs (sample
4            (dirichlet (repeat num-clusters 1)))
5          means (repeatedly num-clusters
6                            #(sample (normal 0 1)))
7          vars (repeatedly num-clusters
8                           #(sample (gamma 1 1)))]
9      (loop [data data
10            clusters []]
11       (if (empty? data)
12         [cluster-probs means vars clusters]
13         (let [cluster (sample (discrete cluster-probs))]
14           (observe (normal (nth means cluster)
15                            (nth vars cluster))
16                    (first data))
17           (recur (rest data) (conj clusters cluster)))))))
```

Figure 2.5: Open-Universe Gaussian mixture model in Anglican. The prior distribution on the number of clusters, in line 2, has infinite support. Hence, the number of cluster probabilities, means and variances (lines 2–8) is unbounded. This distinguishes HOPPLs from FOPPLs.

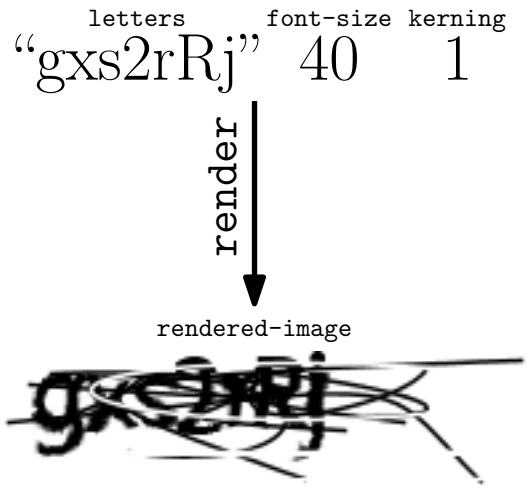


Figure 2.6: The Captcha renderer takes as inputs values such as the font size, kerning and the letters and outputs a Captcha image. In the *vision as inverse graphics* paradigm, we can solve Captchas by inverting this renderer. Probabilistic programming allows us to write the rendering process as a generative model in which inference corresponds to computing posterior distributions over the renderer inputs.

Inverting a Captcha renderer. Finally, we consider a generative model of a stochastic Captcha renderer. Given an actual Captcha image, posterior inference in this program “inverts” the rendering process. The posterior represents a distribution over inputs to the renderer that could have given rise to the given Captcha image. This is an instance of *vision as inverse graphics*—a paradigm that poses the problem of computer vision as the inverse of graphics rendering. In order to be able to formulate this as probabilistic inference, PPLs can use graphics rendering libraries as part of the probabilistic programs. In Figure 2.7, we show an Anglican program that “inverts” a Captcha renderer. Lines 2–4 define a prior distribution on the number of letters, font size and kerning. Kerning refers to the distance between two rendered letters. Line 5 defines a prior distribution on letter identities which are used in line 6 to select the actual letters from a pre-specified alphabet. Line 10 uses the `render` function, which comes from a Captcha rendering library (see Fig. 2.6). It takes the letters, font size and kerning as input and outputs a rendered Captcha image. The `observe` statement in lines 11–12 acts as an approximate Bayesian computation (ABC) (Sisson et al., 2018) distance. The “likelihood” defined by this `observe` statement is high when the `rendered-image` is similar to `observed-image`

and low otherwise. The distance between images are implemented as a scaled L_2 distance between the vectors obtained through flattening the image matrices although other choices, like distance in the feature space, are also possible.

```
1  (defquery captcha [observed-image]
2    (let [num-letters (sample (uniform-discrete 6 8))
3          font-size (sample (uniform-discrete 38 44))
4          kerning (sample (uniform-discrete -2 2))
5          letter-ids (repeatedly
6            num-letters
7            #(sample (uniform-discrete 0 (count letter-dict))))
8          letters (apply str (map (partial nth letter-dict)
9                                  letter-ids))
10         rendered-image (render letters font-size kerning)]
11     (observe (abc-dist rendered-image abc-sigma)
12              observed-image)
13     letters))
```

Figure 2.7: Inverting a Captcha renderer in Anglican. PPLs let us define generative models using the full machinery of a programming language. In this case, we use the `render` function in line 10 that comes from a Captcha rendering library.

2.2.2 Differences to First Order Languages

FOPPLs can be shown to be equivalent to finite graphical models (van de Meent et al., 2018). Van de Meent et al. (2018) formalize a notion of HOPPL that extends FOPPLs in two ways. First, functions can be recursive. Second, functions can accept other functions as arguments. These additional features let HOPPLs denote generative models which don't have fixed number of random variables. Executing these programs by sampling from the prior results in an arbitrary number of calls to `sample` and `observe` statements that can be different from one execution to another. This can arise in at least two ways. First, using recursion, where the recursive condition is a function of a random variable, can result in variable number of recursive calls each of which can have their own `sample` statements. This allows writing models such as probabilistic context-free grammars and stick-breaking non-parametric models which have infinite number of random variables (though for inference, only a finite number of them need to be evaluated). Second, the number

of iterations in a loop can be sampled from a distribution with an infinite support (like Poisson). If each loop iteration has a `sample` statement, the number of them is not fixed and unbounded. The open-universe GMM is an example of this.

From a practical perspective, a HOPPL lets users write their models in an unrestricted manner since they can use the full machinery of the underlying programming language, including libraries, as has been the case in the Captcha example.

2.2.3 Addressing

Consider, again, the open-universe GMM program in Figure 2.5. Generating from the prior defined by this program (without conditioning on observed values) results in a variable number of per-cluster parameters, depending on the number of clusters K which is sampled from a Poisson distribution. In particular, the sequence of sampled values is $(K, \mu_1, \ldots, \mu_K, \Sigma_1, \ldots, \Sigma_K, z_1, x_1, \ldots, z_N, x_N)$ whose length, $1 + 2K + 2N$, depends on the value of K.

Many sampling-based inference algorithms used for probabilistic programming require uniquely identifying the elements of a sequence of samples. In Markov chain Monte Carlo (MCMC)-based algorithms such as single-site MCMC (Wingate et al., 2011) and particle MCMC (Andrieu et al., 2010; Wood et al., 2014), one needs to implement a transition kernel which is a conditional distribution of one sample sequence given another sample sequence, and evaluate the acceptance probability of accepting a new sample sequence given by the transition kernel. This requires a unique identifier for each sample statement in order to correctly align elements from the old and the new sample sequences. In particle-based methods such as importance sampling (IS) and sequential Monte Carlo (SMC) that don't propose from the prior also need a unique identifier for each sample statement in order to align the proposal distribution with the prior distribution so that we can evaluate the weights.

Wingate et al. (2011) propose a *dynamic* addressing scheme which transforms probabilistic program to an equivalent program which additionally generates an address during runtime for each sample statement based on the current call-stack.

In this thesis, we use a *static* addressing scheme which assigns an address to each sample statement during compilation according to its lexical position. Each sampled value is then uniquely identified using its static address together with the instance number. The instance number refers to the number of times the same sample statement (including the current one) has been encountered so far. In the case of open-universe GMM program, assume that the sample statements have been assigned static addresses `"a1"` through `"a5"` as indicated in Table 2.1. This can be either done automatically during compilation, or manually by the user. The sample instances of sampled values for K and π are both 1 since we encounter these values only once when running the program forward. For $\mu_{1:K}$ and $\Sigma_{1:K}$, the sample instances increase from 1 to $K = 3$. For example, the address of the second sampled value obtained from `(sample (normal 0 1))` (in lines 5–6) is $(\texttt{"a3"}, 2)$.

We choose the *static* addressing scheme due to the ease of implementation and the ability to uniquely identify sample statements. This addressing scheme requires the user to manually annotate sample statements (e.g. with `"a1"` through `"a5"`) and has no information about the program stack. However, in our applications, this information is not used.

2.2.4 Execution Trace

We formalize the concept of a sample sequence of a probabilistic program through execution traces. An execution trace of a probabilistic program is obtained by successively executing the program deterministically, except when encountering `sample` statements at which point a value is generated according to the specified probability distribution and appended to the execution trace. We denote the sequence of observed values encountered during execution by $x := (x_n)_{n=1}^{N}$. In general, the order and the number of `observe` statements that are encountered can be dynamic and different from one execution to another. However, some inference algorithms like SMC require x to be fixed.

Depending on the probabilistic program and the values generated at `sample` statements, the order in which the execution encounters `sample` statements as well

as the number of encountered `sample` statements may be different from one trace to another. Therefore, given an addressing scheme which assigns a unique static address to each `sample` statement according to its lexical position in the probabilistic program, we represent an execution trace of a probabilistic program as a sequence

$$(z_t, a_t, i_t)_{t=1}^{T} \ , \tag{2.18}$$

where z_t, a_t, and i_t are respectively the sample value, address, and instance (call number) of the tth entry in a given trace, and T is a execution-dependent length. Instance values $i_t = \sum_{j=1}^{t} \mathbb{1}(a_t = a_j)$ count the number of sample values obtained from the specific `sample` statement at address a_t, up to time step t. For each trace, a sequence $z := (z_t)_{t=1}^{T}$ holds the T sampled values from the `sample` statements.

Table 2.1 shows an example of execution trace by running the open-universe GMM program in Figure 2.5 forward, and sampling from the prior distribution when encountering `sample` statements.

Table 2.1: An execution trace obtained by running the open-universe GMM program in Figure 2.5.

Sample statement	Line number(s)	Sampled value z_t	Address	
			Static address a_t	Sample instance i_t
`(sample (poisson 3))`	2	2	`"a1"`	1
`(sample (dirichlet (repeat num-clusters 1)))`	3–4	$[0.3120, 0.0685, 0.6196]$	`"a2"`	1
`(sample (normal 0 1))`	5–6	1.0337	`"a3"`	1
		-0.7997		2
		0.4271		3
`(sample (gamma 1 1))`	7–8	0.9887	`"a4"`	1
		0.8729		2
		0.7617		3
`(sample (discrete cluster-probs))`	13	2	`"a5"`	1
		0		2
		2		3
		1		4
		2		5

2.2.5 Joint and Posterior Distributions

The joint probability density of an execution trace is

$$p(z,x) := \prod_{t=1}^{T} f_{a_t}\left(z_t|z_{1:t-1}\right) \prod_{n=1}^{N} g_n(x_n|z_{1:\tau(n)}), \tag{2.19}$$

where f_{a_t} is the probability density specified by the `sample` statement at address a_t and g_n is the probability distribution specified by the nth `observe` statement. $f_{a_t}(\cdot|z_{1:t-1})$ is called the prior conditional density given the sample values $z_{1:t-1}$ obtained before encountering the tth `sample` statement. $g_n(\cdot|z_{1:\tau(n)})$ is called the likelihood density given the sample values $z_{1:\tau(n)}$ obtained before encountering the nth `observe` statement, where τ is a mapping from the index n of the `observe` statement to the index of the last `sample` statement encountered before this `observe` statement during the execution of the program. Inference in such models amounts to computing an approximation of $p(z|x)$ and its expected values $\mathbb{E}_{p(z|x)}[f(z)]$ of chosen functions f.

2.3 Inference Algorithms

We have discussed Bayesian machine learning (Section 2.1) as a coherent conceptual framework for unsupervised learning and probabilistic programming languages (Section 2.2) as a way to define probabilistic models as programs. Now, we present inference algorithms that are suitable for HOPPLs in that they only depend on being able to evaluate the probabilistic program forward. We introduce importance sampling (IS) (Section 2.3.1), sequential Monte Carlo (SMC) (Section 2.3.2) and variational inference (VI) (Section 2.3.3). These so-called evaluation-based inference algorithms for HOPPLs are foundational for amortized inference and model learning algorithms presented later in the thesis.

We omit algorithms from the MCMC family (Wingate et al., 2011; Ritchie et al., 2016a; Le, 2015) and more advanced ones from the particle MCMC family (Andrieu et al., 2010; Paige et al., 2014; Rainforth et al., 2016b) which are also suitable for HOPPL, however not directly needed for algorithms presented later in this chapter.

In the following, we omit derivations which can instead be found in Appendix A.

2.3.1 Importance Sampling

Given a function $f : \mathcal{Z} \to \mathbb{R}$ and a probability distribution on $\mathcal{Z}$ with the density π, importance sampling (IS) aims to provide a low-variance estimator of the following integral:

$$I := \mathbb{E}_{\pi(z)}[f(z)] := \int f(z)\pi(z)\,\mathrm{d}z. \tag{2.20}$$

In Bayesian inference, we are usually interested in evaluating $\mathbb{E}_{p(z|x)}[f(z)]$ given data x and some query function f, in which case $\pi(z) = p(z|x)$.

Intuitively, IS aims to reduce the variance of the estimator for I by sampling a proposal distribution different than π which puts mass on regions where the integrand is high and later accounting for this change in the sampling distribution by a multiplicative factor.

We first introduce Monte Carlo (MC) estimation which forms the foundation of IS and other sampling-based algorithms. We then introduce basic IS where we assume the density $\pi : \mathcal{Z} \to \mathbb{R}$ can be evaluated pointwise. Next, we relax this assumption and introduce self-normalized IS, where π can only be evaluated up to a normalizing constant. Finally, we describe how IS can be used for performing inference in HOPPLs.

IS is also known as likelihood weighting. It has first been proposed in (Kahn, 1950a,b). For a good textbook treatment, see e.g. Owen (2013).

2.3.1.1 Monte Carlo Estimation

Given K independently and identically distributed (IID) samples $z_k \sim \pi, k = 1, \ldots, K$, the MC estimator of I is given by the following empirical average:

$$I_K^{\mathrm{MC}} = \frac{1}{K}\sum_{k=1}^{K} f(z_k). \tag{2.21}$$

It can be shown via linearity of expectations that I_K^{MC} is an unbiased estimator of I, i.e. $\mathbb{E}[I_K^{\mathrm{MC}}] = I$. The variance can be expressed as

$$\mathrm{Var}\left[I_K^{\mathrm{MC}}\right] = \frac{1}{K}\mathrm{Var}[f(z)] = \frac{1}{K}\left(\int \pi(z)f(z)^2\,\mathrm{d}z - I^2\right). \tag{2.22}$$

This is due to additivity of variance for independent random variables and the fact that given a constant $a \in \mathbb{R}$ and a random variable z, $\mathrm{Var}[az] = a^2\,\mathrm{Var}[z]$.

2.3.1.2 Basic Importance Sampling

Assume there exists a proposal distribution on $\mathcal{Z}$ which we can sample from and whose density q can be evaluated pointwise. Additionally, we require $\int_A f(z)\pi(z)\,\mathrm{d}z = 0$ whenever $\int_A q(z)\,\mathrm{d}z = 0$ for all measurable sets A. Then the basic IS estimator of I is given by a weighted average:

$$I_K^{\mathrm{IS}} = \frac{1}{K}\sum_{k=1}^{K} w_k f(z_k), \tag{2.23}$$

where $z_k \sim q$ and $w_k = \pi(z_k)/q(z_k)$. The samples $z_{1:K}$ are often called *particles*. The ratio w_k is called an *importance weight* (or just weight).

It can be shown that I_K^{IS} is an unbiased estimator I, i.e. $\mathbb{E}\left[I_K^{\mathrm{IS}}\right] = I$ and the variance can expressed as (see Appendix A.1.1):

$$\mathrm{Var}\left[I_K^{\mathrm{IS}}\right] = \frac{1}{K}\left(\int \frac{\pi(z)^2 f(z)^2}{q(z)}\,\mathrm{d}z - I^2\right). \tag{2.24}$$

This means that increasing number of particles K decreases the variance, provided it is finite. This might not be the case. Intuitively, this happens when q is small in large areas where πf is not small. Owen (2013, Example 9.1) provides the following example where the variance $\mathrm{Var}\left[I_K^{\mathrm{IS}}\right]$ is infinite. Consider $\mathcal{Z} = \mathbb{R}$ where $\pi(z) = \mathrm{Normal}(z|0, 1^2)$, $q(z) = \mathrm{Normal}(z|0, \sigma^2)$ and $f(z) = z$. The variance of the IS estimator is only finite when $\sigma > 1/2$. *In practice, if we don't know f, q should be designed to have heavier tails than π.*

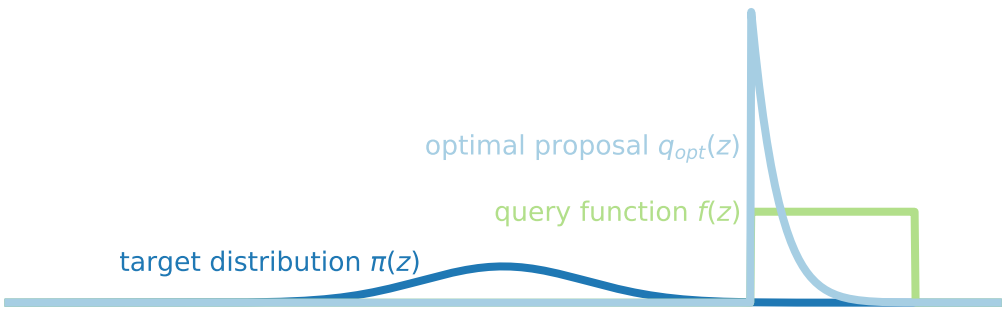


Figure 2.8: The optimal proposal for estimating $I = \int \pi(z)f(z)\,\mathrm{d}z$ using basic IS has the form $q_{\mathrm{opt}}(z) \propto |f(z)|\pi(z)$. This proposal minimizes the variance of the estimator I_K^{IS} given in (2.23). Intuitively, samples from q_{opt} come from an "important" region of $\mathcal{Z}$.

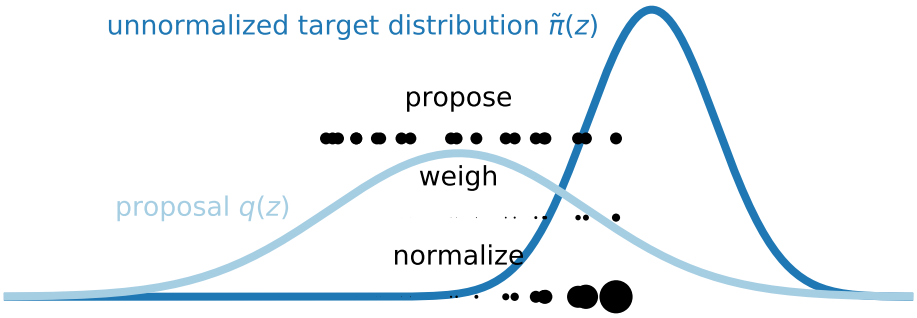


Figure 2.9: Illustration of self-normalized IS. There are three steps: 1) propose $z_k \sim q$, 2) weigh $w_k = \tilde{\pi}(z_k)/q(z_k)$ and 3) normalize $\bar{w}_k = w_k / \sum_{\ell=1}^{K} w_\ell$. The samples $z_{1:K}$ are also called *particles*. The target distribution π is approximated using weighted delta masses $\pi(z) \approx \sum_{k=1}^{K} \bar{w}_k \delta_{z_k}(z)$. The integral $I = \int \pi(z) f(z) \, \mathrm{d}z$ is estimated as the weighted sum $\sum_{k=1}^{K} \bar{w}_k f(z_k)$. The horizontal position of the particles represents their values $z_{1:K}$. Their areas represent the weights $w_{1:K}$.

One notion of an optimal proposal for IS is minimization of $\mathrm{Var}\,[I_K^{\mathrm{IS}}]$. Such proposal, denoted q_{opt} has the form (see Appendix A.1.1):

$$q_{\mathrm{opt}}(z) = \frac{|f(z)|\pi(z)}{\int |f(z')|\pi(z') \, \mathrm{d}z'}. \tag{2.25}$$

In other words, the optimal proposal q_{opt} is proportional to $\pi(z)|f(z)|$. Intuitively, a good proposal has high probability mass in a region that is "important" for estimating I. We illustrate this in Figure 2.8. However, such proposal is difficult to obtain in practice since it requires the difficult evaluation of the expectation $\mathbb{E}_\pi[f(z)]$.

2.3.1.3 Self-Normalized Importance Sampling

We wish to use IS for Bayesian posterior inference where the target distribution $\pi(z)$ is the posterior distribution $p(z|x)$. Basic IS is not suitable for this since we can seldom evaluate the posterior density pointwise. However, we can often evaluate the joint density $p(z, x)$ which is proportional to the posterior density $p(z|x)$ with the normalizing constant being the marginal likelihood $p(x)$.

Self-normalized IS relaxes the need to evaluate the target density $\pi(z)$. Instead, assume that we can (i) draw samples from a proposal distribution on $\mathcal{Z}$ with density q, (ii) evaluate density q, and (iii) evaluate the density p only up to a normalization constant, i.e. we can evaluate $\tilde{\pi}(z) = Z\pi(z)$ where $Z = \int \tilde{\pi}(z) \, \mathrm{d}z$ is

the normalization constant. We also require that $\int_A \pi(z) = 0$ whenever $\int_A q(z)\,\mathrm{d}z = 0$ for all measurable sets A (i.e. π must be absolutely continuous with respect to q). In the case of Bayesian inference, $\tilde{\pi}(z) = p(z, x)$, $\pi(z) = p(z|x)$ and $Z = p(x)$.

The self-normalized IS estimator of I is given by:

$$\tilde{I}_K^{\mathrm{IS}} = \frac{\frac{1}{K}\sum_{k=1}^K w_k f(z_k)}{\frac{1}{K}\sum_{k=1}^K w_k}, \tag{2.26}$$

where $z_k \sim q$ and $w_k = \tilde{\pi}(z_k)/q(z_k)$. The ratio w_k is often called the *unnormalized weight*. The *normalized weight* is defined as $\bar{w}_k := w_k / \sum_{\ell=1}^K w_\ell$. The estimator in (2.26) be rewritten in terms of the normalized weight as follows:

$$\tilde{I}_K^{\mathrm{IS}} = \sum_{k=1}^K \bar{w}_k f(z_k). \tag{2.27}$$

This estimator is biased: $\mathbb{E}[\tilde{I}_K^{\mathrm{IS}}] \neq I$. However, $\tilde{I}_K^{\mathrm{IS}}$ converges to I almost surely as K goes to infinity because the numerator and the denominator converge almost surely to ZI and Z respectively and so the estimator is consistent.

The target distribution is approximated as a weighted sum of delta masses:

$$\pi(z) \approx \sum_{k=1}^K \bar{w}_k \delta_{z_k}(z). \tag{2.28}$$

Self-normalized IS also gives us an unbiased estimator (see Appendix A.1.2) of the normalizing constant Z:

$$Z_K^{\mathrm{IS}} = \frac{1}{K}\sum_{k=1}^K w_k. \tag{2.29}$$

Being able to estimate Z allows us to estimate the marginal likelihood $p(x)$ of a generative model $p(z, x)$. This is useful for model learning (see Section 2.1.3).

A proposal that minimizes the asymptotic variance of $\tilde{I}_K^{\mathrm{IS}}$ has the form

$$q_{\mathrm{opt}}(z) = \frac{\pi(z)|f(z) - I|}{\int \pi(z)|f(z) - I|\,\mathrm{d}z} \text{ (Owen, 2013, Chapter 9.2).} \tag{2.30}$$

In most cases, f is not known ahead of time and we might want to evaluate I for multiple f. In this case, there is no single optimal proposal distribution in terms of estimator variance minimization. Instead, we often consider an optimal proposal to be

$$q_{\mathrm{opt}}(z) = \pi(z). \tag{2.31}$$

If we manage to achieve this, the self-normalized IS estimator becomes

$$\tilde{I}_K^{\text{IS}} = \frac{1}{K} \sum_{k=1}^{K} f(z_k), \tag{2.32}$$

where $z_k \sim q_{\text{opt}} = \pi$ and hence follows the properties of a standard MC estimator.

2.3.1.4 Importance Sampling for Probabilistic Programming

Using self-normalized IS for inference in HOPPLs requires designing proposal distributions $q_{a,i}$ corresponding to the addresses a of all `sample` statements in the probabilistic program and their instance values i. A proposal execution trace $z^k_{1:T_k}$ is built by executing the program as usual, except when a `sample` statement at address a_t is encountered at time t, a proposal sample value z^k_t is sampled from the proposal distribution $q_{a_t,i_t}(\cdot|z^k_{1:t-1})$ given the proposal sample values until that point. We obtain K proposal execution traces $z_k := z^k_{1:T_k}$ (possibly in parallel) to which we assign weights

$$w_k = \prod_{n=1}^{N} g_n(x_n|z^k_{1:\tau_k(n)}) \cdot \prod_{t=1}^{T_k} \frac{f_{a_t}(z^k_t|z^k_{1:t-1})}{q_{a_t,i_t}(z^k_t|z^k_{1:t-1})} \tag{2.33}$$

for $k = 1, \ldots, K$ with T_k denoting the length of the kth proposal execution trace.

Given a particle set $(w_k, z_k)_{k=1}^K$, we can estimate the integral in (2.20) using the self-normalized IS estimator in (2.26). Since the sequence z_k doesn't necessarily have the same number and type of elements, the domain of the query function f must be the space of possible z_k. The posterior distribution is approximated by the following empirical distribution:

$$\hat{p}(z|x) = \frac{1}{K} \sum_{k=1}^{K} \bar{w}_k \delta_{z_k}(z) \approx p(z|x), \tag{2.34}$$

where $\bar{w}_k = w_k / \sum_{\ell=1}^{K} w_\ell$ is the normalized weight and δ_{z_k} is a delta mass centered on z_k. The marginal likelihood is approximated using the following expression:

$$\hat{p}(x) = \frac{1}{K} \sum_{k=1}^{K} w_k \approx p(x). \tag{2.35}$$

2.3.2 Sequential Monte Carlo

The IS estimator suffers from the curse of dimensionality. It is essentially a guess-and-check procedure. We obtain particles by sampling from a proposal distribution and assign a weight according to how well the sample scores against the target distribution. If the proposal distribution is far away from regions of high probability mass under the target distribution[3], the variance of the estimator can explode. In high dimensions, this becomes worse due to the curse of dimensionality (e.g. (Doucet and Johansen, 2009, Section 3.6)).

We are interested in applying IS to problems of Bayesian inference where the latent variable is often high-dimensional. For instance, in a SSM, the number of latent variables increases with time T. In an SSM, each observation at time t, x_t, gives us additional information about the state of the system until now, $z_{1:t}$. The key idea in SMC is to exploit this intermediate information to direct computation into promising areas of the target distribution unlike IS which only uses the aggregate information at the end. Intuitively, instead of guessing and checking at the last time-step, SMC checks at every time-step and modifies its guess accordingly.

To introduce SMC, we abstract away from SSMs and formulate the inferential problem in terms of sequences of target distributions (Liu and Chen, 1998). This simplifies the presentation while preserving the essence of SMC, allowing us to apply SMC outside of SSMs. Next we introduce the SMC algorithm and the resulting estimators. We then illustrate the SMC algorithm on the LGSSM. Finally, we describe how SMC is used in the context of probabilistic programming.

SMC was perhaps first introduced in the signal processing context by Stewart and McCarty (1992) and Gordon et al. (1993) and is often known as "particle filtering". For a mathematically rigorous treatment of the subject, see e.g. Del Moral (2004). For a more introductory tutorial to the subject, see e.g. Doucet and Johansen (2009) or Doucet et al. (2001). The use of SMC, together with other, more advanced, particle methods in probabilistic programming has been proposed by Wood et al. (2014).

[3]See Section 2.3.1 for a more nuanced discussion.

2.3.2.1 State Space Models as Sequences of Target Distributions

We introduce SMC in terms of sequences of target distributions which are defined as follows. Let $(z_1, z_{1:2}, \ldots, z_{1:T})$ be a sequence of variables of increasing dimensions. Let $\tilde{\pi}_t(z_{1:t})$ be a sequence of *unnormalized target densities* and $\pi_t(z_{1:t}) = \tilde{\pi}_t(z_{1:t})/Z_t$ be a sequence of the corresponding *normalized densities*, where the *normalizing constants* are defined as $Z_t = \int \tilde{\pi}_t(z_{1:t}) \, \mathrm{d}z_{1:t}$. The goal is to

1. approximate the normalized distributions π_t,
2. estimate integrals of the form $I = \int f(z_{1:t})\pi_t(z_{1:t}) \, \mathrm{d}z_{1:t}$, and
3. estimate the normalizing constants Z_t.

This formulation of inference is general and encompasses SSMs as follows.

Recall that a joint density of a general SSM is given by:

$$p(z_{1:T}, x_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t|z_{1:t-1}, x_{1:t-1}) \prod_{t=1}^{T} p(x_t|z_{1:t}, x_{1:t-1}), \tag{2.36}$$

where $z_{1:T}$ is a sequence of latent variables and $x_{1:T}$ is a sequence of observed data points. We are often interested in the smoothing distributions $p(z_{1:t}|x_{1:T})$ and the filtering distributions $p(z_{1:t}|x_{1:t})$. A common choice is to set

$$\tilde{\pi}_t(z_{1:t}) := p(z_{1:t}, x_{1:t}) = p(z_1) \prod_{\tau=2}^{t} p(z_\tau|z_{1:\tau-1}, x_{1:\tau-1}) \prod_{\tau=1}^{t} p(x_\tau|z_{1:\tau}, x_{1:\tau-1}), \tag{2.37}$$

which implies

$$Z_t = \int \tilde{\pi}_t(z_{1:t}) \, \mathrm{d}z_{1:t} = p(x_{1:t}), \tag{2.38}$$

$$\pi_t(z_{1:t}) = \tilde{\pi}_t(z_{1:t})/Z_t = p(z_{1:t}|x_{1:t}). \tag{2.39}$$

The target distributions π_t directly correspond to the filtering distributions. The smoothing distributions can be obtained as a marginal distribution of π_T:

$$p(z_{1:t}|x_{1:T}) = \int \pi_T(z_{1:T}) \, \mathrm{d}z_{t+1:T}. \tag{2.40}$$

Given a sample $z_{1:T}$ from $\pi_T(z_{1:T})$, a sample from $p(z_{1:t}|x_{1:T})$ can be obtained by simply taking the first t elements of $z_{1:T}$ and ignoring the rest. Lastly, the normalizing constant Z_T corresponds to the marginal likelihood $p(x_{1:T})$ which can be used for model learning or model selection. Hence, we can pose both inference and learning tasks in SSMs in terms of sequences of target distributions.

2.3.2.2 Sequential Monte Carlo Estimators

Assume that at every time-step t, there exists a *proposal distribution* $q_t(z_t|z_{1:t-1})$, where for a given $z_{1:t-1}$ we can sample z_t and evaluate its density. For $t = 1$, we assume that there exists $q_1(z_1)$ from which we can sample z_1 and evaluate its density. The SMC algorithm is shown in Algorithm 2.

In the first time-step (lines 1–3), we proceed in the same way as self-normalized IS and sample z_1^k from q_1 and evaluate the weights $w_1^k \sim \tilde{\pi}_1(z_1^k)/q_1(z_1^k)$. Our kth *particle set*, denoted $\bar{z}_1^k$, at this point consists of only z_1^k. At every subsequent step ($t = 2, 3, \ldots, T$), we first perform a resampling step, where we sample an *ancestral index* a_{t-1}^k from a Categorical distribution whose probabilities are given by the normalized weights at the previous time-step $\bar{w}_{t-1}^{1:K}$ (line 5). The ancestral index is used to select which particle set at the previous time $\bar{z}_{1:t-1}^k$ is used for sampling the current particle value: $z_t^k \sim q_t(\cdot|\bar{z}_{1:t-1}^{a_{t-1}^k})$ (line 6). The resampling procedure "kills off" particles that have a low weight $\bar{w}_{t-1}^k$, thus focusing computation on only high-weight particles. The newly sampled particle z_t^k is appended to $\bar{z}_{t-1}^{a_{t-1}^k}$ to form the *kth particle set at time-step t*, denoted $\bar{z}_{1:t}^k$ (line 7). We then compute the new weights based on the new particle set (line 8).

Algorithm 2 Sequential Monte Carlo for a sequence of target distributions.

1: Sample initial particle values $z_1^k \sim q_1$.
2: Initialize particle set: $\bar{z}_1^k \leftarrow z_1^k$
3: Compute and normalize weights:

$$w_1^k = \frac{\tilde{\pi}_1(z_1^k)}{q_1(z_1^k)}, \qquad \bar{w}_1^k = \frac{w_1^k}{\sum_{\ell=1}^K w_1^\ell}. \tag{2.41}$$

4: **for** $t = 2, 3, \ldots, T$ **do**
5: Sample ancestral index $a_{t-1}^k \sim \text{Categorical}(\bar{w}_{t-1}^{1:K})$.
6: Sample particle value $z_t^k \sim q_t(\cdot|\bar{z}_{1:t-1}^{a_{t-1}^k})$.
7: Update particle set $\bar{z}_{1:t}^k \leftarrow (\bar{z}_{1:t-1}^{a_{t-1}^k}, z_t^k)$.
8: Compute and normalize weights:

$$w_t^k = \frac{\tilde{\pi}_t(\bar{z}_{1:t}^k)}{\tilde{\pi}_{t-1}(\bar{z}_{1:t-1}^{a_{t-1}^k}) q_t(z_t^k|\bar{z}_{1:t-1}^{a_{t-1}^k})}, \qquad \bar{w}_t^k = \frac{w_t^k}{\sum_{\ell=1}^K w_t^\ell}. \tag{2.42}$$

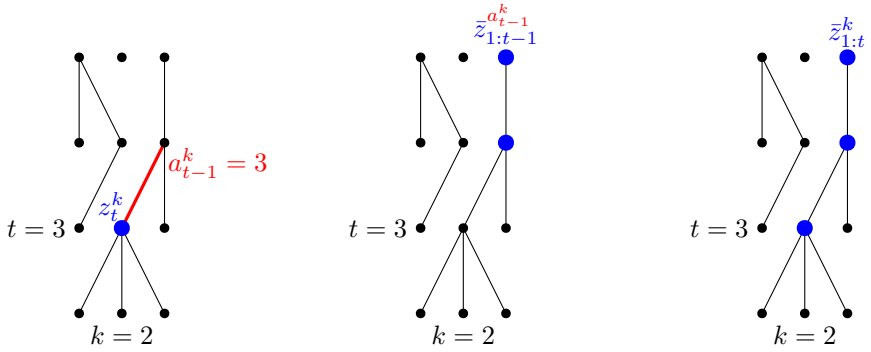


Figure 2.10: SMC in Algorithm 2 generates TK particle values $z_{1:T}^{1:K}$ and $(T-1)K$ ancestral indices $a_{1:T-1}^{1:K}$. We show three diagrams of the same run of SMC with $T = 4$ and $K = 3$. Each *particle value* z_t^k is represented as a filled circle; each *ancestral index* a_{t-1}^k is represented as an edge between z_t^k and $z_{t-1}^{a_{t-1}^k}$. At iteration t of the algorithm, we sample ancestral indices a_{t-1}^k from a Categorical distribution with probabilities $\bar{w}_{1:t-1}^{1:K}$. We then propose z_t^k by sampling from the proposal distribution $q_t(\cdot|\bar{z}_{t-1}^{a_{t-1}^k})$ conditioned on the *prefix of the kth particle set at time-step t*, $\bar{z}_{t-1}^{a_{t-1}^k}$. We append the newly proposed particle z_t^k to $\bar{z}_{t-1}^{a_{t-1}^k}$ in order to obtain the *kth particle set at time-step t*, $\bar{z}_{1:t}^k$.

Approximating target distributions. The target distributions π_t can be approximated as a weighted sum of delta masses:

$$\pi_t(z_{1:t}) \approx \sum_{k=1}^{K} \bar{w}_t^k \delta_{\bar{z}_{1:t}^k}(z_{1:t}). \tag{2.43}$$

Huggins and Roy (2015) prove convergence properties of this approximation. We can also approximate π_t after the resampling process as a sum of equally weighted delta masses, $\frac{1}{K}\sum_{k=1}^{K} \delta_{\bar{z}_{1:t}^{a_t^k}}(z_{1:t})$. However, this approximation results in higher-variance integral estimators.

Integral estimators. We can estimate the expectation of $f(z_{1:t})$ under the target distribution π_t, $\mathbb{E}_{\pi_t}[f(z_{1:t})]$ as

$$I_{\text{SMC}}^K = \sum_{k=1}^{K} \bar{w}_t^k f(\bar{z}_{1:t}^k). \tag{2.44}$$

This can be seen as evaluating the expectation with respect to the approximation given in (2.43).

Normalizing constant estimators. We can obtain the following unbiased estimators of the normalizing constants Z_t $(t = 1, \ldots, T)$:

$$\widehat{Z}_t = \prod_{\tau=1}^{t} \frac{1}{K} \sum_{k=1}^{K} w_\tau^k. \tag{2.45}$$

We are mainly interested in the estimator of the last normalizing constant Z_T, as it corresponds to the marginal likelihood $p(x_{1:T})$ of the SSM provided the sequence of target distributions is defined as (2.37)–(2.39). We provide a proof of unbiasedness of $\widehat{Z}_t$ in Appendix A.2.1.

Resampling. In Algorithm 2, the resampling process consists of sampling ancestral indices a_k $(k = 1, \ldots, K)$ *independently* from the Categorical distribution whose probabilities are given by the normalized weights $\bar{w}_{1:K}$. It is, however, only required that the ancestral indices a_k are marginally distributed according to Categorical$(\bar{w}_{1:K})$, while the joint distribution of $a_{1:K}$ need not factorize. Choosing resampling schemes that leverage this can lead to lower variance integral and normalizing constant estimators. In this thesis, we use *systematic resampling* which generates $a_{1:K}$ by Algorithm 3 below. For a comparison of resampling schemes,

Algorithm 3 Systematic resampling

1: Input: normalized weights $\bar{w}_{1:K}$.
2: Sample a single uniform random variable $u_1 \sim \text{Uniform}([0, 1/K))$.
3: **for** $k = 1, \ldots, K$ **do**
4: Set $u_k = u_1 + (k-1)/K$.
5: Set $a_k = \ell$ for ℓ such that $u_k \in [\sum_{i=1}^{\ell-1} \bar{w}_i, \sum_{i=1}^{\ell} \bar{w}_i]$.
return $a_{1:K}$.

see Douc and Cappé (2005).

2.3.2.3 Sequential Monte Carlo for Probabilistic Programming

The joint probability density of a probabilistic program can be viewed as a a sequence of target distributions if it has fixed number and order of `observe` statements. Recall that the joint probability distribution of an execution trace of a probabilistic program is given by:

$$p(z, x) = \prod_{t=1}^{T} f_{a_t}(z_t | z_{1:t-1}) \prod_{n=1}^{N} g_n(x_n | z_{1:\tau(n)}), \tag{2.46}$$

where $z_{1:\tau(n)}$ refers to the sample values obtained before encountering the nth `observe` statement and that there are no `sample` statements after the last `observe` statement. We define the unnormalized target distribution $\tilde{\pi}_n$ as the joint distribution of the program up until the nth `observe` statement:

$$\tilde{\pi}_n(z_{1:\tau(n)}) := \prod_{t=1}^{\tau(n)} f_{a_t}\left(z_t | z_{1:t-1}\right) \prod_{i=1}^{n} g_i(x_i | z_{1:\tau(i)}). \tag{2.47}$$

Hence, $\pi_N(z_{1:\tau(N)})$ corresponds to the posterior $p(z|x)$ and Z_N corresponds to the marginal likelihood $p(x)$.

We can run Algorithm 2 to obtain the integral and normalizing constant estimators (2.44) and (2.45) as follows. We run K copies of the probabilistic program forward. At every `sample` statement at time t, we sample a value z_t from the proposal distribution $q_{a_t,i_t}(\cdot|z_{1:t-1})$. When we hit an `observe` statement, we interrupt the execution of all K copies of the program in order to weigh and resample. The unnormalized weight of the kth program copy at the nth `observe` statement is computed as

$$w_n^k = \frac{\tilde{\pi}_n(\bar{z}^k_{1:\tau(n)})}{\tilde{\pi}_{n-1}(\bar{z}^{a^k_{n-1}}_{1:\tau(n-1)}) q_n(z^k_{\tau(n-1)+1:\tau(n)} | \bar{z}^{a^k_{n-1}}_{1:\tau(n-1)})} \tag{2.48}$$

$$= g_n(x_n | \bar{z}^k_{1:\tau(n)}) \cdot \left(\prod_{t=\tau(n-1)+1}^{\tau(n)} \frac{f_{a_t}\left(z_t^k | z^k_{\tau(n-1)+1:t-1}, \bar{z}^{a^k_{n-1}}_{1:\tau(n-1)}\right)}{q_{a_t,i_t}\left(z_t^k | z^k_{\tau(n-1)+1:t-1}, \bar{z}^{a^k_{n-1}}_{1:\tau(n-1)}\right)} \right), \tag{2.49}$$

where $\bar{z}^k_{1:\tau(n)} = (z^k_{\tau(n-1)+1:\tau(n)}, \bar{z}^{a^k_{n-1}}_{1:\tau(n-1)})$ is the *kth particle set at time n*. $\bar{z}^{a^k_{n-1}}_{1:\tau(n-1)}$ is the (resampled) *prefix of the kth particle set at time n* and $z^k_{\tau(n-1)+1:\tau(n)}$ are the newly sampled values of the kth particle set between $(n-1)$st and nth `observe` statement. In other words, the weight is a product of the likelihood term g_n and the ratios of the prior and proposal densities of the sample values encountered between $(n-1)$st and nth statement, $f_{a_t}/q_{a_t,i_t}$. Using weights, we can sample ancestral indices.

Implementing the resampling step requires the ability to continue the program execution multiple times from each `observe` statement onwards, while keeping the program state of the prefix the same. In pure functional languages where variable reassignment is disallowed, continuation-passing style compilation can be

used for stopping and/or continuing the program at predefined barriers (such as `observe` statements). This approach has been taken by the WebPPL (Goodman and Stuhlmüller, 2014) and the Anglican (Wood et al., 2014) systems which extend the purely functional subsets of JavaScript and Clojure respectively. In imperative languages, variable reassignment is allowed and we typically must resort to copying the entire program state when execution is copied. This approach is taken by the Turing probabilistic programming system (Ge et al., 2018) which implements execution using co-routines and copies the entire program state when a co-routine is copied. The Probabilistic-C system (Paige and Wood, 2014) propose using the `fork` function of operating systems to copy execution. However, since this language is meant to be a compilation target, the program state is not copied, instead relying on the compiler to disallow variable reassignments.

2.3.3 Variational Methods

In the previous sections, we have introduced IS and SMC which approximate difficult-to-compute posteriors by sampling. Variational methods are a family of approximate inference algorithms that are based on optimization. Recall that given an observation x, our goal is to approximate the posterior distribution $p(z|x)$ and possibly use this to make predictions or to compute expectations of the form $\mathbb{E}_{p(z|x)}[f(z)]$. The idea is to define a family of probability distributions $\mathcal{Q}$ of $\mathcal{Z}$-valued random variables and search for a member $q^*(z) \in \mathcal{Q}$ that is closest to the posterior $p(z|x)$ where closeness refers to some divergence metric:

$$q^*(z) = \operatorname*{argmin}_{q(z)\in\mathcal{Q}} \text{divergence}(q(z), p(z|x)). \tag{2.50}$$

Here, divergence$(\cdot,\cdot)$ is a function that takes two probability distributions and returns a positive value if and only if the two distributions are distinct and zero if and only if they are the same. The set $\mathcal{Q}$ is often called the "variational family" and its element $q(z) \in \mathcal{Q}$ is often called the "variational posterior" or the "variational approximation."

A common choice for the divergence is the Kullback-Leibler (KL) divergence which is defined, for arbitrary distributions π and ρ, as:

$$\mathrm{KL}\left(\pi(z)||\rho(z)\right) = \int \pi(z) \log \frac{\pi(z)}{\rho(z)} \mathrm{d}z. \tag{2.51}$$

Since the KL divergence is not symmetric, one can either choose $\mathrm{KL}\left(q||p\right)$ or $\mathrm{KL}\left(p||q\right)$. In Section 2.3.3.1, we introduce the approach that minimizes the $\mathrm{KL}\left(q||p\right)$ from q to p which is widely used in VI. We call this the "VI KL." In Section 2.3.3.2, we introduce an approach that minimizes the $\mathrm{KL}\left(p||q\right)$ from p to q which is perhaps most prominently used in the expectation propagation (EP) algorithm (Minka, 2001). We call this the "EP KL." In Section 2.3.3.3, we compare the properties of these objectives on the resulting variational approximation.

Variational methods for performing inference have first been introduced by Peterson and Anderson (1987) and Hinton et al. (1995), in the context of training neural networks. $\mathrm{KL}\left(q||p\right)$ and $\mathrm{KL}\left(p||q\right)$ are not the only divergences that can be used in variational methods. For methods that use different divergences, see e.g. Wang et al. (2018a); Li and Turner (2016); Dieng et al. (2017); Minka (2001); Hernández-Lobato et al. (2016). For papers that compare the behavior of different divergences, see e.g. Minka et al. (2005); Turner and Sahani (2011) or Turner et al. (2008). For a tutorial treatment of the subject, see e.g. Jordan et al. (1999); Wainwright et al. (2008) or Blei et al. (2017).

2.3.3.1 Minimizing the "Variational Inference" KL

We are interested in the following optimization problem:

$$q^*(z) = \underset{q(z)\in\mathcal{Q}}{\operatorname{argmin}} \mathrm{KL}\left(q(z)||p(z|x)\right). \tag{2.52}$$

This optimization problem is typically equivalently viewed as maximization of the evidence lower bound (ELBO) which is defined as:

$$\mathrm{ELBO}(q) := \log p(x) - \mathrm{KL}\left(q(z)||p(z|x)\right). \tag{2.53}$$

Maximization of (2.53) is equivalent to (2.52) because the log of the marginal likelihood (or evidence), $\log p(x)$, is independent of q. The ELBO is a lower bound to the the log evidence, $\log p(x)$, since the KL term is non-negative.

The ELBO can be rewritten as a single expectation under $q(z)$ of the log joint density minus the log density of $q(z)$:

$$\begin{aligned}
\text{ELBO}(q) &= \log p(x) - \text{KL}\left(q(z)||p(z|x)\right) \\
&= \mathbb{E}_{q(z)}[\log p(x)] - \mathbb{E}_{q(z)}[\log q(z) - \log p(z|x)] \\
&= \mathbb{E}_{q(z)}[\log p(x) - \log q(z) + \log p(z|x)] \\
&= \mathbb{E}_{q(z)}[\log p(z, x) - \log q(z)]. \qquad (2.54)
\end{aligned}$$

There are several ways to maximize the ELBO. In *mean-field* VI, one assumes that $\mathcal{Q}$ is a family of fully factorized distributions, i.e. for $z = z_{1:D}$, $\mathcal{Q} = \{q : q(z) = \prod_{d=1}^{D} q_d(z_d)\}$. Each distribution q_d has its own parameters, say ϕ_d. The mean-field optimization procedure then iteratively updates each parameter ϕ_d by solving the local maximization problem $\text{argmax}_{\phi_d} \text{ELBO}(q)$ at each iteration, while fixing all other parameters. One drawback of this method is that these local maximization problems must be solved analytically, something that is model-specific and only possible in a limited set of models.

In this thesis, we will focus on a set of methods called *black-box* VI (Ranganath et al., 2014; Paisley et al., 2012; Wingate and Weber, 2013; Kucukelbir et al., 2017), which relies on stochastic optimization (Robbins and Monro, 1951). Black-box VI merely requires that the joint density $p(z, x)$ can be evaluated. Assuming that the variational approximation $q_\phi(z)$ is parameterized by parameters ϕ, we would like to obtain noisy but unbiased estimator of the gradient $\nabla_\phi \text{ELBO}(\phi)$ using which we can update ϕ until convergence.

In general, we can obtain an unbiased gradient estimator using the REINFORCE trick:

$$\hat{g}_{\text{REINFORCE}} := (\log p(z, x) - \log q_\phi(z)) \nabla_\phi \log q_\phi(z), \qquad (2.55)$$

where $z \sim q_\phi(z)$. That this is unbiased can be seen as follows:

$$\begin{aligned}
\mathbb{E}_{q_\phi(z)}[\hat{g}_{\text{REINFORCE}}] &= \mathbb{E}_{q_\phi(z)}[(\log p(z,x) - \log q_\phi(z))\nabla_\phi \log q_\phi(z)] \\
&= \mathbb{E}_{q_\phi(z)}\left[(\log p(z,x) - \log q_\phi(z))\frac{1}{q_\phi(z)}\nabla_\phi q_\phi(z)\right] \\
&= \int (\log p(z,x) - \log q_\phi(z))\nabla_\phi q_\phi(z)\,\mathrm{d}z \\
&= \nabla_\phi \int (\log p(z,x) - \log q_\phi(z)) q_\phi(z)\,\mathrm{d}z \\
&= \nabla_\phi \,\text{ELBO}(\phi).
\end{aligned}$$

Note that the term $f(z) := (\log p(z,x) - \log q_\phi(z))$ can be any function f and the REINFORCE trick can be used for estimating any expectation of the form $\nabla_\phi \mathbb{E}_{q_\phi(z)}[f(z)]$. In fact, this estimator originated from the reinforcement learning (RL) literature (Williams, 1992) where it is used to maximize expected cumulative reward with z being the state-action trajectory sampled by the policy q_ϕ parameterized by ϕ and $f(z)$ being the expected cumulative reward. Unfortunately, for the purposes of VI, $g_{\text{REINFORCE}}$ often has too high variance which prevents effective optimization. Ranganath et al. (2014) propose variance reduction schemes which to some extent mitigate this issue.

Alternatively, we can use the reparameterization trick (Kingma and Welling, 2014; Rezende et al., 2014). This requires the random variable $z \sim q_\phi(z)$ to be *reparameterizable*. This means that there exists (i) a simple distribution $s(\epsilon)$ over $\epsilon \in \mathcal{E}$ which is independent of ϕ, and (ii) a reparameterizing function $r : \Phi \times \mathcal{E} \to \mathcal{Z}$ differentiable with respect to ϕ, such that the distribution of the random variable $r(\phi, \epsilon)$ is the same as that of z. For example, if z is normally distributed with the mean and standard deviation given by $\phi = (\mu, \sigma)$, we could pick $s(\epsilon)$ to be the standard normal distribution and $r(\phi, \epsilon) = \mu + \sigma\epsilon$. We can estimate $\nabla_\phi \,\text{ELBO}(\phi)$ using a MC estimator:

$$\hat{g}_{\text{REPARAM}} := \nabla_\phi(\log p(r(\phi,\epsilon), x) - \log q_\phi(r(\phi,\epsilon))), \tag{2.56}$$

where $\epsilon \sim s(\epsilon)$. This additionally requires $p(z,x)$ and $q_\phi(z)$ to be differentiable with respect to z and q_ϕ to be differentiable with respect to ϕ. The estimator in

(2.56) is unbiased because the distribution s is independent of ϕ:

$$\begin{aligned}\mathbb{E}_{s(\epsilon)}[\hat{g}_{\text{REPARAM}}] &= \mathbb{E}_{s(\epsilon)}\left[\nabla_\phi(\log p(r(\phi,\epsilon),x) - \log q_\phi(r(\phi,\epsilon)))\right] \\ &= \nabla_\phi \mathbb{E}_{s(\epsilon)}\left[\log p(r(\phi,\epsilon),x) - \log q_\phi(r(\phi,\epsilon))\right] \\ &= \nabla_\phi \mathbb{E}_{q_\phi(z)}\left[\log p(z,x) - \log q_\phi(z)\right].\end{aligned}$$

The last equality can be verified either through the change-of-variable formula for densities of dependent random variables or directly via the measure-theoretic definition of expected values. The $\hat{g}_{\text{REPARAM}}$ estimator typically has much lower variance than $\hat{g}_{\text{REINFORCE}}$. Unfortunately, not all random variables are reparameterizable. Perhaps most importantly, discrete random variables are not reparameterizable.

2.3.3.2 Minimizing the "Expectation Propagation" KL

Here, the goal is to minimize the KL divergence in the other direction:

$$q^*(z) = \operatorname*{argmin}_{q(z)\in\mathcal{Q}} \text{KL}\left(p(z|x)||q(z)\right). \tag{2.57}$$

Unlike minimizing the "variational inference" KL, we cannot rewrite this as an expectation under q. There are many ways to perform the optimization problem in (2.57) including the EP algorithm (Minka et al., 2018).

We focus on an approach based on stochastic optimization. Again, assume that the variational approximation $q_\phi(z)$ is parameterized by variational parameters ϕ. The gradient $\nabla_\phi \text{KL}\left(p(z|x)||q_\phi(z)\right)$ can be estimated using self-normalized IS (see Section 2.3.1.3). First, rewrite the gradient as follows:

$$\begin{aligned}\nabla_\phi \text{KL}\left(p(z|x)||q_\phi(z)\right) &= \nabla_\phi \mathbb{E}_{p(z|x)}[\log p(z|x) - \log q_\phi(z)] \\ &= \mathbb{E}_{p(z|x)}[-\nabla_\phi \log q_\phi(z)].\end{aligned}$$

This has the form $\mathbb{E}_{p(z|x)}[f(z)]$ which can be estimated using self-normalized IS:

$$\hat{g}_{\text{SNIS}} := \sum_{k=1}^{K} \bar{w}_k(-\nabla_\phi \log q_\phi(z_k)), \tag{2.58}$$

where $z_k \sim q_\phi(z)$ are independently sampled and $\bar{w}_k = w_k / \sum_{\ell=1}^{K} w_\ell$ are the normalized weights with $w_k = p(z_k,x)/q_\phi(z_k)$. $\hat{g}_{\text{SNIS}}$ is biased but converges to

the correct gradient almost surely and its asymptotic variance decreases linearly with the number of particles K. Convergence of stochastic gradient descent (SGD) is theoretically guaranteed on convex objectives and empirically on non-convex objectives (Chen and Luss, 2018). Learning ϕ using this estimator has been proposed by Oh and Berger (1992). It also forms the wake-phase update for ϕ in reweighted wake-sleep (RWS) (Bornschein and Bengio, 2015; Le et al., 2018b). Gu et al. (2015) and Perov et al. (2015) propose using an SMC-based estimator instead of self-normalized IS.

2.3.3.3 Differences Between the Two KLs

If the variational family $\mathcal{Q}$ contains the posterior distribution $p(z|x)$, the minimizer of both $\mathrm{KL}\left(q||p\right)$ and $\mathrm{KL}\left(p||q\right)$ is the posterior, $q^*(z) = p(z|x)$. How different are the minimizers when $\mathcal{Q}$ doesn't contain $p(z|x)$? Which divergence should we minimize?

The minimizer of $\mathrm{KL}\left(q||p\right) = \mathbb{E}_{q(z)}[\log q(z) - \log p(z|x)]$ generally tends to be zero-avoiding (a.k.a. mode-seeking). It will avoid placing mass to regions where $p(z)$ has small mass. This is because the term $-\log p(z|x)$ will be too large (because of the log), making the expectation too large as well. This means that if $p(z|x)$ is multi-modal, $q^*(z)$ will generally concentrate on one of the modes.

The minimizer of $\mathrm{KL}\left(p||q\right) = \mathbb{E}_{p(z|x)}[\log p(z|x) - \log q(z)]$ generally tends to be mass-covering (a.k.a. mean-seeking). It will attempt to place mass wherever there is non-zero mass in $p(z|x)$. This is because missing any mass of $p(z|x)$ will mean $-\log q(z)$ will be too large (because of the log), again making the expectation too large. If $p(z|x)$ is multi-modal, $q^*(z)$ will generally try to cover all modes and match the mean.

The zero-avoiding and mass-covering behavior is illustrated in Fig. 2.11 on a simple example where

$$
\begin{aligned}
p(z) &= \frac{1}{2}\mathrm{Normal}(z|0, 1^2) + \frac{1}{2}\mathrm{Normal}(z|\mu_p, 1^2), \\
\mathcal{Q} &= \{q(z) = \mathrm{Normal}(z|\mu_q, \sigma_q^2) : \mu_q \in \mathbb{R}, \sigma_q^2 > 0\},
\end{aligned}
$$

where μ_p is varied evenly from 0 to 10. Since the mode-seeking behavior is only a consequence of the zero-avoiding behavior, it doesn't arise when there isn't

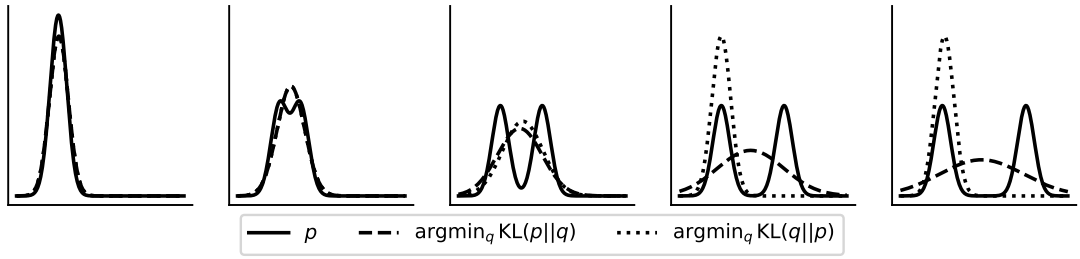


Figure 2.11: Minimizing the EP KL divergence leads to the mass-covering behavior while minimizing the VI KL divergence leads to the zero-avoiding (or mode-seeking) behavior.

sufficiently small mass in $p(z|x)$ that $q(z)$ needs to avoid. This can be seen in the second and third plots of Fig. 2.11: there are two modes but q^* is not mode-seeking because there there isn't sufficiently small mass between the two modes.

Which divergence we should minimize depends on the model. In models such as the GMM, the posterior contains many modes, each corresponding to one permutation of labels $\{1, \ldots, K\}$. A good approximation of one of the modes already lets us answer questions such as "Are data points x_i and x_j in the same cluster?", "What's the mean of the cluster of x_i?", "How many points are the cluster of x_i?". Hence, minimizing KL $(q||p)$ is suitable. In other models, we are interested in an accurate approximation of the posterior over the full space of latent variables. There, it is more suitable to minimize KL $(p||q)$. Moreover, if q is to be used as a proposal for IS, it is preferable to minimize KL $(p||q)$ since in IS, it is required that the proposal puts mass on non-zero regions of the posterior.

3
Amortized Inference

In the previous chapter, we introduced Bayesian machine learning, probabilistic programming and several general-purpose inference algorithms suitable for higher-order probabilistic programming languages (HOPPLs). These inference algorithms, however, must be re-run for every new instance of observed data. Can we do better?

Amortized inference is about making repeated inference fast. In this chapter, we propose a method for amortizing inference in probabilistic programming systems: given a probabilistic model, can we obtain an amortized inference artifact which during test time makes inference faster than performing inference from scratch? In other words, if we take seriously the hypothesis that human reasoning can be computationally explained as Bayesian inference, how is it possible for humans to make some inferences—that would in principle be exponential-time—in what seems like constant-time, or just one feed-forward, computation? We call our approach *inference compilation* as it "compiles away" the cost of performing test-time inference in a probabilistic program.

Section 3.1 provides background and related work for amortized inference. In Section 3.2, we introduce our method for amortizing inference in HOPPLs. We conclude this chapter by discussing potential future research (Section 3.3).

3.1 Background and Related Work

Approaches for making *single-shot* (as opposed to repeated) inference fast are often known as adaptive Monte Carlo methods. Adaptive importance sampling (IS) methods aim to to design proposal distributions in an online fashion to make the estimator of the integral in (2.20) have low variance (Owen, 2013, Chapter 10), (Cheng and Druzdzel, 2000). Oh and Berger (1992) target the minimization of $\mathrm{KL}\,(p||q)$. Such approaches have been widely studied in the context of Bayesian networks (Shachter and Peot, 2013; Yuan and Druzdzel, 2012; Yu and Van Engelen, 2012; Hernández et al., 1998; Salmerón et al., 2000; Ortiz and Kaelbling, 2000). Recent advances (Chwialkowski et al., 2016; Liu et al., 2016; Liu and Wang, 2016) in bridging the gap between Stein's method (Stein, 1972) and kernel methods (Hofmann et al., 2008) have allowed the development of adaptive IS methods that don't require a parametric family for the proposal distribution (Han and Liu, 2017). Adaptive sequential Monte Carlo (SMC) methods, in general, try to minimize the variance of intermediate weights in SMC by making the intermediate proposal distributions as close as possible to the so-called optimal proposal distribution (Cornebise, 2009; Cornebise et al., 2014). Other work in adaptive SMC aims to adapt the resampling step (Del Moral et al., 2012) and provide more exploration (Rainforth et al., 2018b). Adaptive Markov chain Monte Carlo (MCMC) methods focus on learning parameters of the MCMC algorithm (Roberts and Rosenthal, 2009; Haario et al., 2006; Wang et al., 2018b; Li et al., 2017), although one must take care not to violate ergodicity properties (Gilks et al., 1998; Andrieu and Thoms, 2008).

Approaches for making *repeated* inference fast, which is the focus of this chapter, can be categorized according to whether inference is learned (i) for a set of related models or for one model and (ii) before seeing data or by reusing previous inferences.

- Gershman and Goodman (2014) provide experimental evidence that humans tend to reuse inferences on related models made in the past in order to speed up current inference. With the same motivation, Choi et al. (2019) propose a method that extends the variational auto-encoder (VAE) (Kingma and Welling,

2014; Rezende et al., 2014)—in a similar way to the conditional VAE (Sohn et al., 2015)—to learn inference for related models.

- In this thesis, we focus on amortized inference for one model.
  - Methods that learn inference before seeing data are mostly in the family of the wake-sleep algorithm (Dayan et al., 1995; Hinton et al., 1995) which sample training data for an inference-aiding neural network from the prior of the probabilistic model. Paige and Wood (2016) adopt this approach for learning SMC proposals for fixed graphical models.
  - Methods for learning inference by reusing previous inferences may be categorized according to the objective they target. VAEs and related methods target $\mathrm{KL}\,(q||p)$ while methods such as reweighted wake-sleep (Bornschein and Bengio, 2015; Le et al., 2018b) and their SMC counterparts (Gu et al., 2015; Perov et al., 2015) target $\mathrm{KL}\,(p||q)$.
  - Stuhlmüller et al. (2013) and Bornschein and Bengio (2015) propose a combination of learning before seeing data and learning by reusing previous inferences.

Algorithmically, our work is similar to the methods for amortizing inference for one model, most notably, the early work on the wake-sleep algorithm and more recently VAEs. These methods also amortize inference in the manner we describe, but additionally learn the generative model. However, the class of models they focus on consists of a typically uninterpretable, independent multivariate normal latent variable and a likelihood whose parameters are obtained via a non-linear transformation of this latent variable.

Our aim is to amortize inference in structured and interpretable models defined by probabilistic programs. The approach of Paige and Wood (2016) is closest to ours in spirit: they propose learning autoregressive neural density estimation networks offline that approximate inverse factorizations of graphical models so that at test time, the trained "inference network" starts with the values of all observed quantities

and progressively proposes parameters for latent nodes in the original structured model. Our goal is to go beyond fixed graphical models to universal probabilistic programs where the inversion of the dependency structure is no longer possible. Our approach instead focuses on learning proposals for "forward" inference methods in which no model dependency inversion is performed. In this sense, our work can be seen as being inspired by that of Kulkarni et al. (2015) and Ritchie et al. (2016b) where program-specific neural proposal networks are trained to guide forward inference for inverse graphics and procedural graphics applications respectively.

What *exactly* do we mean by amortized inference? In this thesis, we define amortized inference as reducing the cost of performing repeated inference by spending time upfront to learn a fast "inference network". An inference network is a mapping from an observation x to a distribution on the latent variable z which is in some sense close to the posterior $p(z|x)$. Formally, we denote this mapping as $q : \mathcal{X} \to \mathcal{P}(\mathcal{Z})$ where $\mathcal{Z}$ and $\mathcal{X}$ refer to supports of the latent and observed variables (respectively) and $\mathcal{P}(\mathcal{Z})$ refers to the set of distributions with the support $\mathcal{Z}$. Hence, an inference network is a conditional probability distribution and we will denote it as $q(z|x)$. We take the variational approach. Given a family of inference networks, $\mathcal{Q}'$, we minimize a loss of the following form:

$$q^*(z|x) = \underset{q(z|x)\in\mathcal{Q}'}{\operatorname{argmin}} \mathbb{E}_{a(x)} \left[\text{divergence}(q(z|x), p(z|x))\right], \tag{3.1}$$

where $a(x)$ is a distribution over x. In principle, minimizing this objective amortizes inference. This is because achieving the global minimum of zero implies that the required divergence is zero almost surely. In practice, we can't achieve perfect amortization and so $a(x)$ directly affects how well inference is amortized for different observations x. It is usually chosen to be the data distribution $p(x)$, meaning that we want to amortize inference well for data sampled from the data distribution. The inference network is typically a function parameterized by, say ϕ, mapping from x to parameters of a distribution on z. Hence, we often write the inference network as $q_\phi(z|x)$, i.e. the family $\mathcal{Q}'$ is parameterized by ϕ.

Differences between variational and amortized inference. In Section 2.3.3, we describe variational inference (VI), where given a family of distributions on z, the goal is to find a member in this family that minimizes the divergence from the true posterior. As defined in the previous paragraph, in amortized inference, we start with a family of conditional distributions of z given x (or in other words functions from x to distributions on z) and find a member in this family that minimizes the expected divergence from the true posterior, where the expectation is with respect to some distribution over x. More generally, amortized inference as a method for speeding up repeated inference can be combined with non-variational approaches such as MCMC.

Differences between the two KLs. Like in VI, we have the freedom to choose the form of the divergence in (3.1). In the non-amortized setting, choosing $\mathrm{KL}\left(q||p\right)$ (the VI KL) typically results in a zero-avoiding behavior, and choosing $\mathrm{KL}\left(p||q\right)$ (the EP KL) typically results in a mass-covering behavior (see discussion in Section 2.3.3.3). This transfers to the amortized inference setting. In particular, amortized inference based on the loss of the form $\mathbb{E}[\mathrm{KL}\left(q||p\right)]$ results in inference networks that map to distributions that are zero-avoiding. On the other hand, using the loss $\mathbb{E}[\mathrm{KL}\left(p||q\right)]$ results in inference networks that map to distributions that are mass-covering. This effect is illustrated in Fig. 3.1 on an example where the generative model is given by

$$p(z) = \frac{1}{2}\mathrm{Normal}(z|-5, 1^2) + \frac{1}{2}\mathrm{Normal}(z|5, 1^2), \tag{3.2}$$

$$p(x|z) = \mathrm{Normal}(x|z, 10^2), \tag{3.3}$$

and the inference network is given by

$$q_\phi(z|x) = \mathrm{Normal}(z|\eta^1_{\phi_1}(x), \eta^2_{\phi_2}(x)), \tag{3.4}$$

where $\eta^1_{\phi_1}$ and $\eta^2_{\phi_2}$ are multi-layer perceptrons parameterized by $\phi = (\phi_1, \phi_2)$.

Which (expected) divergence we should minimize depends on the model in the same way as in the non-amortized setting. For models where a good approximation of one of the posterior modes is good enough, we should minimize $\mathbb{E}[\mathrm{KL}\left(q||p\right)]$. For models where we need to approximate the posterior over the full latent space, we

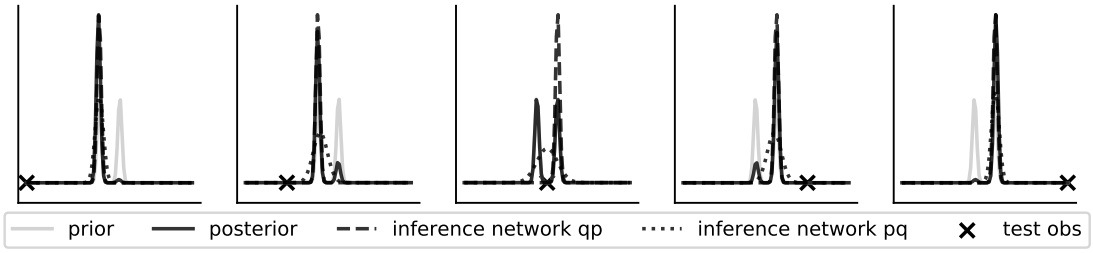


Figure 3.1: Minimizing the expected EP and VI KLs as is typically done in variational methods for inference amortization leads to the mass-covering and zero-avoiding behaviors in the inference network respectively. We show the prior distribution, the posterior distribution, and the optimal inference network under both divergences for five different observations x.

should minimize $\mathbb{E}[\text{KL}\,(p||q)]$. Moreover, if q is to be used as a proposal in IS, we should minimize $\mathbb{E}[\text{KL}\,(p||q)]$ since it is required that the proposal q puts non-zero mass on regions with non-zero mass under p (Section 2.3.1).

3.2 Inference Compilation and Universal Probabilistic Programming

In this section, we introduce a method for using deep neural networks to amortize the cost of inference in models from the family induced by universal probabilistic programming languages, establishing a framework that combines the strengths of probabilistic programming and deep learning methods. We call what we do "compilation of inference" because our method transforms a denotational specification of an inference problem in the form of a probabilistic program written in a universal programming language into a trained neural network denoted in a neural network specification language. When at test time, this neural network is fed observational data and executed, it performs approximate inference in the original model specified by the probabilistic program. Our training objective and learning procedure are designed to allow the trained neural network to be used as a proposal distribution in a IS inference engine. We illustrate our method on mixture models and Captcha solving and show significant speedups in the efficiency of inference.

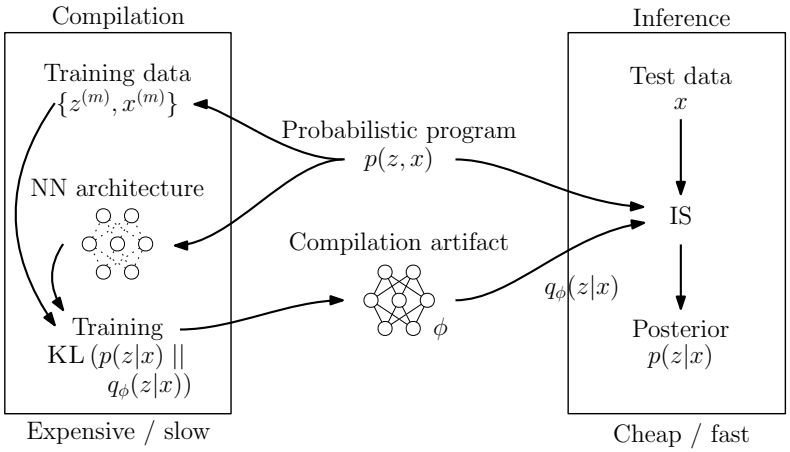


Figure 3.2: Our approach to compiled inference. Given only a probabilistic program $p(z, x)$, during *compilation* we automatically construct a neural network architecture comprising an LSTM core and various embedding and proposal layers specified by the probabilistic program and train this using an infinite stream of training data $\{z^{(m)}, x^{(m)}\}$ generated from the model. When this expensive compilation stage is complete, we are left with an artifact of weights ϕ and neural architecture specialized for the given probabilistic program. During *inference*, the probabilistic program and the compilation artifact is used in a IS procedure, where the artifact parameterizes the proposal distribution $q_\phi(z|x)$.

We focus on performing inference in generative models specified as probabilistic programs while recognizing that alternative methods exist for amortizing inference while simultaneously learning model structure. Our contributions are twofold: (1) We work out ways to handle the complexities introduced when compiling inference for the class of generative models induced by universal probabilistic programming languages and establish a technique to embed neural networks in forward probabilistic programming inference methods such as IS (Doucet and Johansen, 2009). (2) We develop an adaptive neural network architecture, comprising a recurrent neural network core and embedding and proposal layers specified by the probabilistic program, that is reconfigured on-the-fly for each execution trace and trained with an infinite stream of training data sampled from the generative model. This establishes a framework combining deep neural networks and generative modeling with universal probabilistic programs (Figure 3.2).

In Section 3.2.1 we introduce inference compilation for IS, the objective function, and the neural network architecture. Section 3.2.2 demonstrates our approach on two examples, mixture models and Captcha solving, followed by the discussion

in Section 3.2.4.

3.2.1 Learning Proposals For Importance Sampling

Recall that probabilistic programs allow us to define probabilistic models as programs that include `sample` and `observe` statements. An execution trace of a probabilistic program is obtained by successively executing the program deterministically, except when encountering `sample` statements at which point a value is generated according to the specified probability distribution and appended to the execution trace. Depending on the probabilistic program and the values generated at `sample` statements, the order in which the execution encounters `sample` statements as well as the number of encountered `sample` statements may be different from one trace to another. Therefore, given a scheme which assigns a unique address to each `sample` statement according to its lexical position in the probabilistic program, we represent an execution trace of a probabilistic program as a sequence

$$(z_t, a_t, i_t)_{t=1}^{T} \ , \tag{3.5}$$

where z_t, a_t, and i_t are respectively the sample value, address, and instance (call number) of the tth entry in a given trace, and T is a trace-dependent length. The instance value $i_t = \sum_{j=1}^{t} \mathbb{1}(a_t = a_j)$ refers to the number of `sample` statements with the same address as the address of the current one, a_t, that was encountered during the execution of the program. For each trace, a sequence $z := (z_t)_{t=1}^{T}$ holds the T sampled values from the `sample` statements. Given observations $x := x_{1:N}$, the joint probability of a single execution trace is defined in (2.19).

We achieve inference compilation in universal probabilistic programming systems through proposal distribution adaptation, approximating $p(z|x)$ in the framework of IS. Assuming we have a set of adapted proposals $q_{a_t,i_t}(z_t|z_{1:t-1}, x)$ such that their joint $q(z|x)$ is close to $p(z|x)$, the resulting inference algorithm remains unchanged from the one described in Section 2.3.1.4, except the replacement of $q_{a_t,i_t}(z_t|z_{1:t-1})$ by $q_{a_t,i_t}(z_t|z_{1:t-1}, x)$.

Figure 3.3: Results from counting and localizing objects detected in the PASCAL VOC 2007 dataset (Everingham et al., 2010). We use the corresponding categories of object detectors (i.e., person, cat, bicycle) from the MatConvNet (Vedaldi and Lenc, 2015) implementation of the Fast R-CNN (Girshick, 2015). The detector output is processed by using a high detection threshold and summarized by representing the bounding box detector output by a single central point. Inference using a single trained neural network was able to accurately identify both the number of detected objects and their locations for all categories. MAP results from 100 particles.

Inference compilation amounts to minimizing a function, specifically the loss of a neural network architecture, which makes the proposal distributions good in the sense that we specify in Section 3.2.1.1. The process of generating training data for this neural network architecture from the generative model is described in Section 3.2.1.2. At the end of training, we obtain a compilation artifact comprising the neural network components—the recurrent neural network core and the embedding and proposal layers corresponding to the original model denoted by the probabilistic program—and the set of trained weights, as described in Section 3.2.1.3.

3.2.1.1 Objective Function

We use the Kullback-Leibler (KL) divergence $\mathrm{KL}\left(p(z|x)||q_\phi(z|x)\right)$ as our measure of closeness between $p(z|x)$ and $q_\phi(z|x)$. We choose $\mathrm{KL}\left(p||q\right)$ as opposed to $\mathrm{KL}\left(q||p\right)$ since the former typically leads to mass-covering q, which is more suitable as a proposal distribution for IS. To achieve closeness over many possible x's, we take the expectation of this quantity under the distribution of $p(x)$ and ignore the terms excluding ϕ in the last equality:

$$\begin{aligned}\mathcal{L}(\phi) &:= \mathbb{E}_{p(x)}\left[\mathrm{KL}\left(p(z|x)||q_\phi(z|x)\right)\right] &(3.6)\\ &= \int_x p(x) \int_z p(z|x) \log \frac{p(z|x)}{q_\phi(z|x)} \,\mathrm{d}z \,\mathrm{d}x \\ &= \mathbb{E}_{p(z,x)}\left[-\log q_\phi(z|x)\right] + \text{const.} &(3.7)\end{aligned}$$

This objective function corresponds to the negative entropy criterion. Individual adapted proposals $q_{a_t,i_t}(z_t|\eta_t(z_{1:t-1},x,\phi)) =: q_{a_t,i_t}(z_t|z_{1:t-1},x)$ depend on η_t, the output of the neural network at time step t, parameterized by ϕ.

Considering the factorization

$$q_\phi(z|x) = \prod_{t=1}^{T} q_{a_t,i_t}(z_t|\eta_t(z_{1:t-1},x,\phi)) \;, \tag{3.8}$$

the neural network architecture must be able to map to a variable number of outputs, and incorporate sampled values in a sequential manner, concurrent with the running of the inference engine. We describe our neural network architecture in detail in Section 3.2.1.3.

3.2.1.2 Training Data

Since Eq. 3.7 is an expectation over the joint distribution, we can use the following noisy unbiased estimate of its gradient to minimize the objective:

$$\frac{\partial}{\partial\phi}\mathcal{L}(\phi) \approx \frac{1}{M}\sum_{m=1}^{M}\frac{\partial}{\partial\phi}\left(-\log q(z^{(m)}|x^{(m)};\phi)\right) \tag{3.9}$$

$$(z^{(m)},x^{(m)}) \sim p(z,x),\; m=1,\ldots,M \;. \tag{3.10}$$

Here, $(z^{(m)},x^{(m)})$ is the mth training (probabilistic program execution) trace generated by running an unconstrained probabilistic program corresponding to the original one. This unconstrained probabilistic program is obtained by a program transformation which replaces each `observe` statement in the original program by `sample` and ignores its second argument.

Universal probabilistic programming languages support stochastic branching and can generate execution traces with a changing (and possibly unbounded) number of random choices. We must, therefore, keep track of information about the addresses and instances of the samples $z_t^{(m)}$ in the execution trace, as introduced in Eq. 3.5. Specifically, we generate our training data in the form of minibatches (Cotter et al., 2011) sampled from the generative model $p(z,x)$:

$$\mathcal{D}_{\text{train}} = \left\{\left(z_t^{(m)},a_t^{(m)},i_t^{(m)}\right)_{t=1}^{T^{(m)}},\left(x_n^{(m)}\right)_{n=1}^{N}\right\}_{m=1}^{M} \;, \tag{3.11}$$

where M is the minibatch size, and, for a given trace m, the sample values, addresses, and instances are respectively denoted $z_t^{(m)}$, $a_t^{(m)}$, and $i_t^{(m)}$, and the values sampled from the distributions in `observe` statements are denoted $x_n^{(m)}$.

During compilation, training minibatches are generated on-the-fly from the probabilistic generative model and streamed to a stochastic gradient descent (SGD) procedure, specifically Adam (Kingma and Ba, 2015), for optimizing the neural network weights ϕ.

Minibatches of this infinite stream of training data are discarded after each SGD update; we therefore have no notion of a finite training set and associated issues such as overfitting to a set of training data and early stopping using a validation set (Prechelt, 1998). We do sample a validation set that remains fixed during training to compute validation losses for tracking the progress of training in a less noisy way than that admitted by the training loss.

3.2.1.3 Neural Network Architecture

Our compilation artifact is a collection of neural network components and their trained weights, specialized in performing inference in the model specified by a given probabilistic program. The neural network architecture comprises a non-domain-specific recurrent neural network (RNN) core and domain-specific observation embedding and proposal layers specified by the given program. We denote the set of the combined parameters of all neural network components ϕ.

RNNs are a popular class of neural network architecture which are well-suited for sequence-to-sequence modeling (Sutskever et al., 2014) with a wide spectrum of state-of-the-art results in domains including machine translation (Bahdanau et al., 2014), video captioning (Venugopalan et al., 2014), and learning execution traces (Reed and de Freitas, 2016). We use RNNs in this work owing to their ability to encode dependencies over time in the hidden state. In particular, we use the long short-term memory (LSTM) architecture which helps mitigate the vanishing and exploding gradient problems of RNNs (Hochreiter and Schmidhuber, 1997).

The overall architecture (Figure 3.4) is formed by combining the LSTM core with a domain-specific `observe` embedding layer f^{obs}, and several `sample` embedding layers $f^{\text{smp}}_{a,i}$ and proposal layers $f^{\text{prop}}_{a,i}$ that are distinct for each address–instance pair (a, i). As described in Section 3.2.1.2, each probabilistic program execution trace can be of different length and composed of a different sequence of addresses and instances. To handle this complexity, we define an adaptive neural network architecture that is reconfigured for each encountered trace by attaching the corresponding embedding and proposal layers to the LSTM core, creating new layers on-the-fly on the first encounter with each (a, i) pair.

Evaluation starts by computing the `observe` embedding $f^{\text{obs}}(x)$. This embedding is computed once per trace and repeatedly supplied as an input to the LSTM at each time step. Another alternative is to supply this embedding only once in the first time step, an approach preferred by Karpathy and Fei-Fei (2015) and Vinyals et al. (2015) to prevent overfitting (also see Section 3.2.2.2).

At each time step t, the input ρ_t of the LSTM is constructed as a concatenation of

1. the `observe` embedding $f^{\text{obs}}(x)$,
2. the embedding of the previous `sample` $f^{\text{smp}}_{a_{t-1},i_{t-1}}(z_{t-1})$, using zero for $t = 1$, and
3. the one-hot encodings of the current address a_t, instance i_t, and proposal type type(a_t) of the `sample` statement

for which the artifact will generate the parameter η_t of the proposal distribution $q_{a_t,i_t}(\cdot|\eta_t)$. The parameter η_t is obtained via the proposal layer $f^{\text{prop}}_{a_t,i_t}(h_t)$, mapping the LSTM output h_t through the corresponding proposal layer. The LSTM network has the capacity to incorporate inputs in its hidden state. This allows the parametric proposal $q_{a_t,i_t}(z_t|\eta_t(z_{1:t-1}, x, \phi))$ to take into account all previous samples and all observations.

During training (compilation), we supply the actual sample values $z^{(m)}_{t-1}$ to the embedding $f^{\text{smp}}_{a_{t-1},i_{t-1}}$, and we are interested in the parameter η_t in order to calculate the per-sample gradient $\frac{\partial}{\partial\phi} - \log q_{a^{(m)}_t,i^{(m)}_t}(z^{(m)}_t|\eta_t(z_{1:t-1}, x, \phi))$ to use in SGD.

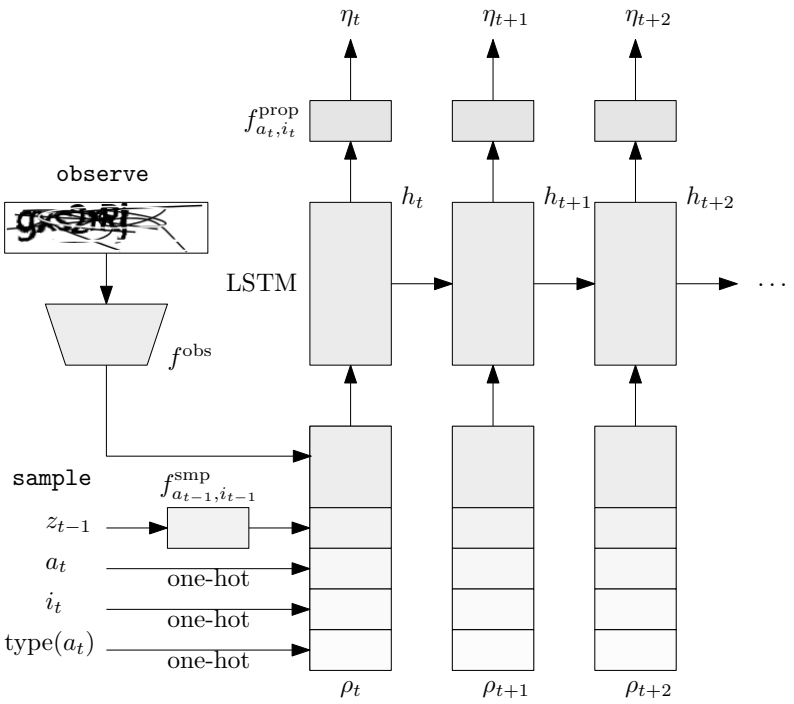


Figure 3.4: The neural network architecture. f^{obs}: `observe` embedding; $f^{\text{smp}}_{a_{t-1},i_{t-1}}$: sample embeddings; z_{t-1}: previous `sample` value; a_t, i_t, type(a_t): one-hot encodings of current address, instance, proposal type; ρ_t: LSTM input; h_t: LSTM output; $f^{\text{prop}}_{a_t,i_t}$: proposal layers; η_t: proposal parameters. Note that the LSTM core can possibly be a stack of multiple LSTMs.

During inference, the evaluation proceeds by requesting proposal parameters η_t from the artifact for specific address–instance pairs (a_t, i_t) as these are encountered. The value z_{t-1} is sampled from the proposal distribution in the previous time step.

The neural network artifact is implemented in Torch (Collobert et al., 2011), and it uses a ZeroMQ-based protocol for interfacing with the Anglican probabilistic programming system (Wood et al., 2014). This setup allows distributed training (e.g., Dean et al. (2012)) and inference with GPU support across many machines, which is beyond the scope of this paper.

3.2.2 Experiments

We demonstrate our inference compilation framework on two examples. In our first example we demonstrate an open-universe mixture model. In our second, we

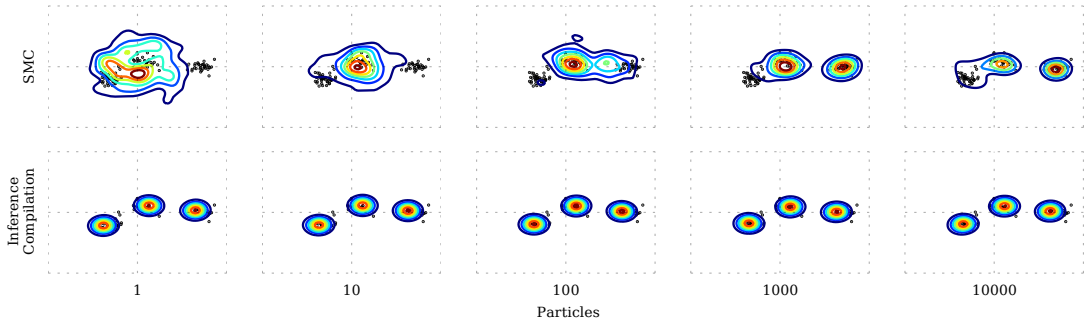


Figure 3.5: Typical inference results for an isotropic Gaussian mixture model with number of clusters fixed to $K = 3$. Shown in all panels: kernel density estimation of the distribution over maximum a posteriori values of the means $\{\max_{\mu_k} p(\mu_k|x)\}_{k=1}^{3}$ over 50 independent runs. This figure illustrates the uncertainty in the estimate of where cluster means are for each given number of particles, or equivalently, fixed amount of computation. The top row shows that, given more computation, inference, as expected, slowly becomes less noisy in expectation. In contrast, the bottom row shows that the proposal learned and used by inference compilation produces a low-noise, highly accurate estimate given even a very small amount of computation. Effectively, the encoder learns to simultaneously localize all of the clusters highly accurately.

demonstrate Captcha solving via probabilistic inference (Mansinghka et al., 2013).[1]

3.2.2.1 Mixture Models

Mixture modeling, e.g. the Gaussian mixture model (GMM) shown in Figure 3.6, is about density estimation, clustering, and counting. The inference problems posed by a GMM, given a set of vector observations, are to identify how many, where, and how big the clusters are, and optionally, which data points belong to each cluster.

We investigate inference compilation for a two-dimensional GMM in which the number of clusters is unknown. Inference arises from observing the values of x_n (Figure 3.6, line 9) and inferring the posterior number of clusters K and the set of cluster mean and covariance parameters $\{\mu_k, \Sigma_k\}_{k=1}^{K}$. We assume that the input data to this model has been translated to the origin and normalized to lie within $[-1, 1]$ in both dimensions.

In order to make good proposals for such inference, the neural network must be able to count, i.e., extract and represent information about how many clusters there are and, conditioned on that, to localize the clusters. Towards that end, we

[1] A video of inference on real test data for both examples is available at: https://youtu.be/m-FYEXVyQjQ

select a convolutional neural network as the observation embedding, whose input is a two-dimensional histogram image of binned observed data x.

In presenting observational data x assumed to arise from a mixture model to the neural network, there are some important considerations that must be accounted for. In particular, there are symmetries in mixture models (Nishihara et al., 2013) that must be broken in order for training and inference to work. First, there are $K!$ (factorial) ways to label the classes. Second, there are $N!$ ways the individual data points could be permuted. Even in experiments like ours with $K < 6$ and $N \approx 100$, this presents a major challenge for neural network training. We break the first symmetry by, at training time, sorting the clusters by the Euclidian distance of their means from the origin and relabeling all points with a permutation that labels points from the cluster nearest the original as coming from the first cluster, next closest the second, and so on. This is only approximately symmetry breaking as many different clusters may be very nearly the same distance away from the origin. Second, we avoid the $N!$ symmetry by only predicting the number, means, and covariances of the clusters, not the individual cluster assignments. The net effect of the sorting is that the proposal mechanism will learn to propose the nearest cluster to the origin as it receives training data always sorted in this manner.

Figure 3.5, where we fix the number of clusters to 3, shows that we are able to learn a proposal that makes inference dramatically more efficient than SMC (Doucet and Johansen, 2009). Figure 3.3 shows one kind of application such an efficient inference engine can do: simultaneous object counting (Lempitsky and Zisserman, 2010) and localization for computer vision, where we achieve counting by setting the prior $p(K)$ over number of clusters to be a uniform distribution over $\{1, 2, \ldots, 5\}$.

3.2.2.2 Captcha Solving

We also demonstrate our inference compilation framework by writing generative probabilistic models for Captchas (von Ahn et al., 2003) and comparing our results with the literature. Captcha solving is well suited for a generative probabilistic programming approach because its latent parameterization is low-dimensional and

procedure GAUSSIANMIXTURE
  Sample $K \sim p(K)$ ▷ `sample` number of clusters
  $\pi \sim$uniform$(1, K)$ ▷ `sample` cluster probabilities
  for $k = 1, \ldots, K$ **do**
    $\mu_k, \Sigma_k \sim p(\mu_k, \Sigma_k)$ ▷ `sample` cluster parameters
  Generate data:
  for $n = 1, \ldots, N$ **do**
    $z_n \sim p(z_n|\pi)$ ▷ `sample` class label
    $x_n \sim p(x_n|z_n = k, \mu_k, \Sigma_k)$ ▷ `sample` data
  return $x_{1:N}$

Figure 3.6: Pseudo algorithm for generating Gaussian mixtures of a variable number of clusters. At test time we `observe` data x_n and infer $K, \{\mu_k, \Sigma_k\}_{k=1}^K$.

interpretable by design. Using conventional computer vision techniques, the problem has been previously approached using segment-and-classify pipelines (Starostenko et al., 2015; Bursztein et al., 2014; Gao et al., 2014, 2013), and state-of-the-art results have been obtained by using deep convolutional neural networks (CNNs) (Goodfellow et al., 2014b; Stark et al., 2015), at the cost of requiring very large (in the order of millions) labeled training sets for supervised learning.

We start by writing generative models for each of the types surveyed by Bursztein et al. (2014), namely Baidu 2011 (), Baidu 2013 (), eBay (), Yahoo (), reCaptcha (), and Wikipedia (). Figure 3.7 provides an overall summary of our modeling approach. The actual models include domain-specific letter dictionaries, font styles, and various types of renderer noise for matching each Captcha style. In particular, implementing the displacement fields technique of Simard et al. (2003) proved instrumental in achieving our results. Given a gray-scale image, a displacement field is generated by sampling a two-dimensional displacement vector from a uniform distribution for each pixel (x, y). Then, we apply a Gaussian convolution over this vector field and normalize the vectors to obtain $(\Delta_x(x, y), \Delta_y(x, y))$ for each (x, y). Finally, a new gray level value for (x, y) is obtained by bilinearly interpolating the original image at $(x + \Delta_x(x, y), y + \Delta_y(x, y))$. Note that the parameters of stochastic renderer noise are not inferred in the example of Figure 3.7. Our experiments have shown that we can successfully train artifacts that also extract renderer noise parameters, but excluding these from the list

```
1: procedure CAPTCHA
2:     ν ∼ p(ν)                                  ▷ sample number of letters
3:     κ ∼ p(κ)                                  ▷ sample kerning value
4:     ▷ Generate letters:
5:     Λ ← {}
6:     for i = 1, ..., ν do
7:         λ ∼ p(λ)                              ▷ sample letter identity
8:         Λ ← append(Λ, λ)
9:     ▷ Render:
10:    γ ← render(Λ, κ)
11:    π ∼ p(π)                                  ▷ sample noise parameters
12:    γ ← noise(γ, π)
13:    return γ
```

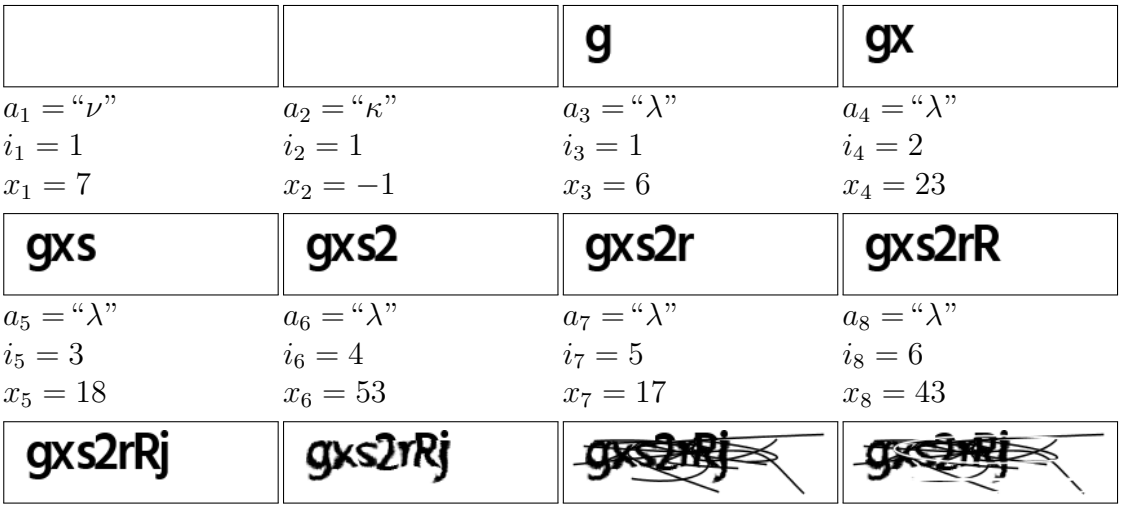


Figure 3.7: Pseudo algorithm and a sample trace of the Facebook Captcha generative process. Variations include sampling font styles, coordinates for letter placement, and language-model-like letter identity distributions $p(\lambda|\lambda_{1:t-1})$ (e.g., for meaningful Captchas). Noise parameters π may or may not be a part of inference. At test time we `observe` image γ and infer ν, Λ.

of addresses for which we learn proposal distributions improves robustness when testing with data not sampled from the same model. This corresponds to the well-known technique of adding synthetic variations to training data for transformation invariance, as used by Simard et al. (2003), Varga and Bunke (2003), Jaderberg et al. (2014), and many others.

For the compilation artifacts we use a stack of two LSTMs of 512 hidden units each,

Table 3.1: Captcha recognition rates.

	Baidu 2011	Baidu 2013	eBay	Yahoo	reCaptcha	Wikipedia	Facebook
Our method	99.8%	99.9%	99.2%	98.4%	96.4%	93.6%	91.0%
Bursztein et al. (2014)	38.68%	55.22%	51.39%	5.33%	22.67%	28.29%	
Starostenko et al. (2015)				91.5%	54.6%		
Gao et al. (2014)	34%			55%	34%		
Gao et al. (2013)		51%		36%			
Goodfellow et al. (2014b)					99.8%		
Stark et al. (2015)					90%		

an `observe`-embedding CNN consisting of six convolutions and two linear layers organized as [2×Convolution]-MaxPooling-[3×Convolution]-MaxPooling-Convolution-MaxPooling-Linear-Linear, where convolutions are 3×3 with successively 64, 64, 64, 128, 128, 128 filters, max-pooling layers are 2×2 with step size 2, and the resulting embedding vector is of length 1024. All convolutions and linear layers are followed by ReLU activation. Depending on the particular style, each artifact has approximately 20M trainable parameters. Artifacts are trained end-to-end using Adam (Kingma and Ba, 2015) with initial learning rate $\alpha = 0.0001$, hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and minibatches of size 128.

Table 3.1 reports inference results with test images sampled from the model, where we achieve very high recognition rates across the board. The reported results are obtained after approximately 16M training traces. With the resulting artifacts, running inference on a test Captcha takes < 100 ms, whereas durations ranging from 500 ms (Starostenko et al., 2015) to 7.95 s (Bursztein et al., 2014) have been reported with segment-and-classify approaches. We also compared our approach with the one by Mansinghka et al. (2013). Their method is slow since it must be run anew for each Captcha, taking in the order of minutes to solve one Captcha in our implementation of their method. The probabilistic program must also be written in a way amenable to Markov Chain Monte Carlo inference such as having auxiliary indicator random variables for rendering letters to overcome multimodality in the posterior.

We subsequently investigated how the trained models would perform on Captcha images collected from the web. We identified Wikipedia and Facebook as two major

services still making use of textual Captchas, and collected and labeled test sets of 500 images each.[2] Initially obtaining low recognition rates ($< 10\%$), with several iterations of model modifications (involving tuning of the prior distributions for font size and renderer noise), we were able to achieve 81% and 42% recognition rates with real Wikipedia and Facebook datasets, considerably higher than the threshold of 1% needed to deem a Captcha scheme broken (Bursztein et al., 2011). The fact that we had to tune our priors highlights the issues of model bias and "synthetic gap" (Zhang et al., 2015) when training models with synthetic data and testing with real data.[3]

In our experiments we also investigated feeding the `observe` embeddings to the LSTM at all time steps versus only in the first time step. We empirically verified that both methods produce equivalent results, but the latter takes significantly (approx. 3 times) longer to train. This is because we are training $f^{\mathbf{obs}}$ end-to-end from scratch, and the former setup results in more frequent gradient updates for $f^{\mathbf{obs}}$ per training trace.[4]

In summary, we only need to write a probabilistic generative model that produces Captchas sufficiently similar to those that we would like to solve. Using our inference compilation framework, we get the inference neural network architecture, training data, and labels for free. If you can create instances of a Captcha, you can break it.

3.2.3 Training Neural Networks Using Synthetic Data is Amortized Inference

How do we go about solving Captchas, and for that matter, any regression or classification task if we lack labeled training data? We must either resort to performing inference in a generative model or train using synthetic data. In the case of Captcha, Mansinghka et al. (2013) have used the former approach, which

[2]Facebook Captchas are collected from a page for accessing groups. Wikipedia Captchas appear on the account creation page.

[3]Note that the synthetic/real boundary is not always clear: for instance, we assume that the Captcha results in Goodfellow et al. (2014b) closely correspond to our results with synthetic test data because the authors have access to Google's true generative process of reCaptcha images for their synthetic training data. Stark et al. (2015) both train and test their model with synthetic data.

[4]Both Karpathy and Fei-Fei (2015) and Vinyals et al. (2015), who feed CNN output to an RNN only once, use pretrained embedding layers.

involves running inference afresh for every new instance of a Captcha image. This can take several minutes per Captcha. The deep learning community has reported remarkable results taking the latter approach, either in the limited form of data augmentation (Simard et al., 2003; Krizhevsky et al., 2012), where a dataset is artificially enlarged using label-preserving transformations, or training models solely on synthetic data, such as the groundbreaking work on text recognition in the wild (Jaderberg et al., 2014, 2016; Gupta et al., 2016), which was achieved by training a neural network to recognize text using synthetically generated realistic renders. Goodfellow et al. (2014b) addressed recognition of house numbers in Google Street View images in a supervised fashion, also solving reCaptcha (von Ahn et al., 2008) images using synthetic data to train a recognition network from image to latent text. More recently, Stark et al. (2015) also used synthetic data for Captcha-solving and Wang et al. (2015) for font identification.

Training the inference network using the objective function in (3.7) is equivalent to training a neural network using data (z, x) that comes from a synthetic data-generating distribution $p(z, x)$. If the latent variables are discrete, the objective corresponds to the cross-entropy loss that is typically used to train classifiers. A consequence of this is that there is no need to ever reuse training data, as "infinite" labeled training data can be generated at training time from the generative model. Another consequence that in training a neural network using synthetic data allows us to obtain an inference network for performing fast, repeated inference in a well-calibrated Bayesian generative model. This allows us to obtain interpretable uncertainties. In the case of Captchas, posteriors over letter identities reflect the uncertainties one would expect a human to have—the posterior is peaked if the Captcha is unambiguous and multi-modal if the Captcha is ambiguous (Figure 3.8).

The connection between training using synthetic data and amortized inference also can be seen as a reminder and guidance to the neural network community as it continues to move towards tackling unsupervised inference and problems in which labeled training data are difficult or impossible to obtain. One can draw from insights from the Bayesian model misspecification literature (Gelman and Shalizi,

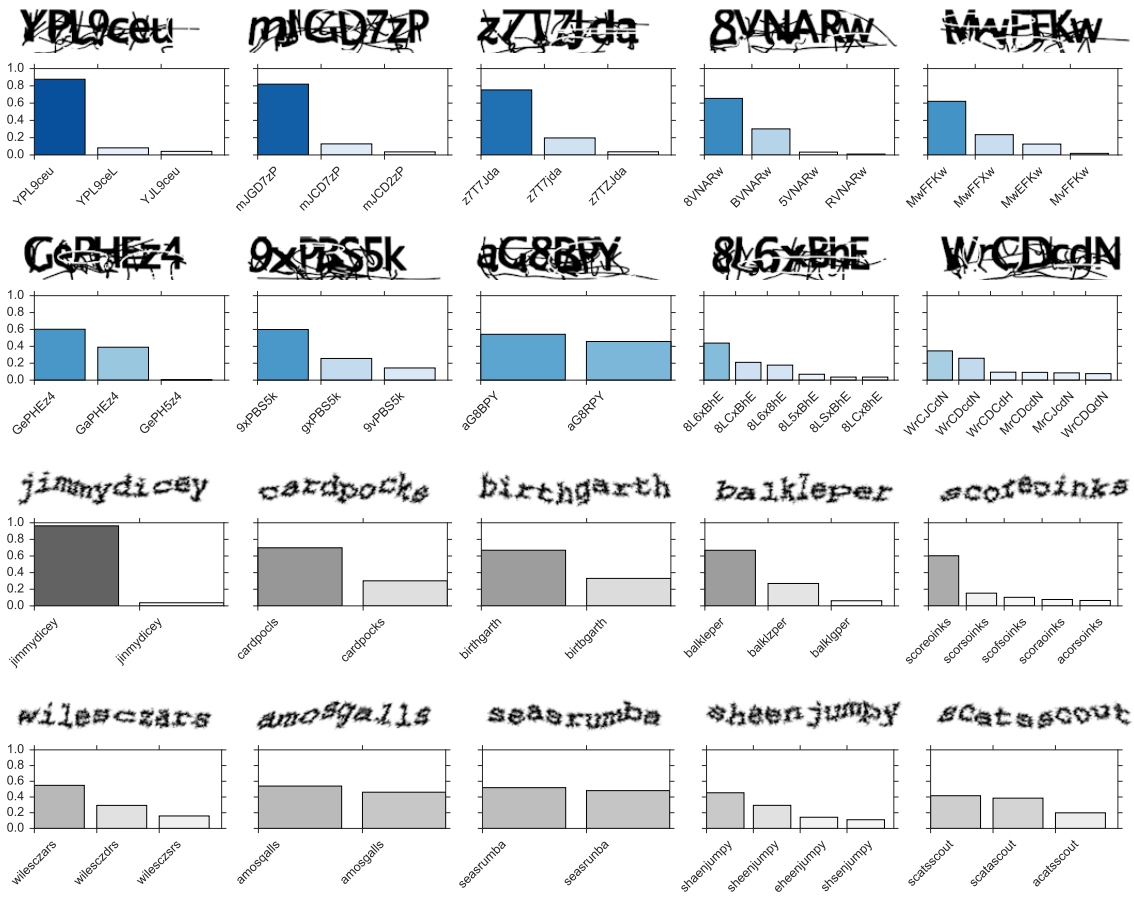


Figure 3.8: Posteriors of real Facebook and Wikipedia Captchas. Conditioning on each Captcha, we show an approximate posterior produced by a set of weighted importance sampling particles $\{(w_m, x^{(m)})\}_{m=1}^{M=100}$.

2013; Shalizi et al., 2009). The answer to the question "To what extent can we trust posterior distributions if our generative model is misspecified?" is likely to be useful in answering the question "To what extent can we trust neural networks trained with synthetic data whose distribution doesn't match the true data distribution?".

3.2.4 Discussion

We have explored making use of deep neural networks for amortizing the cost of inference in probabilistic programming. In particular, we transform an inference problem given in the form of a probabilistic program into a trained neural network architecture that parameterizes proposal distributions during IS. The amortized inference technique presented here provides a framework within which to integrate

the expressiveness of universal probabilistic programming languages for generative modeling and the processing speed of deep neural networks for inference. This merger addresses several fundamental challenges associated with its constituents: fast and scalable inference on probabilistic programs, interpretability of the generative model, an infinite stream of labeled training data, and the ability to correctly represent and handle uncertainty.

Our experimental results show that, for the family of models on which we focused, the proposed neural network architecture can be successfully trained to approximate the parameters of the posterior distribution in the `sample` space with nonlinear regression from the `observe` space. There are two aspects of this architecture that we are currently working on refining. Firstly, the structure of the neural network is not wholly determined by the given probabilistic program: the invariant LSTM core maintains long-term dependencies and acts as the glue between the embedding and proposal layers that are automatically configured for the address–instance pairs (a_t, i_t) in the program traces. We would like to explore architectures where there is a tight correspondence between the neural artifact and the computational graph of the probabilistic program. Secondly, domain-specific `observe` embeddings such as the convolutional neural network that we designed for the Captcha-solving task are hand picked from a range of fully-connected, convolutional, and recurrent architectures and trained end-to-end together with the rest of the architecture. Future work will explore automating the selection of potentially pretrained embeddings.

A limitation that comes with not learning the generative model itself—as is done by the models organized around the variational autoencoder (Kingma and Welling, 2014; Burda et al., 2016)—is the possibility of model misspecification (Shalizi et al., 2009; Gelman and Shalizi, 2013). Section 3.2.1.2 explains that our training setup is exempt from the common problem of overfitting to the training set. But as demonstrated by the fact that we needed alterations in our Captcha model priors for handling real data, we do have a risk of overfitting to the model. Therefore we need to ensure that our generative model is ideally as close as possible to the true data generation process and remember that misspecification in terms

of broadness is preferable to a misspecification where we have a narrow, but uncalibrated, model.

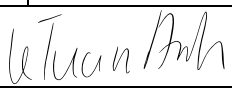
Statement of Authorship for joint/multi-authored papers for PGR thesis

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

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

Title of Paper	Inference compilation and universal probabilistic programming
Publication Status	☑ Published □ Accepted for Publication □ Submitted for Publication □ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, volume 54, pages 1338–1348, 2017a.

Student Confirmation

Student Name:	Tuan Anh Le		
Contribution to the Paper	The initial idea is FW's and mine. The initial formalization of the idea is mine. Both AGB and I worked on the experiments. I led the mixture model experiments. AGB led the Captcha experiments. The core of the paper is written by me, with help from AGB and FW. I led the initial implementations of the framework based on Anglican and Torch with AGB's help. AGB has subsequently reimplemented the framework in Pytorch, together with a new probabilistic programming system, pyprob, for use in future projects. In this section, I have modified the introduction and background to take into account preceding parts of the thesis. Section 3.2.3 contains my summary of a related publication: Tuan Anh Le, Atılım Güneş Baydin, Robert Zinkov, and Frank Wood. Using synthetic data to train neural networks is model-based reasoning. In 30th International Joint Conference on Neural Networks, pages 3514–3521, Anchorage, AK, USA, 2017b. IEEE.		
Signature		Date	27 March 2019

Supervisor Confirmation

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

Supervisor name and title: Dr Frank Wood		
Supervisor comments Tuan Anh's assertions lean towards too generous to others but are generally accurate.		
Signature 	Date	27/3/19

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

3.3 Conclusions and Future Work

In this chapter, we have introduced an objective and a neural network architecture that can be used to amortize inference in any arbitrary probabilistic program. We have shown the utility of our approach for amortizing inference in a Captcha generative model and open-universe GMMs. Baydin et al. (2018) have extended the proposed framework for amortizing inference in a generative model of particle collision and detection system. Inference in this model results in a posterior distribution over the properties of particle collisions given the particle detections. The advantage of amortizing inference in this model is that the forward generative model is known to high accuracy, owing to the accuracy of the Standard Model. However, we are far from being able to *effectively* amortize inference for all probabilistic models of interest. In the following, we outline several research directions we think can further improve amortized inference.

Automatic design of inference networks. The key requirement for amortized inference in a HOPPL is that the inference network must return a distribution at every `sample` statement which is uniquely identified using its address. We must be able to sample from this distribution and evaluate the gradient of its log density with respect to the parameters ϕ. In Le et al. (2017a), we choose an autoregressive factorization for the inference network which is highly expressive but potentially inefficient to learn. How can we design a family of inference networks that exploits the highly structured generative model? There are two promising directions to explore. First, Webb et al. (2018) propose a way of automatically deriving minimal factorizations of the inference network based on a generative model represented by a general probabilistic graphical model. For example, given a generative model $p(z)p(y|z)p(x|y)$ of an observation x, the autoregressive family of the inference network factorizes as $q(z, y|x) = q(z|x)q(y|x, z)$. However, $q(z, y|x) = q(y|x)q(z|y)$ can be easier to learn since (i) we can decompose the learning of the inference network to two parts ($q(y|x)$, and $q(z|y)$) and (ii) this factorization has to learn conditional distributions which condition on fewer variables than the autoregressive

factorization. This insight has been used to learn 3D shapes (z) from 2D images (x) via 2.5D sketches (y) (Wu et al., 2017). Second, we can use attention (Vaswani et al., 2017) (i) when the graph inversion algorithm of Webb et al. (2018) is not applicable due to the graphical model not being fixed (e.g. when the model has an open-universe or a non-parametric prior) or (ii) to further specialize the inference network conditional distributions to attend to relevant context.

Amortizing more powerful inference algorithms. Our proposed amortized inference scheme learns an inference network that maps to an efficient proposal distribution for IS. IS performs inference essentially by guessing and checking. Without an inference network that guides the guessing, IS is very inefficient. Can we amortize inference for a more powerful inference algorithm? For instance, we might want to learn an efficient sequence of proposal distributions for SMC or a proposal kernel for MCMC. The key problem in SMC is that the optimal proposal distribution has the form $q_t(z_t|z_{1:t-1}, x_{1:T}) \approx p(z_t|z_{1:t-1}, x_{1:t})$ which doesn't make use of the future data $x_{t+1:T}$. On the other hand, if we make use of the future data and learn a proposal distribution of the form $q_t(z_t|z_{1:t-1}, x_{1:T}) \approx p(z_t|z_{1:t-1}, x_{1:T})$, it is no longer optimal for the standard sequence of target distributions $\gamma_t(z_{1:t}) = p(z_{1:t}, x_{1:t})$. Instead, this proposal is optimal for the sequence of target distributions that takes into account the future observations $\gamma_t(z_{1:t}) = p(z_{1:t}, x_{1:T})$. In this sequence of target distributions, $p(z_t|z_{1:t-1}, x_{1:T})$ which takes into account the future data $x_{t+1:T}$ is the optimal proposal distribution, but now, we cannot evaluate the intermediate weights exactly since they contain terms of the form $p(x_{t+1:T}|z_{1:t})$. Lawson et al. (2018) present a potential solution to this problem, making it possible to take the best of amortized inference and SMC. Wang et al. (2018b) present a method for learning transition kernels for MCMC.

Amortized decision-making. Our amortized inference scheme learns an inference network that is close the the posterior distribution $p(z|x)$. However, this posterior distribution is almost always eventually used for evaluating an expectation of the form $\mathbb{E}_{p(z|x)}[f(z)]$. We can estimate this expectation using IS. However,

a proposal distribution that matches the posterior distribution is not optimal. Golinski et al. (2018) propose taking f into consideration when amortizing inference as well as amortizing over the distribution of fs. In decision-making, we go one step further: given a utility function (a.k.a. reward function or negative loss function) $U : \mathcal{Z} \to \mathbb{R}$ which takes in an action a, we want to choose a^* such that it minimizes the expected utility $a^* = \text{argmax}_a \mathbb{E}_{p(z|x,a)}[f(z)]$. Can we amortize the process of maximization with respect to a?

Nested amortized inference. Many probabilistic models of agent behavior (Stuhlmüller and Goodman, 2014; Seaman et al., 2018) require nested probabilistic models (Rainforth, 2018). There is an inner and an outer probabilistic model. Inference in the inner model are linked with the observations in the outer model. For instance, consider the problem of inferring agent's intention i given the observed effect o of its action a^*. Given that the agent observes x and has a latent state z, we can model its decision-making as maximization of the expected utility under the inner model $p_{\text{in}}(z|i,a)p_{\text{in}}(x|z,i,a)$, i.e. $a^* = \text{argmax}_a \mathbb{E}_{p_{\text{in}}(z|x,i,a)}[f(z)]$ given a utility function f. The outer model consists of a prior $p_{\text{out}}(x,i)$ and a likelihood $p_{\text{out}}(o|a^*)$ using which we can obtain the posterior over agent's intention $p_{\text{out}}(i|o)$. Inference in nested models is typically doubly-intractable because even evaluating the outer model's joint probability requires inference in the inner model. Can we amortize inference for the outer model by first amortizing inference for the inner one? The key technical challenge is that the inner model typically has a free variable which can vary depending on the outer model and hence we must amortize inference for a family of models. We can take inspiration from the conditional VAEs (Sohn et al., 2015) which amortizes inference over a family of models indexed by the condition variable.

4
Model Learning

In the previous chapter, we have introduced a method for amortizing inference suitable for higher-order probabilistic programming languages (HOPPLs). This allowed us to train a neural network, called the "inference network", that would take as an input the observation x and output an approximation to the posterior distribution $p(z|x)$. The evaluation of this neural network only takes one feed-forward computation and is much faster than performing approximate inference for observation x from scratch. In the case of inverting the Captcha renderer, the inference network would output a posterior distribution over letter identities, fonts, and other inputs that are fed to the renderer.

In this chapter, we focus on model learning. The key assumption of the previous chapter is that we have access to the generative model. In the Captcha example, we had to fine-tune the renderer so that the generated Captchas resemble real Captchas. Failing this would result in inaccurate inferences. Can we learn the model in a more automated way?

In Section 4.1, we introduce background knowledge and situate our work in context of other work in this area. In Section 4.2, we introduce an approach for model learning in state-space models called auto-encoding sequential Monte Carlo (AESMC). AESMC improves on previous state-of-the-art methods by exploiting the sequential nature of state-space models. In Section 4.3, we revisit an approach for

model learning called reweighted wake-sleep (RWS) (Bornschein and Bengio, 2015). RWS allows us to perform model learning in general class of HOPPL. In particular, the algorithm can deal with discrete latent variables and stochastic control flow. We conclude this chapter by discussing potential future research (Section 4.4).

4.1 Background and Related Work

We are interested in gradient-based approaches to model learning: given a family of generative models $\{p_\theta(z, x) : \theta \in \Theta\}$ parameterized by θ, we search for the best θ using gradient-based optimization. Typically, the optimization objective is the marginal likelihood $p_\theta(x)$. For methods for model learning that are not gradient-based, see literature on Bayes net structure learning (e.g. Margaritis (2003)), search-based program induction (e.g. Solar-Lezama (2008)), automatic discovery of compositional Gaussian process kernels (e.g. Duvenaud et al. (2013)), or causal inference (e.g. Pearl (2009)).

In Section 4.1.1, we revisit maximum marginal likelihood in the context of multiple observations. In Section 4.1.2, we review the variational auto-encoder (VAE) family of methods for model learning. This is based on maximizing the evidence lower bound (ELBO) which simultaneously amortizes inference.

4.1.1 Maximum Marginal Likelihood With Multiple Observations

As introduced in Section 2.1.3 model learning in latent-variable models is typically done via maximum marginal likelihood:

$$\theta^* = \operatorname*{argmax}_\theta p_\theta(x), \tag{4.1}$$

where x is a single observation under the model—in the Gaussian unknown mean model (Section 2.1.2.1), x refers to a single scalar, whereas in a state-space model (SSM), $x = x_{1:T}$ refers to a sequence of observations since the whole sequence is observed under $p(z_{1:T}, x_{1:T})$.

In this thesis, we will treat model learning as learning from *multiple* observations. Given a dataset $(x^{(n)})_{n=1}^N$, sampled independently and identically distributed (IID) from $p(x)$ and a family of generative models $p_\theta(z, x)$ of a latent variable z and data x, we would like to learn the model using maximum marginal likelihood:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \prod_{n=1}^{N} p_\theta(x^{(n)}) = \underset{\theta}{\operatorname{argmax}} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(x^{(n)}), \tag{4.2}$$

where the second equality is due to $\log(\cdot)/N$ being a monotonically increasing function which doesn't change the result of an argmax. The latter can be seen as a Monte Carlo (MC) approximation of $\mathbb{E}_{p(x)}[\log p_\theta(x)]$ whose maximization is equivalent to the minimization of $\mathrm{KL}\left(p(x)||p_\theta(x)\right) = \mathbb{E}_{p(x)}[\log p(x)] - \mathbb{E}_{p(x)}[\log p_\theta(x)]$ which is a monotonically decreasing function of $\mathbb{E}_{p(x)}[\log p_\theta(x)]$. Thus, maximum marginal likelihood of a dataset $(x^{(n)})_{n=1}^N$ in (4.2) is equivalent to minimizing the Kullback-Leibler (KL) divergence, $\mathrm{KL}\left(p(x)||p_\theta(x)\right)$. The maximization in (4.2) can also be equivalently viewed as maximum marginal likelihood under the model which contains the full dataset $p(z)\prod_{n=1}^N p(x^{(n)}|z)$.

Note that this KL divergence is mass-covering (see Section 2.3.3.3). This can be one of the reasons why generative models of images learned via maximum marginal likelihood tend to produce blurry images if $p_\theta(x)$ is not expressive enough.

4.1.2 Variational Auto-Encoders

VAE (Kingma and Welling, 2014; Rezende et al., 2014) refers to a family of methods for simultaneous model-learning and amortized inference based on the maximization of the ELBO. Like in Section 4.1.1, we are given a dataset of observations $(x^{(n)})_{n=1}^N$, sampled IID from $p(x)$ and a family of generative models $p_\theta(z, x)$. In addition, we have a family of mappings from the data space to the space of distributions on the latent variable $q_\phi(z|x)$[1], parameterized by ϕ, called the *encoder* or the *inference network*. The VAE maximizes $\frac{1}{N}\sum_{n=1}^N \mathrm{ELBO}(\theta, \phi, x^{(n)})$, where the ELBO is defined as

$$\mathrm{ELBO}(\theta, \phi, x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(z, x) - \log q_\phi(z|x)]. \tag{4.3}$$

[1]See discussion on the notation in Section 3.1.

This quantity is a lower bound to $\log p_\theta(x)$ and maximizing the VAE objective simultaneously performs model learning and inference amortization. In this thesis, we will focus on this view of the VAE. In Section 4.1.2.2, we describe how a VAE can also be viewed as a method for representation learning.

4.1.2.1 Simultaneous Model Learning and Inference Amortization

The ELBO in (4.3) can be rewritten as

$$\begin{aligned}
\text{ELBO}(\theta, \phi, x) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(z, x) - \log q_\phi(z|x)] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x) + \log p_\theta(z|x) - \log q_\phi(z|x)] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x)] - \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \\
&= \log p_\theta(x) - \text{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right). \qquad (4.4)
\end{aligned}$$

From (4.4), it follows that $\text{ELBO}(\theta, \phi, x)$ is a lower bound to $\log p_\theta(x)$ and that the gap (also called the variational bound), given by $\text{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right)$, is zero if and only if $q_\phi(z|x) = p_\theta(z|x)$ almost everywhere.

The maximization of $\frac{1}{N}\sum_{n=1}^{N} \text{ELBO}(\theta, \phi, x^{(n)})$ is an approximation to

$$\theta^*, \phi^* = \underset{\theta,\phi}{\text{argmax}} \left\{\mathbb{E}_{p(x)}[\log p_\theta(x)] - \mathbb{E}_{p(x)}[\text{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right)]\right\}, \qquad (4.5)$$

which can be seen as simultaneous model learning and inference amortization in the following sense.

Proposition 4.1. *Assuming that $\phi \in \Phi$ indexes the family of inference networks that contains all mappings from $\mathcal{X}$ to distributions on $\mathcal{Z}$, $\mathcal{Q}' = \{q : \mathcal{X} \to \mathcal{P}(\mathcal{Z})\}$, the solution to the maximization problem in* (4.5) *satisfies:*

$$\theta^* = \underset{\theta}{\text{argmax}}\, \mathbb{E}_{p(x)}[\log p_\theta(x)], \;\; \textit{and} \qquad (4.6)$$

$$\phi^* = \underset{\phi}{\text{argmin}}\, \mathbb{E}_{p(x)}\left[\text{KL}\left(q_\phi(z|x)||p_{\theta^*}(z|x)\right)\right]. \qquad (4.7)$$

The proof is given in Appendix B.1. Proposition 4.1 says that the maximization in (4.5) simultaneously achieves (4.6) and (4.7), where (4.6) achieves model learning via maximum marginal likelihood (with multiple observations) and (4.7) achieves inference amortization for the learned model (see Section 3.1).

4.1.2.2 Representation Learning

The ELBO in (4.3) can also be rewritten as

$$\begin{aligned}\text{ELBO}(\theta, \phi, x) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(z, x) - \log q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z) + \log p_\theta(z) - \log q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}\left(q_\phi(z|x)||p_\theta(z)\right). \end{aligned} \tag{4.8}$$

Given a fixed prior $p_\theta(z)$, one can see $q_\phi(z|x)$ as an encoder and $p_\theta(x|z)$ as a decoder in an auto-encoder, where the first term in (4.8) is the reconstruction loss and the second term is a regularizer for the encoder. The prior $p_\theta(z)$ is typically taken to be the standard multivariate Normal distribution, the decoder $p_\theta(x|z)$ is typically a neural network mapping from z to the means and variances of independent Normal distributions on x, and the encoder is typically a neural network mapping from x to the means and variances of independent Normal distributions on z. A summary statistic (typically the mean) of the encoder $q_\phi(z|x)$ is treated as a low-dimensional representation of data x. ELBO maximization is then a method for learning these representations in an unsupervised fashion.

4.2 Auto-Encoding Sequential Monte Carlo

In this section, we introduce a method for gradient-based model learning that builds on VAEs (Kingma and Welling, 2014; Rezende et al., 2014) and the importance weighted auto-encoder (IWAE) (Burda et al., 2016) which we call AESMC. AESMC is similarly based on maximizing a lower bound to the log marginal likelihood, but uses sequential Monte Carlo (SMC) (Doucet and Johansen, 2009) as the underlying marginal likelihood estimator instead of importance sampling (IS). For a very wide array of models, particularly those with sequential structure, SMC is a substantially more powerful inference method than IS, typically returning lower variance estimates for the marginal likelihood. Consequently, by using SMC for its marginal likelihood estimation, AESMC often leads to improvements in model learning compared with VAEs and IWAEs. We provide experiments on time-series data that show that AESMC

based learning was able to learn useful representations of the latent space for both reconstruction and prediction more effectively than the IWAE counterpart.

We express the gap between a generic ELBO based on an sampling-based inference algorithm (like IS or SMC) and the log marginal likelihood as a KL divergence between two distributions on an extended sampling space. Doing so allows us to investigate the behavior of this family of algorithms when the objective is maximized perfectly, which occurs only if the KL divergence becomes zero. In the IWAE case, this implies that the proposal distributions are equal to the posterior distributions under the learned model. In the AESMC case, it has implications for both the proposal distributions and the intermediate set of targets that are learned. We demonstrate that, somewhat counter-intuitively, using lower variance estimates for the marginal likelihood can actually be harmful to proposal learning. Using these insights, we experiment with an adaptation to the AESMC algorithm, which we call *alternating* ELBOs, that uses different lower bounds for updating the model parameters and proposal parameters. We observe that this adaptation can, in some cases, improve model learning and proposal adaptation.

4.2.1 Background

We start by briefly reviewing notation for SSMs, SMC and IWAE. The first two topics are covered in more detail in Sections 2.1.2.3, 2.3.2 respectively.

4.2.1.1 State-Space Models

SSMs are probabilistic models over a set of latent variables $z_{1:T}$ and observed variables $x_{1:T}$. Given parameters θ, a SSM is characterized by an initial density $\mu_\theta(z_1)$, a series of transition densities $f_{t,\theta}(z_t|z_{1:t-1})$, and a series of emission densities $g_{t,\theta}(x_t|z_{1:t})$ with the joint density being $p_\theta(z_{1:T}, x_{1:T}) = \mu_\theta(z_1)\prod_{t=2}^{T} f_{t,\theta}(z_t|z_{1:t-1})\prod_{t=1}^{T} g_{t,\theta}(x_t|z_{1:t})$. We will consider model learning as a problem of maximizing the marginal likelihood $p_\theta(x_{1:T}) = \int p_\theta(z_{1:T}, x_{1:T})\,\mathrm{d}z_{1:T}$ in the family of models parameterized by θ.

4.2.1.2 Sequential Monte Carlo

SMC performs approximate inference on a sequence of target distributions $(\pi_t(z_{1:t}))_{t=1}^T$. In the context of SSMs, the target distributions are often taken to be $(p_\theta(z_{1:t}|x_{1:t}))_{t=1}^T$. SMC assumes that we have access to the proposal distributions $q_{1,\phi}(z_1|x_1)$ and $(q_{t,\phi}(z_t|x_{1:t}, z_{1:t-1}))_{t=2}^T$ from which we can sample and whose densities we can evaluate. These proposal distributions are parameterized by a set of parameters ϕ. Algorithm 4 shows SMC in the context of an SSM parameterized by θ and proposal distributions parameterized by ϕ. This is the same algorithm as the general one shown in Algorithm 2 which is in the context of sequence of target distributions.

Using the set of weighted particles $(\tilde{z}_{1:T}^k, w_T^k)_{k=1}^K$ at the last time step, we can approximate the posterior as $\sum_{k=1}^K \bar{w}_T^k \delta_{\tilde{z}_{1:T}^k}(z_{1:T})$ and the integral I_φ as $\sum_{k=1}^K \bar{w}_T^k \varphi(\tilde{z}_{1:T}^k)$, where $\bar{w}_T^k := w_T^k / \sum_j w_T^j$ is the normalized weight and δ_z is a Dirac measure centered on z. Furthermore, one can obtain an unbiased estimator of the marginal likelihood $p_\theta(x_{1:T})$ using the intermediate particle weights:

$$\hat{Z}_{\text{SMC}} := \prod_{t=1}^T \left[\frac{1}{K} \sum_{k=1}^K w_t^k \right]. \tag{4.9}$$

The sequential nature of SMC and the resampling step are crucial in making SMC scalable to large T. The former makes it easier to design efficient proposal distributions as each step need only target the next set of variables z_t. The resampling step allows the algorithm to focus on promising particles in light of new observations, avoiding the exponential divergence between the weights of different samples that occurs for importance sampling as T increases. This can be demonstrated both empirically and theoretically (Del Moral, 2004, Chapter 9). We refer the reader to (Doucet and Johansen, 2009) for an in-depth treatment of SMC.

4.2.1.3 Importance Weighted Auto-Encoders

Given a dataset of observations $(x^{(n)})_{n=1}^N$, a generative network $p_\theta(z, x)$ and an inference network $q_\phi(z|x)$, IWAE (Burda et al., 2016) maximizes $\frac{1}{N}\sum_{n=1}^N \text{ELBO}_{\text{IS}}(\theta, \phi, x^{(n)})$

Algorithm 4 Sequential Monte Carlo

1: Sample initial particle values $z_1^k \sim q_{1,\phi}(\cdot|x_1)$.
2: Compute and normalize weights:

$$w_1^k = \frac{\mu_\theta(z_1^k) g_{1,\theta}(x_1|z_1^k)}{q_{1,\phi}(z_1^k|x_1)}, \qquad \bar{w}_1^k = \frac{w_1^k}{\sum_{\ell=1}^K w_1^\ell}.$$

3: Initialize particle set: $\tilde{z}_1^k \leftarrow z_1^k$
4: **for** $t = 2, 3, \ldots, T$ **do**
5: Sample ancestor index $a_{t-1}^k \sim \text{Categorical}(\cdot|\bar{w}_{t-1}^1, \ldots, \bar{w}_{t-1}^K)$.
6: Sample particle value $z_t^k \sim q_{t,\phi}(\cdot|x_{1:t}, \tilde{z}_{1:t-1}^{a_{t-1}^k})$.
7: Update particle set $\tilde{z}_{1:t}^k \leftarrow (\tilde{z}_{1:t-1}^{a_{t-1}^k}, z_t^k)$.
8: Compute and normalize weights:

$$w_t^k = \frac{f_{t,\theta}(z_t^k|\tilde{z}_{1:t-1}^{a_{t-1}^k}) g_{t,\theta}(x_t|\tilde{z}_{1:t}^k)}{q_{t,\phi}(z_t^k|x_{1:t}, \tilde{z}_{1:t-1}^{a_{t-1}^k})}, \qquad \bar{w}_t^k = \frac{w_t^k}{\sum_{\ell=1}^K w_t^\ell}.$$

9: Compute marginal likelihood: $\hat{Z}_{\text{SMC}} = \prod_{t=1}^T \frac{1}{K} \sum_{k=1}^K w_t^k$.
10: **return** particles $(\tilde{z}_{1:T}^k)_{k=1}^K$, weights $(w_T^k)_{k=1}^K$, marginal likelihood estimate $\hat{Z}_{\text{SMC}}$

where, for a given observation x, the ELBO_{IS} (with K particles) is a lower bound on $\log p_\theta(x)$ by Jensen's inequality:

$$\text{ELBO}_{\text{IS}}(\theta, \phi, x) = \int Q_{\text{IS}}(z^{1:K}) \log \hat{Z}_{\text{IS}}(x^{1:K}) \, \mathrm{d}x^{1:K} \leq \log p_\theta(x), \text{ where} \tag{4.10}$$

$$Q_{\text{IS}}(z^{1:K}) = \prod_{k=1}^K q_\phi(z^k|x), \quad \hat{Z}_{\text{IS}}(x^{1:K}) = \frac{1}{K} \sum_{k=1}^K \frac{p_\theta(x^k, y)}{q_\phi(z^k|x)}. \tag{4.11}$$

Optimization of $\text{ELBO}_{\text{IS}}(\theta, \phi, x)$ is performed using stochastic gradient ascent (SGA) where a sample from $\left(\prod_{k=1}^K q_\phi(z^k|x^{(n)})\right)$ is obtained using the reparameterization trick (Kingma and Welling, 2014) and the gradient $\frac{1}{N} \sum_{n=1}^N \nabla_{\theta,\phi} \log \left(\sum_{k=1}^K \frac{p_\theta(x^k, x^{(n)})}{q_\phi(z^k|x^{(n)})}\right)$ is used to perform an optimization step.

Note that for $K = 1$ particle, this objective reduces to a VAE objective which we will also refer to as

$$\text{ELBO}_{\text{VAE}}(\theta, \phi, x) = \int q_\phi(z|x)(\log p_\theta(z, x) - \log q_\phi(z|x)) \, \mathrm{d}z. \tag{4.12}$$

4.2.2 Approach

AESMC implements model learning, proposal adaptation, and inference amortization in a similar manner to the VAE and the IWAE: it uses SGA on an empirical average

of the ELBO over observations. However, it varies in the form of this ELBO. In this section, we will introduce the AESMC ELBO, explain how gradients of it can be estimated, and discuss the implications of these changes.

4.2.2.1 Objective Function

Consider a family of SSMs $\{p_\theta(z_{1:T}, x_{1:T}) : \theta \in \Theta\}$ and a family of proposal distributions $\{q_\phi(z_{1:T}|x_{1:T}) = q_{1,\phi}(z_1|x_1)\prod_{t=2}^{T} q_{t,\phi}(z_t|z_{1:t-1}, x_{1:t}) : \phi \in \Phi\}$. AESMC uses an ELBO objective based on the SMC marginal likelihood estimator (4.9). In particular, for a given $x_{1:T}$, the objective is defined as

$$\mathrm{ELBO}_{\mathrm{SMC}}(\theta, \phi, x_{1:T}) := \int Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \,\mathrm{d}z_{1:T}^{1:K} \,\mathrm{d}a_{1:T-1}^{1:K}, \tag{4.13}$$

where $\hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$ is defined in (4.9) and Q_{SMC} is the sampling distribution of SMC,

$$Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) = \left(\prod_{k=1}^{K} q_{1,\phi}(z_1^k)\right)\left(\prod_{t=2}^{T}\prod_{k=1}^{K} q_{t,\phi}(z_t^k|\tilde{z}_{1:t-1}^{a_{t-1}^k}) \cdot \mathrm{Categorical}(a_{t-1}^k|w_{t-1}^{1:K})\right). \tag{4.14}$$

$\mathrm{ELBO}_{\mathrm{SMC}}$ forms a lower bound to the log marginal likelihood $\log p_\theta(x_{1:T})$ due to Jensen's inequality and the unbiasedness of the marginal likelihood estimator. Hence, given a dataset $(x_{1:T}^{(n)})_{n=1}^N$, we can perform model learning based on maximizing the lower bound of $\frac{1}{N}\sum_{n=1}^N \log p_\theta(x_{1:T}^{(n)})$ as a surrogate target, namely by maximizing

$$\mathcal{J}(\theta, \phi) := \frac{1}{N}\sum_{n=1}^{N} \mathrm{ELBO}_{\mathrm{SMC}}(\theta, \phi, x_{1:T}^{(n)}). \tag{4.15}$$

For notational convenience, we will talk about optimizing ELBOs in the rest of this section. However, we note that the main intended use of AESMC is to amortize over datasets, for which the ELBO is replaced by the dataset average $\mathcal{J}(\theta, \phi)$ in the optimization target. Nonetheless, rather than using the full dataset for each gradient update, will we instead use mini-batches, noting that this forms unbiased estimator.

4.2.2.2 Gradient Estimation

We describe a gradient estimator used for optimizing $\text{ELBO}_{\text{SMC}}(\theta, \phi, x_{1:T})$ using SGA. The SMC sampler in Algorithm 4 proceeds by sampling $z_1^{1:K}, a_1^{1:K}, z_2^{1:K}, \ldots$ sequentially from their respective distributions $\prod_{k=1}^K q_1(z_1^k)$, $\prod_{k=1}^K \text{Categorical}(a_1^k | w_1^{1:K})$, $\prod_{k=1}^K q_2(z_2^k | z_1^{a_1^k}), \ldots$ until the whole particle-weight trajectory $(z_{1:K}^{1:T}, a_{1:T-1}^{1:K})$ is sampled. From this trajectory, using equation (4.9), we can obtain an estimator for the marginal likelihood.

Assuming that the sampling of latent variables $z_{1:T}^{1:K}$ is reparameterizable, we can make their sampling independent of (θ, ϕ). In particular, assume that there exists a set of auxiliary random variables $\epsilon_{1:T}^{1:K}$ where $\epsilon_t^k \sim s_t$ and a set of reparameterization functions r_t. We can simulate the SMC sampler by first sampling $\epsilon_1^{1:K} \sim \prod_{k=1}^K s_1$ and setting $z_1^k = r_1(\epsilon_1^k)$ and $\tilde{z}_1^k = z_1^k$, then for $t = 2, \ldots, T$ cycling through sampling $a_{t-1}^{1:K} \sim \prod_{k=1}^K \text{Categorical}(a_{t-1}^k | w_{t-1}^{1:K})$ and $\epsilon_t^{1:K} \sim \prod_{k=1}^K s_t$, and setting $z_t^k = r_t(\epsilon_t^k, \tilde{z}_{1:t-1}^{a_{t-1}^k})$ and $\tilde{z}_{1:t}^k = (\tilde{z}_{1:t-1}^{a_{t-1}^k}, z_t^k)$. We use the resulting reparameterized sample of $(z_{1:K}^{1:T}, a_{1:T-1}^{1:K})$ to evaluate the gradient estimator $\nabla_{\theta,\phi} \log \hat{Z}_{\text{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$.

To account for the discrete choices of ancestor indices a_t^k one could additionally use the REINFORCE (Williams, 1992) trick, however in practice, we found that the additional term in the estimator has problematically high variance. We explore various other possible gradient estimators and empirical assessments of their variances in Appendix B.2.1. This exploration confirms that including the additional REINFORCE terms leads to problematically high variance, justifying our decision to omit them, despite introducing bias into the gradient estimates.

4.2.2.3 Bias & Implications on the Proposals

In this section, we express the gap between ELBOs and the log marginal likelihood as a KL divergence and study implications on the proposal distributions. We present a set of claims and propositions whose full proofs are in Appendix B.2.2. These give insight into the behavior of AESMC and show the advantages, and disadvantages, of using our different ELBO. This insight motivates Section 4.2.3 which proposes an algorithm for improving proposal learning.

Definition 4.2. *Given an* unnormalized target density $\tilde{P} : \mathcal{Z} \to [0, \infty)$ *with* normalizing constant $Z_P > 0$, $P := \tilde{P}/Z_P$, *and a* proposal density $Q : \mathcal{Z} \to [0, \infty)$, *then*

$$\textsc{elbo} := \int Q(z) \log \frac{\tilde{P}(z)}{Q(z)} \, \mathrm{d}z, \tag{4.16}$$

is a lower bound on $\log Z_P$ *and satisfies*

$$\textsc{elbo} = \log Z_P - \textsc{kl}\left(Q||P\right). \tag{4.17}$$

This is a standard identity used in variational inference and VAEs. In the case of VAEs, applying Definition 4.2 with P being $p_\theta(z|x)$, $\tilde{P}$ being $p_\theta(z, x)$, Z_P being $p_\theta(x)$, and Q being $q_\phi(z|x)$, we can directly rewrite (4.12) as $\textsc{elbo}_{\text{VAE}}(\theta, \phi, x) = \log p_\theta(x) - \textsc{kl}\left(q_\phi(z|x)||p_\theta(z|x)\right)$.

The key observation for expressing such a bound for general ELBOs such as $\textsc{elbo}_{\text{IS}}$ and $\textsc{elbo}_{\text{SMC}}$ is that the target density P and the proposal density Q need not directly correspond to $p_\theta(z|x)$ and $q_\phi(z|x)$. This allows us to view the underlying sampling distributions of the marginal likelihood Monte Carlo estimators such as Q_{IS} in (4.11) and Q_{SMC} in (4.14) as proposal distributions on an extended space $\mathcal{Z}$. The following claim uses this observation to express the bound between a general ELBO and the log marginal likelihood as KL divergence from the extended space sampling distribution to a corresponding target distribution.

Claim 4.3. *Given a non-negative unbiased estimator* $\hat{Z}_P(z) \geq 0$ *of the normalizing constant* Z_P *where* z *is distributed according to the proposal distribution* $Q(z)$, *the following holds:*

$$\textsc{elbo} = \int Q(z) \log \hat{Z}_P(z) \, \mathrm{d}z = \log Z_P - \textsc{kl}\left(Q||P\right), \tag{4.18}$$

$$\textit{where} \quad P(z) = \frac{Q(z)\hat{Z}_P(z)}{Z_P} \tag{4.19}$$

is the implied normalized target density.

In the case of IWAEs, we can apply Claim 4.3 with Q and $\hat{Z}_P$ being Q_{IS} and $\hat{Z}_{\mathrm{IS}}$ respectively as defined in (4.11) and Z_P being $p_\theta(x)$. This yields

$$\mathrm{ELBO}_{\mathrm{IS}}(\theta, \phi, x) = \log p_\theta(x) - \mathrm{KL}\left(Q_{\mathrm{IS}} || P_{\mathrm{IS}}\right), \text{ where} \tag{4.20}$$

$$P_{\mathrm{IS}}(z^{1:K}) = \frac{1}{K}\sum_{k=1}^{K}\left(q_\phi(z^1|x)\cdots q_\phi(z^{k-1}|x)p_\theta(z^k|x)q_\phi(z^{k+1}|x)\cdots q_\phi(z^K|x)\right). \tag{4.21}$$

Similarly, in the case of AESMC, we obtain

$$\mathrm{ELBO}_{\mathrm{SMC}}(\theta, \phi, x_{1:T}) = \log p_\theta(x_{1:T}) - \mathrm{KL}\left(Q_{\mathrm{SMC}} || P_{\mathrm{SMC}}\right), \text{ where} \tag{4.22}$$

$$P_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) = Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})\hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})/p_\theta(x_{1:T}). \tag{4.23}$$

Having expressions for the target distribution P and the sampling distribution Q for a given ELBO allows us to investigate what happens when we maximize that ELBO, remembering that the KL term is strictly non-negative and zero if and only if $P = Q$. For the VAE and IWAE cases then, provided the proposal is sufficiently flexible, one can always perfectly maximize the ELBO by setting $p_\theta(z|x) = q_\phi(z|x)$ for all x. The reverse implication also holds: if $\mathrm{ELBO}_{\mathrm{VAE}} = \log Z_P$ then it must be the case that $p_\theta(z|x) = q_\phi(z|x)$. However, for AESMC, achieving $\mathrm{ELBO} = \log Z_P$ is only possible when one also has sufficient flexibility to learn a particular series of intermediate target distributions, namely the marginals of the final target distribution. In other words, it is necessary to learn a particular factorization of the generative model, not just the correct individual proposals, to achieve $P = Q$ and thus $\mathrm{ELBO}_{\mathrm{SMC}} = Z_P$. These observations are formalized in Propositions 4.4 and 4.5 below.

Proposition 4.4. *$Q_{IS}(z^{1:K}) = P_{IS}(z^{1:K})$ for all $z^{1:K}$ if and only if $q(z|x) = p(z|x)$ for all z.*

Proposition 4.5. *If $K > 1$, then $P_{SMC}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) = Q_{SMC}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$ for all $(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$ if and only if*

1. *$\pi_t(z_{1:t}) = \int p(z_{1:T}|x_{1:T})\,\mathrm{d}z_{t+1:T} = p(z_{1:t}|x_{1:T})$ for all $z_{1:t}$ and $t = 1, \ldots, T$, and*
2. *$q_1(z_1|x_1) = p(z_1|x_{1:T})$ for all z_1 and $q_t(z_t|z_{1:t-1}, x_{1:t}) = p(z_{1:t}|x_{1:T})/p(z_{1:t-1}|x_{1:T})$ for $t = 2, \ldots, T$ for all $z_{1:t}$,*

where $\pi_t(z_{1:t})$ are the intermediate targets used by SMC.

Proposition 4.5 has the consequence that if the family of generative models is such that the first condition does not hold, we will not be able to make the bound tight. This means that, except for a very small class of models, then, for most convenient parameterizations, it will be impossible to learn a perfect proposal that gives a tight bound, i.e. there will be no θ and ϕ such that the above conditions can be satisfied. However, it also means that ELBO_{SMC} encodes important additional information about the implications the factorization of the generative model has on the inference—the model depends only on the final target $\pi_T(z_{1:T}) = p_\theta(z_{1:T}|x_{1:T})$, but some choices of the intermediate targets $\pi_t(z_{1:t})$ will lead to much more efficient inference than others. Perhaps more importantly, SMC is usually a far more powerful inference algorithm than importance sampling and so the AESMC setup allows for more ambitious model learning problems to be effectively tackled than the VAE or IWAE. After all, even though it is well known in the SMC literature that, unlike for IS, most problems have no perfect set of SMC proposals which will generate exact samples from the posterior (Doucet and Johansen, 2009), SMC still gives superior performance on most problems with more than a few dimensions. These intuitions are backed up by our experiments that show that using ELBO_{SMC} regularly learns better models than using ELBO_{IS}.

4.2.3 Improving Proposal Learning

In practice, one is rarely able to perfectly drive the divergence to zero and achieve a perfect proposal. In addition to the implications of the previous section, this occurs because $q_\phi(z_{1:T}|x_{1:T})$ may not be sufficiently expressive to represent $p_\theta(z_{1:T}|x_{1:T})$ exactly and because of the inevitable sub-optimality of the optimization process, remembering that we are aiming to learn an amortized inference artifact, rather than a single posterior representation. Consequently, to accurately assess the merits of different ELBOs for proposal learning, it is necessary to consider their finite-time performance. We therefore now consider the effect the number of particles K has on the gradient estimators for ELBO_{IS} and ELBO_{SMC}.

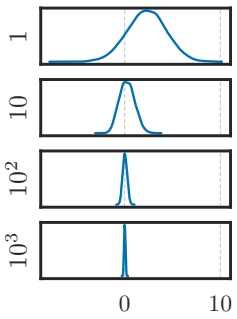


Figure 4.1: Density estimate of ∇_ϕ ELBO for different K

Counter-intuitively, it transpires that the tighter bounds implied by using a larger K is often harmful to proposal learning for both IWAE and AESMC. At a high-level, this is because an accurate estimate for $\hat{Z}_P$ can be achieved for a wide range of proposal parameters ϕ and so the magnitude of ∇_ϕ ELBO reduces as K increases. Typically, this shrinkage happens faster than increasing K reduces the standard deviation of the estimate and so the standard deviation of the gradient estimate relative to the problem scaling (i.e. as a ratio of true gradient ∇_ϕ ELBO) actually increases. This effect is demonstrated in Figure 4.1 which shows a kernel density estimator for the distribution of the gradient estimate for different K and the model given in Section 4.2.4.2. Here we see that as we increase K, both the expected gradient estimate (which is equal to the true gradient by unbiasedness) and standard deviation of the estimate decrease. However, the former decreases faster and so the relative standard deviation increases. This is perhaps easiest to appreciate by noting that for $K > 10$, there is a roughly equal probability of the estimate being positive or negative, such that we are equally likely to increase or decrease the parameter value at the next SGA iteration, inevitably leading to poor performance. On the other hand, when $K = 1$, it is far more likely that the gradient estimate is positive than negative, and so there is clear drift to the gradient steps. We add to the empirical evidence for this behavior in Section 4.2.4. Note the critical difference for model learning is that ∇_θ ELBO does not, in general, decrease in magnitude as K increases. Note also that using a larger K should always give better performance at test time; it may though be better to learn ϕ using a smaller K.

In simultaneously developed work (Rainforth et al., 2017), we formalized this intuition in the IWAE setting by showing that the estimator of $\nabla_\phi \text{ELBO}_{\text{IS}}(\theta, \phi, x)$ with K particles, denoted by I_K, has the following signal-to-noise ratio (SNR):

$$\text{SNR} := \frac{\mathbb{E}[I_K]}{\sqrt{\text{Var}[I_K]}} = O\left(\sqrt{\frac{1}{K}}\right). \tag{4.24}$$

We thus see that increasing K reduces the SNR and so the gradient updates for the proposal will degrade towards pure noise if K is set too high.

4.2.3.1 Alternating ELBOs

To address these issues, we suggest and investigate the alternating ELBOs (ALT) algorithm which updates (θ, ϕ) in a coordinate descent fashion using different ELBOs, and thus gradient estimates, for each. We pick a θ-optimizing pair and a ϕ-optimizing pair $(A_\theta, K_\theta), (A_\phi, K_\phi) \in \{\text{IS}, \text{SMC}\} \times \{1, 2, \dots\}$, corresponding to an inference type and number of particles. In an optimization step, we obtain an estimator for $\nabla_\theta \, \text{ELBO}_{A_\theta}$ with K_θ particles and an estimator for $\nabla_\phi \, \text{ELBO}_{A_\phi}$ with K_ϕ particles which we call g_θ and g_ϕ respectively. We use g_θ to update the current θ and g_ϕ to update the current ϕ. The results from the previous sections suggest that using $A_\theta = \text{SMC}$ and $A_\phi = \text{IS}$ with a large K_θ and a small K_ϕ may perform better model and proposal learning than just fixing $(A_\theta, K_\theta) = (A_\phi, K_\phi)$ to $(\text{SMC}, \text{large})$ since using $A_\phi = \text{IS}$ with small K_ϕ helps learning ϕ (at least in terms of the SNR) and using $A_\theta = \text{SMC}$ with large K_θ helps learning θ. We experimentally observe that this procedure can in some cases improve both model and proposal learning.

4.2.4 Experiments

We now present a series of experiments designed to answer the following questions: 1) Does tightening the bound by using either more particles or a better inference procedure lead to an adverse effect on proposal learning? 2) Can AESMC, despite this effect, outperform IWAE? 3) Can we further improve the learned model and proposal by using ALT?

First we investigate a linear Gaussian state space model (LGSSM) for model learning and a latent variable model for proposal adaptation. This allows us to compare the learned parameters to the optimal ones. Doing so, we confirm our conclusions for this simple problem.

We then extend those results to more complex, high dimensional observation spaces that require models and proposals parameterized by neural networks. We

do so by investigating the *Moving Agents* dataset, a set of partially occluded video sequences.

4.2.4.1 Linear Gaussian State Space Model

Given the following LGSSM

$$p(z_1) = \text{Normal}\left(z_1; 0, 1^2\right), \tag{4.25}$$

$$p(z_t|z_{t-1}) = \text{Normal}\left(z_t; \theta_1 z_{t-1}, 1^2\right), \qquad t = 2, \dots T, \tag{4.26}$$

$$p(x_t|z_t) = \text{Normal}\left(x_t; \theta_2 z_t, \sqrt{0.1}^2\right), \qquad t = 1, \dots, T, \tag{4.27}$$

we find that optimizing $\text{ELBO}_{\text{SMC}}(\theta, \phi, x_{1:T})$ w.r.t. θ leads to better generative models than optimizing $\text{ELBO}_{\text{IS}}(\theta, \phi, x_{1:T})$. The same is true for using more particles.

We generate a sequence $x_{1:T}$ for $T = 200$ by sampling from the model with $\theta = (\theta_1, \theta_2) = (0.9, 1.0)$. We then optimize the different ELBOs w.r.t. θ using the bootstrap proposal $q_1(z_1|x_1) = \mu_\theta(z_1)$ and $q_t(z_t|z_{1:t-1}, x_{1:t}) = f_{t,\theta}(z_t|z_{1:t-1})$. Because we use the bootstrap proposal, gradients w.r.t. to θ are not backpropagated through q.

We use a fixed learning rate of 0.01 and optimize for 500 steps using SGA. Figure 4.2 shows that the convergence of both $\log p_\theta(x_{1:T})$ to $\max_\theta \log p_\theta(x_{1:T})$ and θ to $\text{argmax}_\theta \log p_\theta(x_{1:T})$ is faster when ELBO_{SMC} and more particles are used.

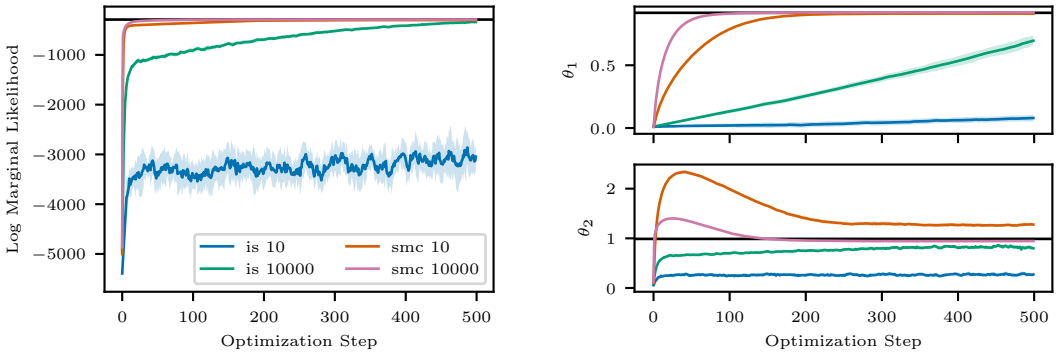


Figure 4.2: (Left) Log marginal likelihood analytically evaluated at every θ during optimization; the black line indicates $\max_\theta \log p_\theta(x_{1:T})$ obtained by the expectation maximization (EM) algorithm. (Right) learning of model parameters; the black line indicates $\text{argmax}_\theta \log p_\theta(x_{1:T})$ obtained by the EM algorithm.

4.2.4.2 Proposal Learning

We now investigate how learning ϕ, i.e. the proposal, is affected by the the choice of ELBO and the number of particles.

Consider a simple, fixed generative model

$$p(\mu)p(x|\mu) = \text{Normal}(\mu; 0, 1^2)\text{Normal}(x; \mu, 1^2),$$

where μ and x are the latent and observed variables respectively and a family of proposal distributions $q_\phi(\mu) = \text{Normal}(\mu; \mu_q, \sigma_q^2)$ parameterized by $\phi = (\mu_q, \log \sigma_q^2)$. For a fixed observation $x = 2.3$, we initialize $\phi = (0.01, 0.01)$ and optimize ELBO_{IS} with respect to ϕ. We investigate the quality of the learned parameter ϕ as we increase the number of particles K during training. Figure 4.3 (left) clearly demonstrates that the quality of ϕ compared to the analytic posterior decreases as we increase K.

Similar behavior is observed in Figure 4.3 (middle, right) where we optimize ELBO_{SMC} with respect to both θ and ϕ for the LGSSM described in Section 4.2.4.1. We see that using more particles helps model learning but makes proposal learning worse. Using our ALT algorithm alleviates this problem and at the same time makes model learning faster as it profits from a more accurate proposal distribution. We provide more extensive experiments exploring proposal learning with different ELBOs and number of particles in Appendix B.2.3.3.

4.2.4.3 Moving Agents

To show that our results are applicable to complex, high dimensional data we compare AESMC and IWAE on stochastic, partially observable video sequences. Figure B.3 in Appendix B.2.3.2 shows an example of such a sequence.

The dataset consists of $N = 5000$ sequences of images $(x_{1:T}^{(n)})_{n=1}^N$ of which 1000 are randomly held out as test set. Each sequence contains $T = 40$ images represented as a 2 dimensional array of size 32×32. In each sequence there is one agent, represented as circle, whose starting position is sampled randomly along the top and bottom of the image. The dataset is inspired by (Ondrúška and Posner, 2016), however with the

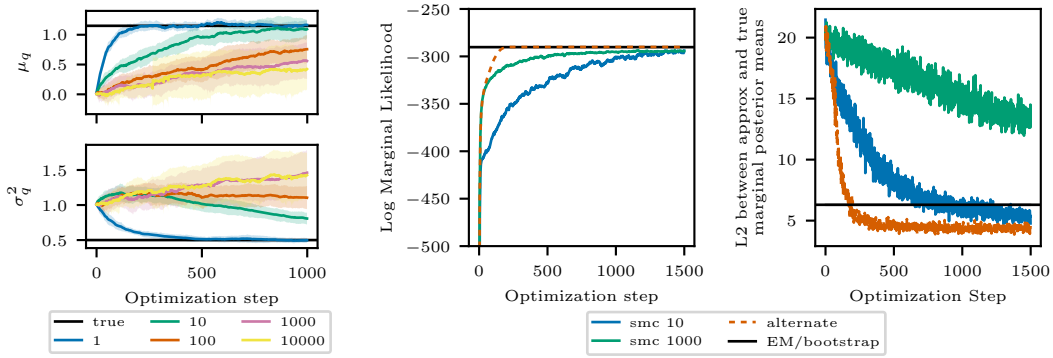


Figure 4.3: *(Left)* Optimizing ELBO_{IS} for the Gaussian unknown mean model with respect to ϕ results in worse ϕ as we increase number of particles K. *(Middle, right)* Optimizing ELBO_{SMC} with respect to (θ, ϕ) for LGSSM and using the ALT algorithm for updating (θ, ϕ) with $(A_\theta, K_\theta) = (\text{SMC}, 1000)$ and $(A_\phi, K_\phi) = (\text{IS}, 10)$. *Right* measures the quality of ϕ by showing $\sqrt{\sum_{t=1}^{T}(\mu_t^{\text{kalman}} - \mu_t^{\text{approx}})^2}$ where μ_t^{kalman} is the marginal mean obtained from the Kalman smoothing algorithm under the model with EM-optimized parameters and μ_t^{approx} is an marginal mean obtained from the set of 10 SMC particles with learned/bootstrap proposal.

crucial difference that the movement of the agent is *stochastic*. The agent performs a directed random walk through the image. At each timestep, it moves according to

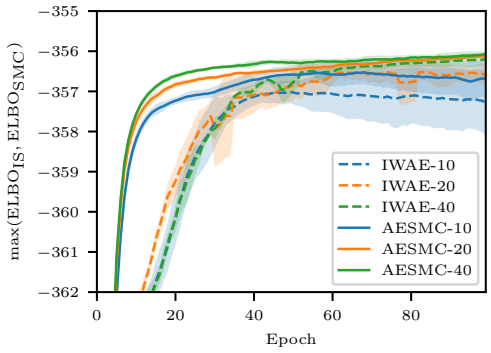
$$\begin{aligned} y_{t+1} &\sim \text{Normal}(y_{t+1}; y_t + 0.15, 0.02^2) \\ x_{t+1} &\sim \text{Normal}(x_{t+1}; 0, 0.02^2) \end{aligned} \tag{4.28}$$

where (x_t, y_t) are the coordinates in frame t in a unit square that is then projected onto 32×32 pixels. In addition to the stochasticity of the movement, half of the image is occluded, preventing the agent from being observed.

For the generative model and proposal distribution we use a variational recurrent neural network (VRNN) (Chung et al., 2015). It extends recurrent neural networks (RNNs) by introducing a stochastic latent state z_t at each timestep t. Together with the observation x_t, this state conditions the deterministic transition of the RNN. By introducing this unobserved stochastic state, the VRNN is able to better model complex long range variability in stochastic sequences. Architecture and hyperparameter details are given in Appendix B.2.3.1.

Figure 4.4 shows $\max(\text{ELBO}_{\text{IS}}, \text{ELBO}_{\text{SMC}})$ for models trained with IWAE and AESMC for different particle numbers. The lines correspond to the mean over three different random seeds and the shaded areas indicate the standard deviation.

The same number of particles was used for training and testing, additional hyper-parameter settings are given in the appendix. One can see that models trained using AESMC outperform IWAE and using more particles improves the ELBO for both. In Appendix B.2.3.2, we inspect different learned generative models by using them for prediction, confirming the results presented here. We also tested ALT on this task, but found that while it did occasionally improve performance, it was much less stable than IWAE and AESMC.



Particles	Method	Moving Agents
10	IWAE	-357.3
	AESMC	**-356.7**
20	IWAE	-356.6
	AESMC	**-356.1**
40	IWAE	-356.2
	AESMC	**-356.1**

Figure 4.4: *(Left)* Rolling mean over 5 epochs of $\max(\text{ELBO}_{\text{SMC}}, \text{ELBO}_{\text{IS}})$ on the test set, lines indicate the average over 3 random seeds and shaded areas indicate standard deviation. The color indicates the number of particles, the line style the used algorithm. *(Right)* The table shows the final $\max(\text{ELBO}_{\text{SMC}}, \text{ELBO}_{\text{IS}})$ for each learned model.

4.2.5 Discussion

We have developed AESMC—a method for performing model learning using a new ELBO objective which is based on the SMC marginal likelihood estimator. This ELBO objective is optimized using SGA and the reparameterization trick. Our approach utilizes the efficiency of SMC in models with intermediate observations and hence is suitable for highly structured models. We experimentally demonstrated that this objective leads to better generative model training than the IWAE objective for structured problems, due to the superior inference and tighter bound provided by using SMC instead of importance sampling.

Our work is most similar to works simultaneously developed by Maddison et al. (2017a) and Naesseth et al. (2018). Maddison et al. (2017a) additionally draw a connection between the variance of the marginal likelihood estimator and

tightness of the variational bound. This is a powerful theoretical result that can drive further research in marrying powerful inference algorithms and gradient-based model-learning algorithms. Naesseth et al. (2018) focus on the inference perspective. They view SMC as a way to enlarge the variational family. This allows for more expressive and thus accurate posterior approximations in variational inference. The loss they minimize ends up being the same as one proposed by us and Maddison et al. (2017a). They prove that this loss is in fact an upper bound on the KL divergence between a distribution obtained by running SMC and resampling at the last time-step and the posterior.

In addition to Maddison et al. (2017a); Naesseth et al. (2018), in Claim 4.3, we provide a simple way to express the bias of objectives induced by log of marginal likelihood estimators as a KL divergence on an extended space. In Propositions 4.4 and 4.5, we investigate the implications of these KLs being zero in the case of IWAE and AESMC. In addition to Maddison et al. (2017a), we prove the "only if" case of Proposition 4.5. We also study the properties of the proposal distribution as we increase number of particles during training. Using our assertion that tighter variational bounds are not necessarily better, we then introduce and test a new method, alternating ELBOs, that addresses some of these issues and observe that, in some cases, this improves both model and proposal learning. We discuss the issue of tighter variational bounds in more detail for the IWAE case in Rainforth et al. (2018a).

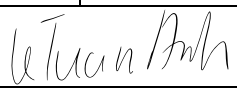
Statement of Authorship for joint/multi-authored papers for PGR thesis

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

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

Title of Paper	Auto-encoding sequential Monte Carlo
Publication Status	☑ Published □ Accepted for Publication □ Submitted for Publication □ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Tuan Anh Le, Maximilian Igl, Tom Rainforth, Tom Jin, and Frank Wood. Auto-encoding sequential Monte Carlo. In International Conference on Learning Representations, 2018a.

Student Confirmation

Student Name:	Tuan Anh Le		
Contribution to the Paper	The initial idea to differentiate through sequential Monte Carlo is FW's. TJ and I formalized the idea and wrote the first implementation, with MI joining the project shortly after. TR and I formalized AESMC and importance weighted auto-encoder as ELBOs on corresponding extended spaces. We also proved claims and propositions in the paper. I noticed that the performance of the proposal distribution worsens with increasing number of particles; TR formalized this in terms of signal-to-noise ratio. MI led the Moving Agents experiments as well as writing of the text describing it in Section 4.2.4.3. I led the other experiments. I led the development of the software framework with MI's help. I wrote the core of the paper with feedback and revisions from all other co-authors. TR led the writing of Section 4.2.2.3 and Section 4.2.3. In this section, I have modified the introduction and background to take into account preceding parts of the thesis. Additional derivations, proofs and experimental details for this section are in Appendix B.2 which has originally appeared as supplementary material of the conference paper.		
Signature		Date	27 March 2019

Supervisor Confirmation

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

Supervisor name and title: Dr Frank Wood		
Supervisor comments Tuan Anh's assertions lean towards too generous to others but are generally accurate.		
Signature 	Date	27/3/19

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

4.3 Revisiting Reweighted Wake-Sleep

In the previous section, we introduced a method, called AESMC, for gradient-based model-learning in SSMs that uses SMC as the underlying marginal likelihood estimator. AESMC is especially useful in models that can be formulated as SSMs as it uses intermediate observations to resample particles and hence direct computation to promising areas of the state-space. Probabilistic programs can be naturally formulated as SSMs and so AESMC is a step forward towards bridging the gap between unstructured models typically found in the deep generative modeling literature, like the sigmoid belief networks, and highly structured probabilistic programs.

There are two issues with AESMC that prevent it from being a fully-fledged solution for model learning and amortized inference in probabilistic programs. First, we require the latent variables to be reparameterizable in order to estimate gradients and to perform stochastic gradient descent (SGD). This means that we cannot have discrete latent variables which are often the key feature in probabilistic programs. Second, with increasing number of particles, the resulting proposal network (or inference network) gets worse. This means that although AESMC is a good method for model-learning, it is unsatisfactory for amortized inference.

In this section, we revisit RWS (Bornschein and Bengio, 2015) which has been introduced as an extension of the wake-sleep algorithm (Hinton et al., 1995; Dayan et al., 1995) in the context of deep generative models like sigmoid belief networks. Through extensive evaluations, show that it circumvents both of the above issues, outperforming current state-of-the-art methods in learning discrete latent-variable models. Moreover, we observe that, unlike the importance weighted auto-encoder, RWS learns better models *and* inference networks with increasing numbers of particles, and that its benefits extend to continuous latent-variable models as well. Our results suggest that RWS is a competitive, often preferable, alternative for learning deep generative models.

4.3.1 Learning Stochastic Control-Flow Models

Stochastic control-flow models (SCFMs) describe generative models that employ branching (i.e., the use of `if` / `else` / `cond` statements) on choices from discrete random variables. Recent years have seen such models gain relevance, particularly in the domain of deep probabilistic programming (Siddharth et al., 2017; Bingham et al., 2018; Tran et al., 2017; van de Meent et al., 2018, Ch. 7), which allows combining neural networks with generative models expressing arbitrarily complex control flow. SCFMs are encountered in a wide variety of tasks including tracking and prediction (Neiswanger et al., 2014; Kosiorek et al., 2018), clustering (Rasmussen, 2000), topic modeling (Blei et al., 2003), model structure learning (Adams et al., 2010), counting (Eslami et al., 2016), attention (Xu et al., 2015), differentiable data structures (Graves et al., 2014, 2016; Grefenstette et al., 2015), speech & language modeling (Juang and Rabiner, 1991; Chater and Manning, 2006), and concept learning (Kemp et al., 2006; Lake et al., 2018).

While a variety of approaches for amortized gradient-based learning (targeting model ELBO) exist for models using discrete random variables, the majority rely on continuous relaxations of the discrete variables (e.g. Rolfe, 2016; Vahdat et al., 2018b,a; van den Oord et al., 2017; Maddison et al., 2017b; Jang et al., 2017), enabling gradient computation through reparameterization (Kingma and Welling, 2014; Rezende et al., 2014). The models used by these approaches typically do not involve any control flow on discrete random choices, instead choosing to feed choices from their continuous relaxations directly into a neural network, thereby facilitating the required learning.

In contrast, SCFMs do not lend themselves to continuous-relaxation-based approaches due to the explicit branching requirement on choices from the discrete variables. Consider for example a simple SCFM—a two-mixture Gaussian mixture model (GMM). Computing the ELBO for this model involves choosing mixture identity (Bernoulli). Since a sample from a relaxed variable denotes a point on the surface of a probability simplex (e.g. [0.2, 0.8] for a Bernoulli random variable), instead of its vertices (0 or 1), computing the ELBO would need evaluation of both

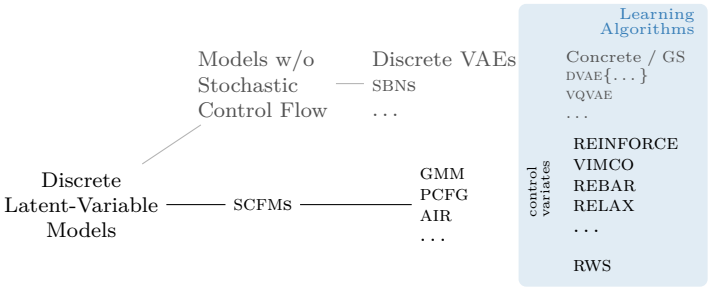


Figure 4.5: An overview of learning algorithms for discrete latent-variable models, with focus on SCFMs.

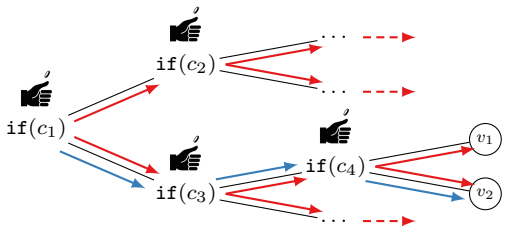


Figure 4.6: The challenge faced by continuous-relaxation methods on SCFMs—requiring exploration of all branches, in contrast to exploring only one branch at a time. Stochastic control flow proceeds through discrete choices (c_i) yielding values (v_i).

branches, weighting the resulting computation under each branch appropriately. This process can very quickly become intractable for more complex SCFMs, as it requires evaluation of *all* possible branches in the computation, of which there may be *exponentially* many, as illustrated in Fig. 4.6.

Alternatives to continuous-relaxation methods mainly involve the use of the IWAE (Burda et al., 2016) framework, employing the REINFORCE (Williams, 1992) gradient estimator, combined with control-variate schemes (Mnih and Gregor, 2014; Mnih and Rezende, 2016; Gu et al., 2016; Tucker et al., 2017; Grathwohl et al., 2018) to help decrease the variance of the naïve estimator. Although this approach ameliorates the problem with continuous relaxations in that it does not require evaluation of all branches, it has other drawbacks. Firstly, with more particles, the IWAE estimator adversely impacts inference-network quality, consequently impeding model learning (Rainforth et al., 2018a). Secondly, its practical efficacy can still be limited due to high variance and the requirement to design and optimize a separate neural network (c.f. Section 4.3.4.3).

Having characterized the class of models we are interested in (c.f. Fig. 4.5), and identified a range of current approaches (along with their characteristics) that might

apply to such models, we revisit RWS (Bornschein and Bengio, 2015). Comparing extensively with state-of-the-art methods for learning in SCFMs, we demonstrate its efficacy in learning better generative models and inference networks, using lower variance gradient estimators, over a range of computational budgets. To this end, we first review state-of-the-art methods for learning deep generative models with discrete latent variables (Section 4.3.2). We then revisit RWS (Section 4.3.3) and present an extensive evaluation of these methods (Section 4.3.4) on (i) a probabilistic context free grammar (PCFG) model on sentences, (ii) the Attend, Infer, Repeat (AIR) model (Eslami et al., 2016) to perceive and localize multiple MNIST digits, and (iii) a pedagogical GMM example that exposes a shortcoming of RWS which we then design a fix for. Our experiments confirm that RWS is a competitive, often preferable, alternative for learning SCFMs.

4.3.2 Background and Related Work

Consider data $(x^{(n)})_{n=1}^N$ sampled from a true (unknown) generative model $p(x)$, a family of generative models $p_\theta(z,x)$ of latent variable z and observation x parameterized by θ and a family of inference networks $q_\phi(z|x)$ parameterized by ϕ. We aim to learn the generative model by maximizing the marginal likelihood over data: $\theta^* = \text{argmax}_\theta \frac{1}{N}\sum_{n=1}^N \log p_\theta(x^{(n)})$. Simultaneously, we would like to learn an inference network $q_\phi(z|x)$ that amortizes inference given observation x; i.e., $q_\phi(z|x)$ maps an observation x to an approximation of $p_{\theta^*}(z|x)$. Amortization ensures this function evaluation is cheaper than performing approximate inference of $p_{\theta^*}(z|x)$ from scratch. Our focus here is on such joint learning of the generative model and the inference network, here referred to as "learning a deep generative model", although we note that other approaches exist that learn the generative model (Goodfellow et al., 2014a; Mohamed and Lakshminarayanan, 2016) or inference network (Paige and Wood, 2016; Le et al., 2017a) in isolation.

We begin with a review of IWAEs (Burda et al., 2016) as a general approach for learning deep generative models using SGD methods, focusing on generative-model families with discrete latent variables, for which the high variance of the

naïve gradient estimator impedes learning. We will also review control-variate and continuous-relaxation methods for gradient-variance reduction. The IWAE used alongside such gradient-variance reduction methods is currently the dominant approach for learning deep generative models with discrete latent variables.

4.3.2.1 Importance Weighted Autoencoders

Burda et al. (2016) introduce the IWAE, maximizing the mean ELBOs over data, $\frac{1}{N}\sum_{n=1}^{N}\text{ELBO}_{\text{IS}}^{K}(\theta,\phi,x^{(n)})$, where, for K particles,

$$\text{ELBO}_{\text{IS}}^{K}(\theta,\phi,x) = \mathbb{E}_{Q_\phi(z_{1:K}|x)}\left[\log\left(\frac{1}{K}\sum_{k=1}^{K} w_k\right)\right], \tag{4.29}$$
$$Q_\phi(z_{1:K}|x) = \prod_{k=1}^{K} q_\phi(z_k|x),\ w_k = \frac{p_\theta(z_k,x)}{q_\phi(z_k|x)}.$$

When $K=1$, this reduces to the VAE (Kingma and Welling, 2014; Rezende et al., 2014). Burda et al. (2016) show that $\text{ELBO}_{\text{IS}}^{K}(\theta,\phi,x)$ is a lower bound on $\log p_\theta(x)$ and that increasing K leads to a tighter lower bound. Further, tighter lower bounds arising from increasing K improve learning of the generative model, but impair learning of the inference network (Rainforth et al., 2018a), as the signal-to-noise ratio of θ's gradient estimator is $O(\sqrt{K})$ whereas ϕ's is $O(1/\sqrt{K})$. Note that although Tucker et al. (2019) solve this for reparameterizable distributions, the issue persists for discrete distributions. Consequently, poor learning of the inference network, beyond a certain point (large K), can actually impair learning of the generative model as well; a finding we explore in Section 4.3.4.3.

Optimizing the IWAE objective using SGD methods requires unbiased gradient estimators of $\text{ELBO}_{\text{IS}}^{K}(\theta,\phi,x)$ with respect to θ and ϕ (Robbins and Monro, 1951). The θ gradient $\nabla_\theta\,\text{ELBO}_{\text{IS}}^{K}(\theta,\phi,x)$ is estimated by sampling $z_{1:K}\sim Q_\phi(\cdot|x)$ and evaluating $\nabla_\theta\log\hat{Z}_K$, where $\hat{Z}_K=\frac{1}{K}\sum_{k=1}^{K}w_k$. While $\nabla_\phi\text{ELBO}_{\text{IS}}^{K}(\theta,\phi,x)$ is estimated similarly for models with reparameterizable latents, discrete (and other non-reparameterizable) latents require the REINFORCE gradient estimator (Williams, 1992)

$$g_{\text{REINFORCE}} = \underbrace{\log\hat{Z}_K\nabla_\phi\log Q_\phi(z_{1:K}|x)}_{①} + \underbrace{\nabla_\phi\log\hat{Z}_K}_{②}. \tag{4.30}$$

4.3.2.2 Continuous Relaxation and Control Variate Methods

Since the gradient estimator in (4.30) typically suffers from high variance, mainly due to the effect of ①, a number of approaches have been developed to ameliorate the issue. These can be broadly categorized into approaches that directly transform the discrete latent variables (continuous relaxations), or approaches that target improvement of the naïve REINFORCE estimator (control variates).

Continuous Relaxations. Here, discrete variables are transformed to enable reparameterization (Kingma and Welling, 2014; Rezende et al., 2014), which helps reduce the variance of the gradient estimators. Approaches range from the use of the Gumbel distribution (Maddison et al., 2017b; Jang et al., 2017), spike-and-X transforms (Rolfe, 2016), overlapping exponentials (Vahdat et al., 2018b), and generalized overlapping exponentials for tighter bounds (Vahdat et al., 2018a).

Besides difficulties inherent to such methods, such as tuning temperature parameters, or the suitability of undirected Boltzmann Machine priors, these methods are not well suited for learning SCFMs as they generate samples on the surface of a probability simplex rather than its vertices. For example, sampling from a transformed Bernoulli distribution yields samples of the form $[\alpha, (1-\alpha)]$ rather than simply 0 or 1—the latter form required for branching. With relaxed samples, as illustrated in Fig. 4.6, one would need to execute *all* the exponentially many discrete-variable driven branches in the model, weighting each branch appropriately—something that can quickly become infeasible for even moderately complex models. However, for purposes of comparison, for relatively simple SCFMs, one could apply methods involving continuous relaxations, as demonstrated in Section 4.3.4.3.

Control Variates. Here, approaches build on the REINFORCE estimator for the IWAE ELBO objective, designing control-variate schemes to reduce the variance of the naïve estimator. Variational inference for Monte Carlo objectives (VIMCO) (Mnih and Rezende, 2016) eschews designing an explicit control variate, instead exploiting

the particle set obtained in IWAE. It replaces ① with

$$g^{①}_{\text{VIMCO}} = \sum_{k=1}^{K} (\log \hat{Z}_K - \Upsilon_{-k}) \nabla_\phi \log q_\phi(z_k|x), \tag{4.31}$$
$$\Upsilon_{-k} = \log \frac{1}{K} \left(\exp\left(\frac{1}{K-1} \sum_{\ell \neq k} \log w_\ell \right) + \sum_{\ell \neq k} w_\ell \right)$$

where $\Upsilon_{-k} \perp\!\!\!\perp z_k$ and highly correlated with $\log \hat{Z}_K$.

Finally, assuming z_k is a discrete random variable with C categories[2], REBAR (Tucker et al., 2017) and RELAX (Grathwohl et al., 2018) improve on Mnih and Gregor (2014) and Gu et al. (2016), replacing ① with

$$g^{①}_{\text{RELAX}} = \left(\log \hat{Z}_K - c_\rho(\tilde{g}_{1:K}) \right) \nabla_\phi \log Q_\phi(z_{1:K}|x) + \nabla_\phi c_\rho(g_{1:K}) - \nabla_\phi c_\rho(\tilde{g}_{1:K}), \tag{4.32}$$

where g_k is a C-dimensional vector of reparameterized Gumbel random variates, z_k is a one-hot argmax function of g_k, and $\tilde{g}_k$ is a vector of reparameterized conditional Gumbel random variates conditioned on z_k. The conditional Gumbel random variates are a form of Rao-Blackwellization used to reduce variance. The control variate c_ρ, parameterized by ρ, is optimized to minimize the gradient variance estimates concurrently with the main ELBO optimization, leading to state-of-the-art performance on, for example, sigmoid belief networks (Neal, 1992). The main practical difficulty in using this method is choosing a suitable family of c_ρ, as some choices lead to higher variance despite the concurrent gradient variance minimization.

4.3.3 Revisiting Reweighted Wake-Sleep

Reweighted wake-sleep (RWS) (Bornschein and Bengio, 2015) comes from a family of algorithms (Hinton et al., 1995; Dayan et al., 1995) for learning deep generative models, eschewing a single objective over parameters θ and ϕ in favour of individual objectives for each. We review the RWS algorithm and discuss its pros and cons.

[2]The assumption is needed only for notational convenience. However, using more structured latents leads to difficulties in picking the control-variate architecture.

4.3.3.1 Reweighted Wake-Sleep

Reweighted wake-sleep (RWS) (Bornschein and Bengio, 2015) is an extension of the wake-sleep algorithm (Hinton et al., 1995; Dayan et al., 1995) both of which, like IWAE, jointly learn a generative model and an inference network given data. While IWAE targets a single objective, RWS alternates between objectives, updating the generative model parameters θ using a *wake-phase* θ *update* and the inference network parameters ϕ using either a *sleep-* or a *wake-phase* ϕ *update* (or both).

Wake-phase θ update. Given ϕ, θ is updated using an unbiased estimate of $\nabla_\theta\Big(\frac{1}{N}\sum_{n=1}^{N}\text{ELBO}_{\text{IS}}^{K}(\theta,\phi,x^{(n)})\Big)$, obtained without reparameterization or control variates, as the sampling distribution $Q_\phi(\cdot|x)$ is independent of θ.[3]

Sleep-phase ϕ update. Here, ϕ is updated to maximize the negative KL divergence between the posteriors under the generative model and the inference network, averaged over the data distribution of the current generative model

$$\mathbb{E}_{p_\theta(x)}[-\text{KL}\left(p_\theta(z|x)||q_\phi(z|x)\right)] = \mathbb{E}_{p_\theta(z,x)}[\log q_\phi(z|x) - \log p_\theta(z|x)]. \tag{4.33}$$

Its gradient, $\mathbb{E}_{p_\theta(z,x)}[\nabla_\phi \log q_\phi(z|x)]$, is estimated by evaluating $\nabla_\phi \log q_\phi(z|x)$, where $z, x \sim p_\theta(z,x)$. The estimator's variance can be reduced at a standard Monte Carlo rate by increasing the number of samples of z, x.

Wake-phase ϕ update. Here, ϕ is updated to maximize the negative KL divergence between the posteriors under the generative model and the inference network, averaged over the true data distribution

$$\mathbb{E}_{p(x)}[-\text{KL}\left(p_\theta(z|x)||q_\phi(z|x)\right)] = \mathbb{E}_{p(x)}[\mathbb{E}_{p_\theta(z|x)}[\log q_\phi(z|x) - \log p_\theta(z|x)]]. \tag{4.34}$$

The outer expectation $\mathbb{E}_{p(x)}[\mathbb{E}_{p_\theta(z|x)}[\nabla_\phi \log q_\phi(z|x)]]$ of the gradient is estimated using a single sample x from the true data distribution $p(x)$, given which, the inner

[3] We assume that the deterministic mappings induced by the parameters θ, ϕ are themselves differentiable, such that they are amenable to gradient-based learning.

expectation is estimated using self-normalized importance sampling with K particles, using $q_\phi(z|x)$ as the proposal distribution. This results in the following estimator

$$\sum_{k=1}^{K}\left(w_k/\sum_{\ell=1}^{K} w_\ell\right)\nabla_\phi \log q_\phi(z_k|x), \tag{4.35}$$

where, similar to (4.29), $x \sim p(x)$, $z_k \sim q_\phi(z_k|x)$, and $w_k = p_\theta(z_k, x)/q_\phi(z_k|x)$. Note that (4.35) is the negative of the second term of the REINFORCE estimator of the IWAE ELBO in (4.30). The crucial difference between the *wake-phase ϕ update* and the *sleep-phase ϕ update* is that the expectation in (4.34) is over the *true data distribution* $p(x)$ and the expectation in (4.33) is under the *current model distribution* $p_\theta(x)$. The former is desirable from the perspective of amortizing inference over data from $p(x)$, and although its estimator is biased, this bias decreases as K increases.

4.3.3.2 Advantages of Reweighted Wake-Sleep

While the gradient update of θ targets the same objective as IWAE, the gradient update of ϕ targets the objective in (4.33) in the sleep case and (4.34) in the wake case. This makes RWS a preferable option to IWAE for learning inference networks for the following reasons.

First, RWS leads to lower variance gradient estimators for ϕ, despite using control-variate methods to reduce variance of the REINFORCE estimator (c.f. Fig. 4.11).

Second, the ϕ updates in RWS directly target minimization of the expected KL divergences from the true to approximate posterior. With an increased computational budget, using more Monte Carlo samples in the *sleep-phase ϕ update* case and more particles K in the *wake-phase ϕ update*, we obtain a better estimator of these expected KL divergences. This is in contrast to IWAE, where optimizing $\text{ELBO}_{\text{IS}}^{K}$ targets a KL divergence on an extended sampling space (Le et al., 2018a) which for $K > 1$ doesn't correspond to a KL divergence between true and approximate posteriors (in any order). Consequently, increasing K in IWAE leads to impaired learning of inference networks (Rainforth et al., 2018a).

Third, targeting $\text{KL}\,(p||q)$ as in RWS can be preferable to targeting $\text{KL}\,(q||p)$ as in VAEs. The former encourages *mean-seeking behavior*, having the inference

network to put non-zero mass in regions where the posterior has non-zero mass, whereas the latter encourages *mode-seeking behavior*, having the inference network to put mass on one of the modes of the posterior (Minka et al., 2005). Using the inference network as an IS proposal requires mean-seeking behavior (Owen, 2013, Theorem 9.2). Moreover, Chatterjee et al. (2018) show that the number of particles required for IS to accurately approximate expectations of the form $\mathbb{E}_{p(z|x)}[f(z)]$ is directly related to $\exp(\mathrm{KL}\,(p||q)))$.

4.3.3.3 Disadvantages of Reweighted Wake-Sleep

While a common criticism of the wake-sleep family of algorithms is the lack of a unifying objective, we have not found any empirical evidence where this is a problem. Perhaps a more relevant criticism is that both the sleep and wake-phase ϕ gradient estimators are biased with respect to $\nabla_\phi \mathbb{E}_{p(x)}[\mathrm{KL}\,(p_\theta(z|x)||q_\phi(z|x))]$. The bias in the sleep-phase ϕ gradient estimator arises from targeting the expectation under the model rather than the true data distribution, and the bias in the wake-phase ϕ gradient estimator results from estimating the KL divergence using self-normalized IS.

In theory, these biases should not affect the fixed point of optimization (θ^*, ϕ^*) where $p_{\theta^*}(x) = p(x)$ and $q_{\phi^*}(z|x) = p_{\theta^*}(z|x)$. First, the data distribution bias should reduce to zero as $\theta \to \theta^*$ through the wake-phase θ update. Second, although the wake-phase ϕ gradient estimator is biased, it is consistent—with a large enough K, convergence of stochastic optimization is theoretically guaranteed on convex objectives and empirically on non-convex objectives (Chen and Luss, 2018). Moreover, this gradient estimator follows the central limit theorem and hence its asymptotic variance decreases linearly with K (Owen, 2013, Equation (9.8)). Thus, using larger K improves learning of the inference network.

In practice, the families of generative models, inference networks, and the data distributions determine which of the biases are more significant. In most of our findings, the bias of the data distribution appears to be the most detrimental. This is due to the fact that initially $p_\theta(x)$ is quite different from $p(x)$, and hence using sleep-phase ϕ updates performs worse than using wake-phase ϕ updates. An exception to

this is the PCFG experiment (c.f. Section 4.3.4.1) where the data distribution bias is not as large and inference using self-normalized IS is extremely difficult.

4.3.3.4 Is This Suitable for Probabilistic Programming?

Can this method for simultaneous model learning and amortized inference be implemented for a HOPPL? What is needed is that we can propose a value $z \sim q_\phi(z|x)$ from an inference network. For the wake-phase θ update, we require the gradient of $\log p_\theta(z, x)$ with respect to θ. This is doable as long as $\log p_\theta(z, x)$ is differentiable with respect to θ and the z sampled from q_ϕ can be matched to the forward generative model using an addressing scheme. For the sleep-phase ϕ update, we require sampling (z, x) from the generative model and evaluating the gradient of $\log q_\phi(z|x)$ with respect to ϕ. This can be done in a manner similar to the inference amortization method introduced in Section 3.2 which also requires an addressing scheme. For the wake-phase ϕ update, we require sampling $z \sim q_\phi(z|x)$ and evaluating (4.35). This requires evaluating $\log p_\theta(z, x)$, $\log q_\phi(z|x)$ and $\nabla_\phi q_\phi(z|x)$. This can also be done, provided there exists an addressing scheme that matches the samples $z \sim q_\phi(z|x)$ with the forward generative model and the inference network itself.

4.3.4 Experiments

The IWAE and RWS algorithms have primarily been applied to problems with continuous latent variables and/or discrete latent variables that do not actually induce branching (such as sigmoid belief networks; Neal (1992)). The purpose of the following experiments is to compare RWS to IWAE combined with control variates and continuous relaxations (c.f Section 4.3.3) on models with conditional branching, and show that it outperform such methods. We empirically demonstrate that increasing the number of particles K can be detrimental in IWAE but advantageous in RWS, as evidenced by achieved ELBOs and average distance between true and amortized posteriors.

In the first experiment, we present learning and amortized inference in a PCFG (Booth and Thompson, 1973) which is an example of a SCFM where continuous

relaxations are inapplicable. We demonstrate that RWS outperforms IWAE with a control variate both in terms of learning and inference.

The second experiment focuses on Attend, Infer, Repeat (AIR), the deep generative model of Eslami et al. (2016). It demonstrates that RWS leads to better learning of the generative model in a setting with both discrete and continuous latent variables, for modeling a complex visual data domain (c.f. Section 4.3.4.2). The final experiment involves a GMM (Section 4.3.4.3), thereby serving as a pedagogical example. It explains the causes of why RWS might be preferable to other methods in more detail.

Notationally, the different variants of RWS will be referred to as wake-sleep (WS) and wake-wake (WW). The *wake-phase* θ *update* is always used. We refer to using it in conjunction with the *sleep-phase* ϕ *update* as WS and using it in conjunction with the *wake-phase* ϕ *update* as WW.

We tried using both *wake-* and *sleep-phase* ϕ *updates*, but found that it doubles the required stochastic sampling, while yielding only minor improvements on the models we considered. The number of particles K used for the *wake-phase* θ and ϕ *updates* is always specified, and computation between them is matched so a *wake-phase* ϕ *update* with batch size B implies a *sleep phase* ϕ *update* with KB samples.

4.3.4.1 Probabilistic Context-Free Grammars

In this experiment we learn model parameters and amortize inference in a PCFG (Booth and Thompson, 1973). Each discrete latent variable in a PCFG chooses a particular child of a node in a tree. Depending on each discrete choice, the generative model can lead to different future latent variables. A PCFG is an example of an SCFM where continuous relaxations cannot be applied—weighing combinatorially many futures by a continuous relaxation is infeasible and doing so for futures which have infinite latent variables is impossible.

While supervised approaches have recently led to state-of-the-art performance in parsing (Chen and Manning, 2014), PCFGs remain one of the key models for unsupervised parsing (Manning et al., 1999). Learning in a PCFG is typically

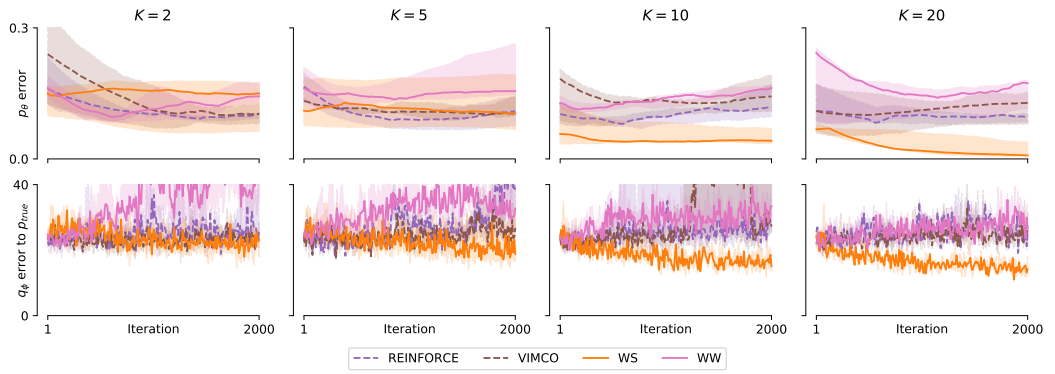


Figure 4.7: PCFG training. *(Top)* Quality of the generative model: While all methods have the same gradient update for θ, the performance of WS improves and is the best as K is increased. Other methods, including WW, do not yield significantly better model learning as K is increased, since WS's inference network learns the fastest. *(Bottom)* Quality of the inference network: VIMCO and REINFORCE do not improve with increasing K. WS performs best as K is increased, and while WW's performance improves, the improvement is not as significant. This can be attributed to the data-distribution bias being less significant than the bias coming from self-normalized IS (c.f. Section 4.3.3.3). Median and interquartile ranges from up to 10 repeats shown (see text).

done via expectation-maximization (Dempster et al., 1977) which uses the inside-outside algorithm (Lari and Young, 1990). Inference methods are based on dynamic programming (Younger, 1967; Earley, 1970) or search (Klein and Manning, 2003). Applying RWS and IWAE algorithms to PCFGs allows learning from large unlabeled datasets through SGD while inference amortization ensures linear-time parsing in the number of words in a sentence, at test-time. Moreover, using the inference network as a proposal distribution in IS provides asymptotically exact posteriors if parses are ambiguous.

A PCFG is defined by sets of terminals (or words) $\{t_i\}$, non-terminals $\{n_i\}$, production rules $\{n_i \to \zeta_j\}$ with ζ_j a sequence of terminals and non-terminals, probabilities for each production rule such that $\sum_j P(n_i \to \zeta_j) = 1$ for each n_i, and a start symbol n_1. Consider the *Astronomers* PCFG given in Manning et al. (1999, Table 11.2) (c.f. Appendix B.3.1). A parse tree z is obtained by recursively applying the production rules until there are no more non-terminals. For example, a parse tree (S (NP astronomers) (VP (V saw) (NP stars))) is obtained by applying the production rules as follows:

$$\mathrm{S} \xrightarrow{1.0} \mathrm{NP\,VP} \xrightarrow{0.1} \text{astronomers}\,\mathrm{VP} \xrightarrow{0.7} \text{astronomers}\,\mathrm{V\,NP}$$
$$\xrightarrow{1.0} \text{astronomers saw}\,\mathrm{NP} \xrightarrow{0.18} \text{astronomers saw stars},$$

where the probability $p(z)$ is obtained by multiplying the corresponding production probabilities as indicated on top of the arrows. The likelihood of a PCFG, $p(x|z)$, is 1 if the sentence x matches the sentence produced by z (in this case "astronomers saw stars") and 0 otherwise. One can easily construct infinitely long z by choosing productions which contain non-terminals, for example: $\mathrm{S} \to \mathrm{NP\,VP} \to \mathrm{NP\,PP\,VP} \to \mathrm{NP\,PP\,PP\,VP} \to \cdots$.

We learn the production probabilities of the PCFG and an inference network computing the conditional distribution of a parse tree given a sentence. The architecture of the inference network is the same as described in (Le et al., 2017a, Section 3.3) except the input to the RNN consists only of the sentence embedding, previous sample embedding, and an address embedding. Each word is represented as a one-hot vector and the sentence embedding is obtained through another RNN. Instead of a hard $\{0, 1\}$ likelihood which can make learning difficult, we use a relaxation, $p(x|z) = \exp(-L(x, s(z))^2)$, where L is the Levenshtein distance and $s(z)$ is the sentence produced by z. Using the Levenshtein distance can also be interpreted as an instance of approximate Bayesian computation (Sisson et al., 2018). Training sentences are obtained by sampling from the *astronomers* PCFG with the true production probabilities, discarding the corresponding parse trees.

We run WW, WS, VIMCO and REINFORCE ten times for $K \in \{2, 5, 10, 20\}$, with batch size $B = 2$, using the Adam optimizer (Kingma and Ba, 2015) with default hyperparameters. We observe that the inference network can often end up sub-optimally sampling very long z (by choosing production rules with many non-terminals), leading to slow and ineffective runs. We therefore cap the run-time to 100 hours—out of ten runs, WW, WS, VIMCO and REINFORCE retain on average 6, 6, 5.75 and 4 runs respectively. In Fig. 4.7, we show both (i) the quality of the generative model as measured by the average KL between the true and the model production probabilities, and (ii) the quality of the inference network as

measured by $\mathbb{E}_{p(x)}[\mathrm{KL}\left((||p)\left(z|x\right), q_\phi(z|x)\right)]$ which is estimated up to an additive constant (the conditional entropy $H(p(z|x))$) by the sleep-ϕ loss Eq. (4.33) using samples from the true PCFG.

Quantitatively, WS improves as K increases and outperforms IWAE-based algorithms both in terms of learning and inference amortization. While WW's inference amortization improves slightly as K increases, it is significantly worse than WS's. This is because IS proposals will rarely produce a parse tree z for which $s(z)$ matches x, leading to extremely biased estimates of the wake-ϕ update. In this case, this bias is more significant than that of the data-distribution which can harm the sleep-ϕ update. We inspect the quality of the inference network by sampling from it.

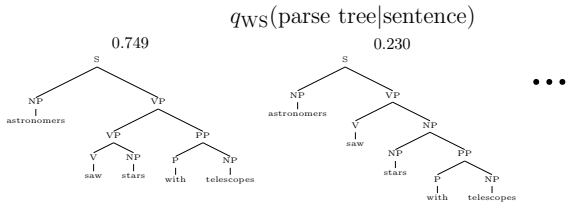


Figure 4.8: Samples from the inference network trained with WS ($K = 20$). Highest probability samples correspond to correct sentences ($s(z) = x$).

Figure 4.8, shows samples from an inference network trained with WS, conditioned on the sentence "astronomers saw stars with telescopes", weighted according to the frequency of occurrence. Appendix B.3.1 further includes samples from an inference network trained with VIMCO, showing that none of them match the given sentence ($s(z) \neq x$), and whose production probabilities are poor, unlike with RWS.

4.3.4.2 Attend, Infer, Repeat

Next, we evaluate WW and VIMCO on AIR (Eslami et al., 2016), a structured deep generative model with both discrete and continuous latent variables. AIR uses the discrete variable to decide how many continuous variables are necessary to explain an image. The sequential inference procedure of AIR poses a difficult problem, since it implies a sequential decision process with possible branching. See (Eslami et al., 2016) and Appendix B.3.2 for the model notation and details.

We set the maximum number of inference steps in AIR to three and train on 50×50 images with zero, one or two MNIST digits. The training and testing data sets consist of 60000 and 10000 images respectively, generated from the respective MNIST train/test datasets. Unlike AIR, which used Gaussian likelihood with fixed standard deviation and continuous inputs (i.e., input $\mathbf{x} \in [0,1]^{50\times 50}$), we use a Bernoulli likelihood and binarized data; the stochastic binarization is similar to Burda et al. (2016). Training is performed over two million iterations by RmsProp (Tieleman and Hinton, 2012) with the learning rate of 10^{-5}, which is divided by three after 400k and 1000k training iterations. We set the glimpse size to 20×20.

We first evaluate the generative model via the average test log marginal where each log marginal is estimated by a one-sample, 5000-particle IWAE estimate. The inference network is then evaluated via the average test KL from the inference network to the posterior under the current model where each $\mathrm{KL}\left(\left(||q\right)_\phi(z|x), p_\theta(z|x)\right)$ is estimated as a difference between the log marginal estimate above and a 5000-sample, one-particle IWAE estimate. Note that this estimate is just a proxy to the desired KL from the inference network to the *true* model posterior.

This experiment confirms (Fig. 4.9) that increasing number of particles improves VIMCO only upto a point, whereas WW improves monotonically with increased K. WW also results in significantly lower variance and better inference networks than VIMCO.

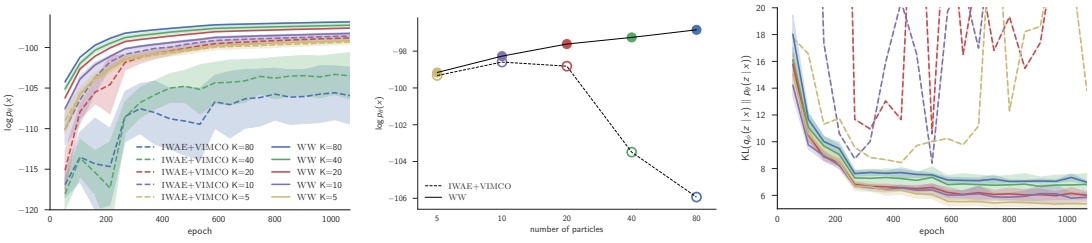


Figure 4.9: Training of AIR. *(Left)* Training curves: training with VIMCO leads to larger variance in training than WW. *(Middle)* Log evidence values at the end of training: increasing number of particles improves WW monotonically but improves VIMCO only up to a point ($K = 10$ is the best). *(Right)* WW results in significantly lower variance and better inference networks than VIMCO. Note that KL is between the inference network and the *current* generative model.

4.3.4.3 Gaussian Mixture Model

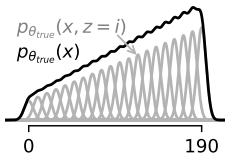


Figure 4.10: The GMM with true model parameters.

In order to examine the differences between WW and IWAE more closely, we study a GMM which branches on a discrete latent variable to select cluster assignments. The generative model and inference network are defined as

$$p_\theta(z) = \text{Categorical}(z|\text{softmax}(\theta)), \;\; p(x|z) = \text{Normal}(x|\mu_z, \sigma_z^2),$$
$$q_\phi(z|x) = \text{Categorical}(z|\text{softmax}(\eta_\phi(x))),$$

where $z \in \{0, \ldots, C-1\}$, C is the number of clusters and μ_c, σ_c^2 are fixed to $\mu_c = 10c$ and $\sigma_c^2 = 5^2$. The generative model parameters are $\theta \in \mathbb{R}^C$. The inference network consists of a multilayer perceptron $\eta_\phi : \mathbb{R} \to \mathbb{R}^C$, with the 1-16-$C$ architecture and the tanh nonlinearity, parameterized by ϕ. The chosen family of inference networks is empirically expressive enough to capture the posterior under the true model. The true model is set to $p_{\theta_{\text{true}}}(x)$ where $\text{softmax}(\theta_{\text{true}})_c = (c+5)/\sum_{i=1}^{C}(i+5)$ $(c = 0, \ldots, C-1)$, i.e. the mixture probabilities are linearly increasing with the z (Fig. 4.10). We fix the mixture parameters in order to study the important features of the problem at hand in isolation.

We train using WS, WW, as well as using IWAE with REINFORCE, RELAX, VIMCO and the Concrete distribution. We attempted variants of DVAE (Rolfe, 2016; Vahdat et al., 2018b) in this setting, but it performed considerably worse than any of the alternatives (c.f. Appendix B.3.4). We fix $C = 20$ and increase number of particles from $K = 2$ to 20. We use the Adam optimizer with the learning rate 10^{-3} and default β parameters. At each iteration, a batch of 100 data points is generated from the true model to train. Having searched over several temperature schedules for the Concrete distribution, we use the one with the lowest trainable

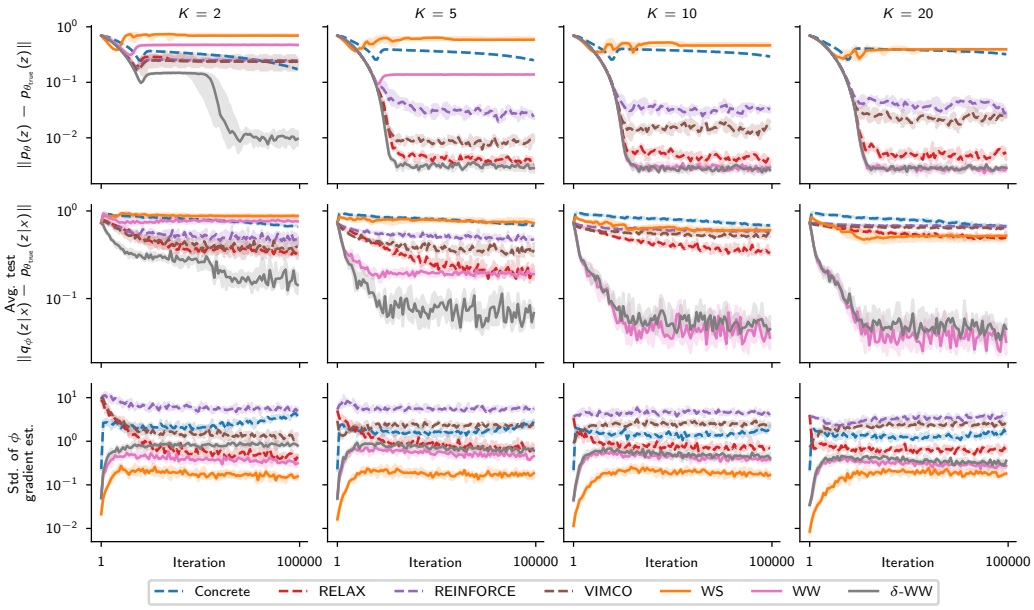


Figure 4.11: GMM training. Median and interquartile ranges from 10 repeats shown. *(Top)* Quality of the generative model: WS and WW improve with more particles thanks to lower variance and lower bias estimators of the gradient respectively. IWAE methods suffer with a larger particle budget (Rainforth et al., 2018a). WS performs the worst as a consequence of computing the expected KL under the model distribution $p_\theta(x)$ Eq. (4.33) instead of the true data distribution $p(x)$ as with WW Eq. (4.34). WW suffers from branch-pruning (see text) in low-particle regimes, but learns the best model fastest in the many-particle regime; δ-WW additionally learns well in the low-particle regime. *(Middle)* Both inference network and generative model quality develop identically. *(Bottom)* WW and WS have lower-variance gradient estimators of ϕ than IWAE, as they avoid the high-variance term ① in (4.30). This is a necessary, but not sufficient, condition for efficient learning, with other factors being gradient direction and the ability to escape local optima.

terminal temperature (linearly annealing from 3 to 0.5). We found that using the control variate $c_\rho(g_{1:K}) = \frac{1}{K}\sum_{k=1}^{K} \text{MLP}_\rho([x, g_k])$, where the architecture of the multilayer perceptron (MLP) is $(1 + C)$-16-16-1 (with tanh nonlinearity) led to most stable training (c.f. Appendix B.3.3).

We evaluate the generative model, inference network and the variance of the of ϕ. The generative model is evaluated via the L_2 distance between the probability mass functions (PMFs) of its prior and true prior as $\|\text{softmax}(\theta) - \text{softmax}(\theta_{\text{true}})\|$. The inference network is evaluated via the L_2 distance between PMFs of the current and true posteriors, averaged over a fixed set ($M = 100$) of observations $(x_{\text{test}}^{(m)})_{m=1}^{M}$ from the true model: $\frac{1}{M}\sum_{m=1}^{M} \|q_\phi(z|x_{\text{test}}^{(m)}) - p_{\theta_{\text{true}}}(z|x_{\text{test}}^{(m)})\|$. Finally, ϕ's gradient-

estimator standard deviation is given by $\frac{1}{D_\phi}\sum_{d=1}^{D_\phi}\text{std}(g_d)$ where g_d is the dth (out of D_ϕ) element of one of ϕ's gradient estimators (e.g. Eq. (4.30) for REINFORCE) and $\text{std}(\cdot)$ is estimated using 10 samples.

Here, we demonstrate that using WS and WW with larger particle budgets leads to a better inference networks whereas this is not the case for IWAE methods (Fig. 4.11, Middle). Recall that the former is because using more samples to estimate the gradient of the sleep ϕ objective Eq. (4.33) for WS reduces variance at a standard Monte Carlo rate and that using more particles in Eq. (4.35) to estimate the gradient of the wake ϕ objective results in a lower bias. The latter is because using more particles results in the signal-to-noise of IWAE's ϕ gradient estimator to drop at the rate $O(1/\sqrt{K})$ (Rainforth et al., 2018a).

Learning of the generative model, as a consequence of inference-network learning, is also better for WS and WW, but worse for IWAE methods with an increased particle budget. This is because the the θ gradient estimator (common to all methods), $\nabla_\theta \text{ELBO}_{\text{IS}}^K(\theta, \phi, x)$ can be seen as an importance sampling estimator whose quality is tied to the proposal distribution (inference network).

WW and WS have lower variance gradient estimators than IWAE, even if used with control-variate and continuous-relaxation methods. This is because ϕ's gradient estimators for WW and WS do not include the high-variance term ① in Eq. (4.30). This is a necessary but not sufficient condition for efficient learning with other important factors being gradient direction and the ability to escape local optima (explored below). Employing the Concrete distribution gives low-variance gradients for ϕ to begin with, but the model learns poorly due to the high gradients bias (due to high temperature hyperparameter).

We now describe a failure mode affecting WS, WW, VIMCO, RELAX and REINFORCE, which we call *branch-pruning*. It is best illustrated by inspecting the generative model and the inference network as training progresses, focusing on the low-particle ($K = 2$) regime (Fig. 4.12). For WS, the generative model $p_\theta(z)$ peaks at $z = 9$ and puts zero mass for $z > 9$; the inference network $q_\phi(z|x)$ becomes the posterior for this model which, here, has support at most $\{0, \ldots, 9\}$ for all x. This

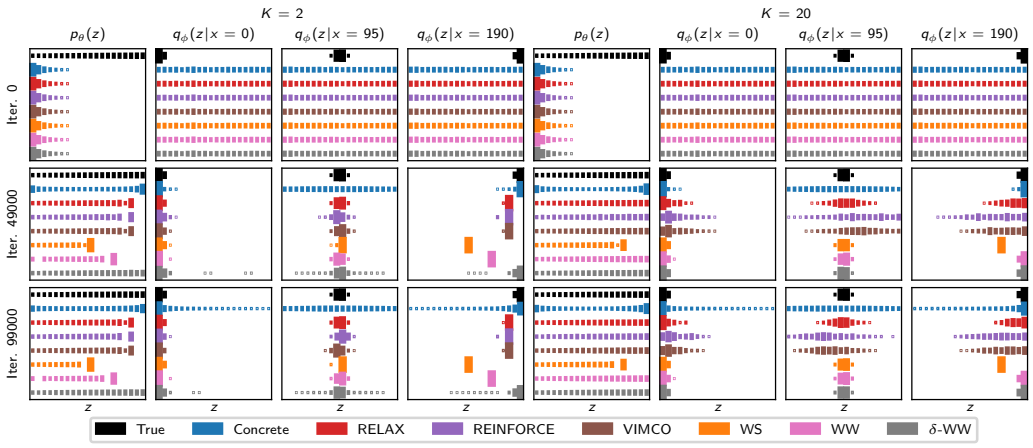


Figure 4.12: Generative model and inference network during GMM training shown as Hinton diagrams where areas are proportional to probability. Rows correspond to start, middle and end of optimization. *(Left half)* Learning with few particles leads to the branch-pruning (described in text) of the inference network (shown as conditional PMF given different x) and the generative model (first column of each half) for all methods except δ-WW. Concrete distribution fails. *(Right half)* Learning with many particles leads to branch-pruning only for WS; WW and δ-WW succeed where IWAE fails, learning a suboptimal final generative model.

is a local optimum for WS as (i) the inference network already approximates the posterior of the model $p_\theta(z, x)$ well, and (ii) the generative model $p_\theta(z)$, trained using samples from $q_\phi(z|x)$, has no samples outside of its current support. Similar failures occur for WW and VIMCO/RELAX/REINFORCE although the support of the locally optimal $p_\theta(z)$ is larger ($\{0, \ldots, 14\}$ and $\{0, \ldots, 17\}$ respectively).

While this failure mode is a particular feature of the GMM, we hypothesize that WS and WW suffer from it more, as they alternate between two different objectives for optimizing θ and ϕ. WS attempts to amortize inference for the current model distribution $p_\theta(x)$ which reinforces the coupling between the generative model and the inference network, making it easier to get stuck in a local optimum. WW with few particles (say $K = 1$) on the other hand, results in a highly-biased gradient estimator Eq. (4.35) that samples z from $q_\phi(\cdot|x)$ and evaluates $\nabla_\phi \log q_\phi(z|x)$; this encourages the inference network to concentrate mass. This behavior is not seen in WW with many particles where it is the best algorithm at learning both a good generative model and inference network (Fig. 4.11; Fig. 4.12, right).

We propose a simple extension of WW, denoted δ-WW, that mitigates this shortcoming by changing the proposal of the self-normalized importance sampling estimator in Eq. (4.35) to $q_{\phi,\delta}(z|x) = (1-\delta)q_\phi(z|x) + \delta\text{Uniform}(z)$. We use $\delta = 0.2$, noting that the method is robust to a range of values. Using a different proposal than the inference network $q_\phi(z|x)$ means that using the low-particle estimator in Eq. (4.35) no longer leads to branch-pruning. This is known as defensive importance sampling (Hesterberg, 1995), and is used to better estimate integrands that have long tails using short-tailed proposals. Using δ-WW outperforms all other algorithms in learning both the generative model and the inference network in the low-K regime and performs similarly as WW in the high-K regime.

4.3.5 Discussion

The central argument here is that where one needs both amortization and model learning for SCFMs, the RWS family of methods is preferable to IWAE with either continuous relaxations or control-variates. The PCFG experiment (Section 4.3.4.1) demonstrates a setting where continuous relaxations are inapplicable due to potentially infinite recursion, but where RWS applies and WS outperforms all other methods. The AIR experiment (Section 4.3.4.2) highlights a case where with more particles, performance of VIMCO degrades for the inference network (Rainforth et al., 2018a) and consequently the generative model as well, but where RWS's performance on both increases monotonically. Finally, the analysis on GMMs (Section 4.3.4.3) focuses on a simple model to understand nuances in the performances of different methods. Beyond implications from prior experiments, it indicates: (i) that gradient-estimator variance can be high despite using control-variates, and (ii) for the few-particle regime, the WW gradient estimator can be biased, leading to poor learning. For the latter, we design an alternative involving defensive sampling that ameliorates the issue. The precise choice of which variant of RWS to employ depends on which of the two kinds of gradient bias described in Section 4.3.3.3 dominates. Where the data distribution bias dominates, as with the AIR and GMM experiments, WW

is preferable, and where the self-normalised IS bias dominates, as in the PCFG experiment, WS is preferable.

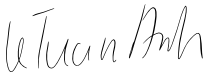
Statement of Authorship for joint/multi-authored papers for PGR thesis

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

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

Title of Paper	Revisiting reweighted wake-sleep
Publication Status	□Published □ Accepted for Publication ☑Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Tuan Anh Le, Adam R. Kosiorek, N. Siddharth, Yee Whye Teh, and Frank Wood. Revisiting reweighted wake-sleep. arXiv preprint 1805.10469, 2018b.

Student Confirmation

Student Name:	Tuan Anh Le		
Contribution to the Paper	The initial idea is mine. Its subsequent development is by FW, NS and me. ARK made the connection to reweighted wake-sleep. I led the PCFG and GMM experiments. ARK led the AIR and continuous MNIST (not included) experiments and writing of the corresponding sections. I wrote the paper with feedback from NS, ARK, FW and YWT. NS helped with figures in Section 4.3.1 and with editing of the full paper. In this section, I have modified the introduction and background to take into account preceding parts of the thesis. I have additionally written Section 4.3.3.4 which discusses the applicability of this method to probabilistic programming. Additional experimental details for this section are in Appendix B.3 which has originally appeared as supplementary material of the conference paper.		
Signature		Date	27 March 2019

Supervisor Confirmation

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

Supervisor name and title: Dr Frank Wood		
Supervisor comments Tuan Anh's assertions lean towards too generous to others but are generally accurate.		
Signature 	Date	27/3/19

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

4.4 Conclusions and Future Work

In this chapter, we have introduced AESMC as a method for learning models that is particularly suitable for SSMs. However, AESMC requires using reparameterizable latent variables and, additionally, with increasing number of particles the performance of the inference network deteriorates. Based on these drawbacks, we revisit RWS which is a simple but powerful method for simultaneously learning a model and amortizing inference in probabilistic models. RWS can be used in models with discrete latent variables and the performance of the inference network doesn't deteriorate with increasing number of particles. All we need is to align samples from the inference network $z \sim q_\phi(z|x)$ to the forward execution of the probabilistic program in order to evaluate the joint probability $p_\theta(z, x)$ and be able to differentiate it with respect to the model parameters θ. In the following, we outline future directions of research that can further improve model learning.

Learning realistic graphics models. Treating computer vision as graphics engine inversion lets us infer interpretable latent variables of scenes such as properties of objects and their relationships. However, the quality of inference crucially depends on the quality of the generative model, which consists of both the prior on latent variables and the graphics engine. For instance, the Captcha renderer must be realistic in order to infer correct character sequences from real Captcha images (Le et al., 2017a). Similarly, the hand-drawing renderer must be realistic in order to infer graphics programs from hand-drawn images (Ellis et al., 2018). In scene understanding, one must write an accurate model of scene generation which must account for, at the very least, the shape and texture of objects. Can we view learning such generative models as gradient-based maximum marginal likelihood and use algorithms such as RWS? How can we develop scene-understanding algorithms where low-level statistical regularities like texture are modeled using an unstructured latent variable while high-level features are modeled using a discrete program-like representation?

Concept learning through model structure learning. How do we learn uniquely human concepts like the natural numbers from core knowledge? Accounts from cognitive science (Barner and Baron, 2016) hypothesize that such conceptual change can occur, for example, from a productive combination of existing concepts, aided by language (Spelke, 2017) or from our ability to induce new "placeholder" symbols and derive their meanings by making analogies between existing concepts (Carey, 2011). In terms of probabilistic programming, this means that the structure of the program is unknown and the problem can be viewed as probabilistic program induction (Goodman et al., 2014) or equivalently inference of hidden causes (Pearl, 2009). In either case, existing gradient-based methods for model learning may not be applicable and an algorithmic solution may seem to require an extremely difficult search over a combinatorially large space. Yet, it seems that if we were to understand concept learning as inducing hidden causes or equivalently functions in a probabilistic program, it must transpire that such search must happen in some tractable way, possibly by restricting the search space. To give a concrete example of a problem that we can attempt to tackle, consider the MNIST dataset of hand-written digits. In a typical VAE, we learn a generative model of data consisting of a latent variable which is an uninterpretable feature vector and a conditional distribution of the image given the latent variable. An arguably more realistic model of data has two latent variables: one discrete latent variable corresponding to the digit identity and the other corresponding to the digit style. What are the mechanisms that can propose the second model from the first one, preferably in an unsupervised, or weakly-supervised setting? In other words, how do we suddenly know there are many distinct things where before there was only one?

Learning models for planning. Can we use the learned models to plan and take actions? The closest to what we mean is our work (Igl et al., 2018) on combining AESMC and decision-making in a partially observable Markov decision process (POMDP). Here, we learn a policy network by using model-free policy-gradient based loss together with an auxiliary loss based on the ELBO_{SMC} which serves to learn the

model. The policy network takes a POMDP belief, represented as an encoding of a set of weighted particles from running SMC. The latent space is an uninterpretable feature vector which serves more as an auto-encoding bottleneck, rather than something that could be used for planning. Given a potentially learned reward function, can we learn a policy that plans using the forward generative model?

5 Conclusion

We have focused on inference and learning methods in probabilistic programming systems. These allow us to express a rich class of probabilistic models as programs and perform inference in them by conditioning the random execution traces.

In Chapter 3, we proposed a method for performing amortized inference in probabilistic programs. The main idea is to train a neural network that maps from observed data to highly accurate approximations of posteriors which are then used as proposal distributions in an importance sampling (IS) inference engine. Once the neural network is trained, we can perform fast, repeated test-time inference in the specified probabilistic program.

In Chapter 4, we tackle the main drawback of the amortized inference approach which assumes that we have an accurate generative model. We propose an algorithm for model learning called auto-encoding sequential Monte Carlo (AESMC) which uses the strengths of sequential Monte Carlo (SMC) in order to improve learning of state-space models (SSMs) through maximum marginal likelihood. We also revisit reweighted wake-sleep (RWS) which addresses two drawbacks of AESMC. First, RWS allows performing simultaneous model learning and amortized inference in a way that the performance of the latter doesn't deteriorate with increasing computation as is the case with AESMC. Second, RWS is applicable to generative models which contain discrete latent variables, stochastic control flow and recursion. This makes

it suitable as a technique for simultaneous model learning and amortized inference in higher-order probabilistic programming languages (HOPPLs).

In Sections 3.3 and 4.4, we outline future research directions in amortized inference and model learning. These research directions are broad but their common thread is improving amortized inference and model learning, in addition to combining these with the decision-making element of Bayesian machine learning.

In amortized inference, the first direction is about automating the design of inference networks to make inference networks easier to train by reducing the number parameters. Secondly, given that the main workhorse behind current amortization algorithms is importance sampling, we want to explore whether amortizing more powerful inference algorithms can make amortized inference more effective in models with high-dimensional latent-variables. Lastly, we want to make computational models of intuitive psychology algorithmically possible by amortizing nested models.

In model learning, the first direction is about truly solving vision as inverse graphics. Current systems heavily rely on tuning graphics engines in order to make the generative model realistic. However, it isn't straightforward to automatically learn such models by applying gradient-based techniques described in this thesis or proposed in the deep generative modeling literature. How can we bridge this gap? The second direction is about learning model structure which seems to be essential part of automatic reasoning and something that, if we are to believe the Bayesian view of cognition (Tenenbaum et al., 2011), must somehow be algorithmically implemented. How can we do this, at least in domains where it seems that humans can do it?

Ultimately, we would like to contribute to model-based reinforcement learning and use learned generative models with amortized inference artifacts in order to plan and take actions.

Appendices

A

Proofs and Derivations for Inference Algorithms

A.1 Importance Sampling

A.1.1 Basic Importance Sampling

This section contains derivations for Section 2.3.1.2 on basic IS.

Properties of the estimator. The expectation of I_K^{IS} is equal to I

$$\begin{aligned}\mathbb{E}\left[I_K^{\text{IS}}\right] &= \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} w_k f(z_k)\right] = \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left[w_k f(z_k)\right] = \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left[\frac{\pi(z_k)}{q(z_k)}f(z_k)\right] \\ &= \frac{1}{K}\sum_{k=1}^{K}\int \frac{\pi(x)}{q(z)}f(z)q(z)\,\mathrm{d}z = \frac{1}{K}\sum_{k=1}^{K}\int \pi(z)f(z)\,\mathrm{d}z = I,\end{aligned}$$

and its variance can be expressed as

$$\begin{aligned}\operatorname{Var}\left[I_K^{\text{IS}}\right] &= \frac{1}{K^2}\sum_{k=1}^{K}\operatorname{Var}\left[\frac{\pi(z_k)}{q(z_k)}f(z_k)\right] \\ &= \frac{1}{K^2}\sum_{k=1}^{K}\left(\mathbb{E}\left[\left(\frac{\pi(z_k)}{q(z_k)}f(z_k)\right)^2\right] - \mathbb{E}\left[\frac{\pi(z_k)}{q(z_k)}f(z_k)\right]^2\right) \\ &= \frac{1}{K^2}\sum_{k=1}^{K}\left(\int\left(\frac{\pi(z)}{q(z)}f(z)\right)^2 q(z)\,\mathrm{d}z - I^2\right) \\ &= \frac{1}{K}\left(\int\left(\frac{\pi(z)}{q(z)}f(z)\right)^2 q(z)\,\mathrm{d}z - I^2\right) = \frac{1}{K}\left(\int\frac{\pi(z)^2 f(z)^2}{q(z)}\,\mathrm{d}z - I^2\right).\end{aligned}$$

An optimal proposal. One notion of an optimal proposal for IS is minimization of $\mathrm{Var}\left[I_K^{\mathrm{IS}}\right]$ for which the optimal proposal density, denoted q_{opt}, is

$$q_{\mathrm{opt}}(z) = \frac{|f(z)|\pi(z)}{\int |f(z')|\pi(z')\,\mathrm{d}z'}.$$

The variance of an estimator using an optimal proposal, $\mathrm{Var}\left[I_K^{\mathrm{IS,opt}}\right]$ is not greater than the variance of an estimator using an arbitrary proposal q, $\mathrm{Var}\left[I_K^{\mathrm{IS}}\right]$:

$$\begin{aligned}
\mathrm{Var}\left[I_K^{\mathrm{IS,opt}}\right] &= \frac{1}{K}\left(\int \frac{\pi(z)^2 f(z)^2}{q_{\mathrm{opt}}(z)}\,\mathrm{d}z - I^2\right) \\
&= \frac{1}{K}\left(\int \frac{\pi(z)^2 f(z)^2}{\frac{|f(z)|\pi(z)}{\int |f(z')|\pi(z')\,\mathrm{d}z'}}\,\mathrm{d}z - I^2\right) \\
&= \frac{1}{K}\left(\left(\int |f(z)|\pi(z)\,\mathrm{d}z\right)^2 - I^2\right) \\
&= \frac{1}{K}\left(\left(\int \frac{|f(z)|\pi(z)}{q(z)} q(z)\,\mathrm{d}z\right)^2 - I^2\right) \\
&\leq \frac{1}{K}\left(\int \left(\frac{|f(z)|\pi(z)}{q(z)}\right)^2 q(z)\,\mathrm{d}z - I^2\right) \\
&= \frac{1}{K}\left(\int \left(\frac{f(z)\pi(z)}{q(z)}\right)^2 q(z)\,\mathrm{d}z - I^2\right) \\
&= \mathrm{Var}\left[I_K^{\mathrm{IS}}\right],
\end{aligned}$$

where we have used the Jensen's inequality.

A.1.2 Self-Normalized Importance Sampling

Unbiasedness of Z_K^{IS} (equation (2.29)) can be established by noting that

$$\begin{aligned}
\mathbb{E}\left[Z_K^{\mathrm{IS}}\right] &= \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} w_k\right] = \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left[w_k\right] = \frac{1}{K}\sum_{k=1}^{K}\int q(z_k)\frac{\tilde{\pi}(z_k)}{q(z_k)}\,\mathrm{d}z_k \\
&= \int q(z)\frac{\tilde{\pi}(z)}{q(z)}\,\mathrm{d}z = Z\int \pi(z)\,\mathrm{d}z = Z.
\end{aligned}$$

A.2 Sequential Monte Carlo

A.2.1 Unbiasedness of the Normalizing Constant Estimator

This is a proof of unbiasedness of the SMC estimator $\hat{Z}_T$ of Z given in (2.45):

$$\hat{Z}_T = \prod_{t=1}^{T} \frac{1}{K} \sum_{k=1}^{K} w_t^k. \tag{A.1}$$

Let the sampling distribution of the SMC algorithm be:

$$Q(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) := \left(\prod_{k=1}^{K} q_1(z_1^k)\right) \left(\prod_{t=2}^{T} \prod_{k=1}^{K} q_t(z_t^k | \bar{z}_{1:t-1}^{a_{t-1}^k})\right) \left(\prod_{t=1}^{T-1} \prod_{k=1}^{K} \bar{w}_t^{a_t^k}\right). \tag{A.2}$$

We want to prove that

$$\mathbb{E}[\hat{Z}_T] = Z_T. \tag{A.3}$$

Let $b_T^k := k$ and $b_t^k := a_t^{b_{t+1}^k}$ $(t = T-1, \ldots, 1)$ so that kth particle set at the final time-step is $\bar{z}_{1:T}^k = (z_1^{b_1^k}, z_2^{b_2^k}, \ldots, z_T^{b_T^k})$. Define the prefix of this particle sequence as $z_{1:t}^{b_{1:t}^k} := (z_1^{b_1^k}, z_2^{b_2^k}, \ldots, z_t^{b_t^k})$. Define the proposal density of $\bar{z}_{1:T}^k$ as

$$q(\bar{z}_{1:T}^k) := \prod_{t=1}^{T} q_t(z_t^{b_t^k} | z_{1:t-1}^{b_{1:t-1}^k}). \tag{A.4}$$

Let $z_{1:T}^{-k} := z_{1:T}^{1:K} \setminus \bar{z}_{1:T}^k$ and $a_{1:T-1}^{-k} := a_{1:T-1}^{1:K} \setminus b_{1:T-1}$.

Let m be a discrete, $\{1, \ldots, K\}$-valued random variable and $P(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}, m)$ be an auxiliary distribution with the following density:

$$P(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}, m) = \pi_T(\bar{z}_{1:T}^m) \left(\frac{1}{K}\right)^T \frac{Q(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})}{q(\bar{z}_{1:T}^m) \left(\prod_{t=1}^{T-1} \bar{w}_t^{b_t^m}\right)}. \tag{A.5}$$

This is a valid density of $(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}, m)$ because the term in blue is a conditional density of $(z_{1:T}^{-m}, a_{1:T-1}^{-m})$ given $(\bar{z}_{1:T}^m, b_{1:T-1}^m)$ while $1/K^T$ is a uniform density of $b_{1:T}^m = (b_1^m, b_2^m, \ldots, b_{T-1}^m, b_T^m) = (b_{1:T-1}^m, m)$, and $\pi_T(\bar{z}_{1:T}^m)$ is a density of $\bar{z}_{1:T}^m$:

$$P(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}, m) = \pi_T(\bar{z}_{1:T}^m) \left(\frac{1}{K}\right)^T Q(z_{1:T}^{-m}, a_{1:T-1}^{-m} | \bar{z}_{1:T}^m, b_{1:T-1}^m).$$

We consider the ratio $P/Q\bar{w}_T^m$:

$$\begin{aligned}
\frac{P(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}, m)}{Q(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})\bar{w}_T^m} &= \frac{\pi_T(\bar{z}_{1:T}^m)\left(\frac{1}{K}\right)^T \frac{\cancel{Q(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})}}{q(\bar{z}_{1:T}^m)\left(\prod_{t=1}^{T-1}\bar{w}_t^{b_t^m}\right)}}{\cancel{Q(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})}\,\bar{w}_T^m} \\
&= \frac{\pi_T(\bar{z}_{1:T}^m)\left(\frac{1}{K}\right)^T}{q(\bar{z}_{1:T}^m)\left(\prod_{t=1}^{T}\bar{w}_t^{b_t^m}\right)} \\
&= \left(\prod_{t=1}^{T}\frac{1}{K}\sum_{k=1}^{K} w_t^k\right)\cdot\frac{\pi_T(\bar{z}_{1:T}^m)}{q(\bar{z}_{1:T}^m)\left(\prod_{t=1}^{T} w_t^{b_t^m}\right)} \\
&= \widehat{Z}_T \cdot \frac{\pi_T(\bar{z}_{1:T}^m)}{q(\bar{z}_{1:T}^m)\frac{\tilde{\pi}_T(\bar{z}_{1:T}^m)}{q(\bar{z}_{1:T}^m)}} \\
&= \widehat{Z}_T / Z_T. \qquad \text{(A.6)}
\end{aligned}$$

Finally,

$$\begin{aligned}
\mathbb{E}[\widehat{Z}_T] &= \int Q(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})\cdot\widehat{Z}_T\,\mathrm{d}z_{1:T}^{1:K}\,\mathrm{d}a_{1:T-1}^{1:K} \\
&= \int Q(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}, m)\bar{w}_T^m\cdot\widehat{Z}_T\,\mathrm{d}z_{1:T}^{1:K}\,\mathrm{d}a_{1:T-1}^{1:K}\,\mathrm{d}m \\
&= \int P(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}, m)\cdot\frac{Z_T}{\widehat{Z}_T}\cdot\widehat{Z}_T\,\mathrm{d}z_{1:T}^{1:K}\,\mathrm{d}a_{1:T-1}^{1:K}\,\mathrm{d}m \\
&= Z_T\int P(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}, m)\,\mathrm{d}z_{1:T}^{1:K}\,\mathrm{d}a_{1:T-1}^{1:K}\,\mathrm{d}m \\
&= Z_T. \qquad \text{(A.7)}
\end{aligned}$$

B

Appendices, Proofs, and Derivations for Model Learning

B.1 Proof of Theorem 4.1

Here is a proof of Theorem 4.1 on page 82.

Proposition 4.1. *Assuming that $\phi \in \Phi$ indexes the family of inference networks that contains all mappings from $\mathcal{X}$ to distributions on $\mathcal{Z}$, $\mathcal{Q}' = \{q : \mathcal{X} \to \mathcal{P}(\mathcal{Z})\}$, the solution to the maximization problem in* (4.5) *satisfies:*

$$\theta^* = \operatorname*{argmax}_{\theta} \mathbb{E}_{p(x)}[\log p_\theta(x)], \text{ and} \tag{4.6}$$

$$\phi^* = \operatorname*{argmin}_{\phi} \mathbb{E}_{p(x)} \left[\mathrm{KL}\left(q_\phi(z|x)||p_{\theta^*}(z|x)\right)\right]. \tag{4.7}$$

Proof. First, let us denote the objective function in (4.5):

$$\theta^*, \phi^* = \operatorname*{argmax}_{\theta,\phi} \left\{\mathbb{E}_{p(x)}[\log p_\theta(x)] - \mathbb{E}_{p(x)}[\mathrm{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right)]\right\}$$

by $f(\theta, \phi) = g(\theta) - h(\theta, \phi)$, where we define:

$$g(\theta) := \mathbb{E}_{p(x)}[\log p_\theta(x)] \text{ and}$$

$$h(\theta, \phi) := \mathbb{E}_{p(x)}[\mathrm{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right)].$$

We want to prove that:

$$\theta^* = \operatorname*{argmax}_{\theta} g(\theta), \text{ and} \tag{B.1}$$

$$\phi^* = \operatorname*{argmin}_{\phi} h(\theta^*, \phi). \tag{B.2}$$

To prove this by contradiction, assume that either

(i) $\theta^* \neq \operatorname{argmax}_\theta g(\theta)$ and $\phi^* = \operatorname{argmin}_\phi h(\operatorname{argmax}_\theta g(\theta), \phi)$, or

(ii) $\theta^* = \operatorname{argmax}_\theta g(\theta)$ and $\phi^* \neq \operatorname{argmin}_\phi h(\operatorname{argmax}_\theta g(\theta), \phi)$, or

(iii) $\theta^* \neq \operatorname{argmax}_\theta g(\theta)$ and $\phi^* \neq \operatorname{argmin}_\phi h(\operatorname{argmax}_\theta g(\theta), \phi)$.

For (i), let $\theta' = \operatorname{argmax}_\theta g(\theta) \neq \theta^*$, such that $g(\theta') > g(\theta^*)$. Since $\phi^* = \operatorname{argmin}_\phi h(\operatorname{argmax}_\theta g(\theta), \phi)$, we have $h(\theta', \phi^*) = 0$. Hence

$$\begin{aligned} f(\theta', \phi^*) &= g(\theta') - h(\theta', \phi^*) \\ &= g(\theta') \\ &> g(\theta^*) \\ &\geq g(\theta^*) - h(\theta^*, \phi^*) \\ &= f(\theta^*, \phi^*), \end{aligned}$$

where the penultimate step follows from the fact that a Kullback-Leibler (KL) divergence is non-negative.

For (ii), $\theta^* = \operatorname{argmax}_\theta g(\theta)$ and let $\phi' = \operatorname{argmin}_\phi h(\theta^*, \phi) \neq \phi^*$, such that $h(\theta^*, \phi') < h(\theta^*, \phi^*)$. Hence

$$\begin{aligned} f(\theta^*, \phi') &= g(\theta^*) - h(\theta^*, \phi') \\ &> g(\theta^*) - h(\theta^*, \phi^*) \\ &= f(\theta^*, \phi^*). \end{aligned}$$

For (iii), let $\theta' = \operatorname{argmax}_\theta g(\theta) \neq \theta^*$, such that $g(\theta') > g(\theta^*)$ and let $\phi' = \operatorname{argmin}_\phi h(\theta', \phi) \neq \phi^*$, such that $0 = h(\theta', \phi') < h(\theta', \phi^*)$. Hence

$$\begin{aligned} f(\theta', \phi') &= g(\theta') - h(\theta', \phi') \\ &= g(\theta') \\ &> g(\theta^*) \\ &\geq g(\theta^*) - h(\theta^*, \phi^*) \\ &= f(\theta^*, \phi^*), \end{aligned}$$

where the penultimate step follows from the fact that a KL divergence is non-negative.

In all cases, we have reached a contradiction, in which we could find a tuple different than (θ^*, ϕ^*) that attains a larger value for the objective in (4.5). □

B.2 Appendices for Auto-Encoding Sequential Monte Carlo

B.2.1 Gradients

The goal is to obtain an unbiased estimator for the gradient

$$\nabla_{\theta,\phi} \int Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \, \mathrm{d}z_{1:T}^{1:K} \, \mathrm{d}a_{1:T-1}^{1:K}. \tag{B.3}$$

B.2.1.1 Full Reinforce

We express the required quantity as

$$\begin{aligned} &\nabla_{\theta,\phi} \int Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \, \mathrm{d}z_{1:T}^{1:K} \, \mathrm{d}a_{1:T-1}^{1:K} && \text{(B.4)} \\ &= \int \nabla_{\theta,\phi} Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) + && \text{(B.5)} \\ &\qquad Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \nabla_{\theta,\phi} \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \, \mathrm{d}z_{1:T}^{1:K} \, \mathrm{d}a_{1:T-1}^{1:K} && \text{(B.6)} \\ &= \int Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \Big[\nabla_{\theta,\phi} \log Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) + && \text{(B.7)} \\ &\qquad \nabla_{\theta,\phi} \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \Big] \, \mathrm{d}z_{1:T}^{1:K} \, \mathrm{d}a_{1:T-1}^{1:K}, && \text{(B.8)} \end{aligned}$$

which we can estimate by sampling $(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$ directly from Q_{SMC} and evaluating $\nabla_{\theta,\phi} \log Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) + \nabla_{\theta,\phi} \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$.

B.2.1.2 Reinforce & Reparameterization

We express the required quantity as

$$\nabla_{\theta,\phi} \int Q_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \,\mathrm{d}z_{1:T}^{1:K} \,\mathrm{d}a_{1:T-1}^{1:K} \tag{B.9}$$

$$= \nabla_{\theta,\phi} \int \left(\prod_{k=1}^{K} q_1(z_1^k)\right) \left(\prod_{t=2}^{T}\prod_{k=1}^{K} q_t(z_t^k | z_{t-1}^{a_{t-1}^k}) \cdot \mathrm{Discrete}(a_{t-1}^k | w_{t-1}^{1:K})\right) \log \hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) \,\mathrm{d}z_{1:T}^{1:K} \,\mathrm{d}a_{1:T-1}^{1:K} \tag{B.10}$$

$$= \nabla_{\theta,\phi} \int \left(\prod_{k=1}^{K} s_1(\epsilon_1^k)\right) \left(\prod_{t=2}^{T}\prod_{k=1}^{K} s_t(\epsilon_t^k) \cdot \mathrm{Discrete}(a_{t-1}^k | w_{t-1}^{1:K})\right) \log \hat{Z}_{\mathrm{SMC}}(r(\epsilon_{1:T}^{1:K}), a_{1:T-1}^{1:K}) \,\mathrm{d}\epsilon_{1:T}^{1:K} \,\mathrm{d}a_{1:T-1}^{1:K} \tag{B.11}$$

$$= \int \left(\prod_{t=1}^{T}\prod_{k=1}^{K} s_t(\epsilon_t^k)\right) \left[\nabla_{\theta,\phi} \prod_{t=2}^{T}\prod_{k=1}^{K} \mathrm{Discrete}(a_{t-1}^k | w_{t-1}^{1:K}) \log \hat{Z}_{\mathrm{SMC}}(r(\epsilon_{1:T}^{1:K}), a_{1:T-1}^{1:K}) + \left(\prod_{t=2}^{T}\prod_{k=1}^{K} \mathrm{Discrete}(a_{t-1}^k | w_{t-1}^{1:K})\right) \nabla_{\theta,\phi} \log \hat{Z}_{\mathrm{SMC}}(r(\epsilon_{1:T}^{1:K}), a_{1:T-1}^{1:K})\right] \mathrm{d}\epsilon_{1:T}^{1:K} \,\mathrm{d}a_{1:T-1}^{1:K} \tag{B.12}$$

$$= \int \left(\prod_{t=1}^{T}\prod_{k=1}^{K} s_t(\epsilon_t^k)\right) \left(\prod_{t=2}^{T}\prod_{k=1}^{K} \mathrm{Discrete}(a_{t-1}^k | w_{t-1}^{1:K})\right) \cdot \left[\nabla_{\theta,\phi} \log \left(\prod_{t=2}^{T}\prod_{k=1}^{K} \mathrm{Discrete}(a_{t-1}^k | w_{t-1}^{1:K})\right) \log \hat{Z}_{\mathrm{SMC}}(r(\epsilon_{1:T}^{1:K}), a_{1:T-1}^{1:K}) + \nabla_{\theta,\phi} \log \hat{Z}_{\mathrm{SMC}}(r(\epsilon_{1:T}^{1:K}), a_{1:T-1}^{1:K})\right] \mathrm{d}\epsilon_{1:T}^{1:K} \,\mathrm{d}a_{1:T-1}^{1:K}, \tag{B.13}$$

where $r(\epsilon_{1:T}^{1:K})$ denotes a sample with identical distribution as $z_{1:T}^{1:K}$ obtained by passing the auxiliary samples $\epsilon_{1:T}^{1:K}$ through the reparameterization function. We can thus estimate the gradient by sampling $\epsilon_{1:T}^{1:K}$ from the auxiliary distribution, reparameterizing and evaluating

$$\nabla_{\theta,\phi} \log \left(\prod_{t=2}^{T}\prod_{k=1}^{K} \mathrm{Discrete}(a_{t-1}^k | w_{t-1}^{1:K})\right) \log \hat{Z}_{\mathrm{SMC}}(r(\epsilon_{1:T}^{1:K}), a_{1:T-1}^{1:K}) + \nabla_{\theta,\phi} \log \hat{Z}_{\mathrm{SMC}}(r(\epsilon_{1:T}^{1:K}), a_{1:T-1}^{1:K}).$$

In Figure B.1, we demonstrate that the estimator in (B.13) has much higher variance if we include the first term.

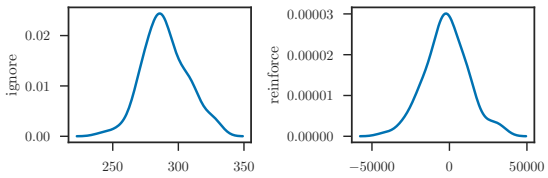


Figure B.1: $T = 200$ model described in Section 4.2.4.1. Kernel density estimation (KDE) of $\nabla_{\theta_1} \text{ELBO}_{\text{SMC}}$ evaluated at $\theta_1 = 0.1$ with $K = 16$ using 100 samples.

B.2.2 Proofs for Bias & Implications on the Proposals

Derivation of (4.17).

$$\begin{aligned} \text{ELBO} &= \int Q(z) \log \frac{Z_P P(z)}{Q(z)} \, \mathrm{d}z && \text{(B.14)} \\ &= \int Q(z) \log Z_P \, \mathrm{d}z - \int Q(z) \log \frac{Q(z)}{P(z)} \, \mathrm{d}z && \text{(B.15)} \\ &= \log Z_P - \text{KL}\left(Q || P\right). && \text{(B.16)} \end{aligned}$$

□

Proof of Claim 4.3. Since $\hat{Z}_P(z) \geq 0$, $Q(z) \geq 0$ and $\int Q(z)\hat{Z}_P(z) \, \mathrm{d}x = Z_P$, we can let the unnormalized target density in Definition 4.2 be $\tilde{P}(z) = Q(z)\hat{Z}_P(z)$. Hence, the normalized target density is $P(z) = Q(z)\hat{Z}_P(z)/Z_P$. Substituting these quantities into (4.16) and (4.17) yields the two equalities in (4.18). □

Proof of Proposition 4.4. ($\Longrightarrow$) Substituting for $Q_{\text{IS}}(z^{1:K}) = P_{\text{IS}}(z^{1:K})$, we obtain

$$\begin{aligned} \prod_{k=1}^{K} q(z^k|x) &= \frac{1}{K} \sum_{k=1}^{K} \frac{\prod_{\ell=1}^{K} q(z^\ell|x)}{q(z^k|x)} p(z^k|x) && \text{(B.17)} \\ &= \frac{1}{K} \sum_{k=1}^{K} \left[q(z^1|x) \cdots q(z^{k-1}|x) p(z^k|x) q(z^{k+1}|x) \cdots q(z^K|x) \right]. && \text{(B.18)} \end{aligned}$$

Integrating both sides with respect to $(x^2, \ldots, x^K)$ over the whole support (i.e. marginalizing out everything except x^1), we obtain:

$$q(z^1|x) = \frac{1}{K} \left[p(z^1|x) + \sum_{k=2}^{K} q(z^1|x) \right]. \tag{B.19}$$

Rearranging gives us $q(z^1|x) = p(z^1|x)$ for all x^1.

($\Longleftarrow$) Substituting $p(z^k|x) = q(z^k|x)$, we obtain

$$P_{\mathrm{IS}}(z^{1:K}) = \frac{1}{K}\sum_{k=1}^{K}\frac{Q_{\mathrm{IS}}(z^{1:K})}{q(z^k|x)}p(z^k|x) \tag{B.20}$$

$$= \frac{1}{K}\sum_{k=1}^{K}Q_{\mathrm{IS}}(z^{1:K}) \tag{B.21}$$

$$= Q_{\mathrm{IS}}(z^{1:K}). \tag{B.22}$$

□

Proof of Proposition 4.5. We consider the general sequence of target distributions $\pi_t(z_{1:t})$ ($p_\theta(z_{1:t}|x_{1:t})$ in the case of SSMs), their unnormalized versions $\gamma_t(z_{1:t})$ ($p_\theta(z_{1:t}, x_{1:t})$ in the case of SSMs), their normalizing constants $Z_t = \int \gamma_t(z_{1:t})\,\mathrm{d}z_{1:t}$ ($p_\theta(x_{1:t})$ in the case of SSMs), where $Z = Z_T = p(x_{1:T})$.

($\Longrightarrow$) It suffices to show that $\hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) = Z$ for all $(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$ implies 1 and 2 in Proposal 4.5 due to equation (4.19).

We first prove that $\hat{Z}_{\mathrm{SMC}}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) = Z$ for all $(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$ implies that the weights

$$w_1(z_1) := \frac{\gamma_1(z_1)}{q_1(z_1)} \tag{B.23}$$

$$w_t(z_{1:t}) := \frac{\gamma_t(z_{1:t})}{\gamma_{t-1}(z_{1:t-1})q_t(z_t|z_{1:t-1})} \qquad \text{for } t = 2, \ldots, T \tag{B.24}$$

are constant with respect to $z_{1:t}$.

Pick $t \in \{1, \ldots, T\}$ and distinct $k, \ell \in \{1, \ldots, K\}$. Also, pick $z_{1:t}$ and $x'_{1:t}$. Now, consider two sets of particle sets $(\bar{z}_{1:T}^{1:K}, \bar{a}_{1:T-1}^{1:K})$ and $(\tilde{z}_{1:T}^{1:K}, \tilde{a}_{1:T-1}^{1:K})$, illustrated

in Figure B.2, such that

$$\bar{z}_\tau^\kappa = \begin{cases} {x'}_\tau & \text{if } \kappa = \ell \text{ and } \tau < t \\ {x'}_\tau & \text{if } (\kappa, \tau) = (k, t) \\ z_\tau & \text{if } \kappa = k \text{ and } \tau < t \\ z_\tau^\kappa & \text{otherwise} \end{cases} \quad \text{for } \tau = 1, \ldots, T,\ \kappa = 1, \ldots, K, \tag{B.25}$$

$$\bar{a}_\tau^\kappa = \begin{cases} \ell & \text{if } (\kappa, \tau) = (k, t-1) \text{ or } (k, t) \\ \kappa & \text{otherwise} \end{cases} \quad \text{for } \tau = 1, \ldots, T-1,\ \kappa = 1, \ldots, K, \tag{B.26}$$

$$\tilde{z}_\tau^\kappa = \begin{cases} {x'}_\tau & \text{if } \kappa = \ell \text{ and } \tau < t \\ z_\tau & \text{if } (\kappa, \tau) = (k, t) \\ z_\tau & \text{if } \kappa = k \text{ and } \tau < t \\ z_\tau^\kappa & \text{otherwise} \end{cases} \quad \text{for } \tau = 1, \ldots, T,\ \kappa = 1, \ldots, K, \tag{B.27}$$

$$\tilde{a}_\tau^\kappa = \begin{cases} \ell & \text{if } (\kappa, \tau) = (k, t) \\ \kappa & \text{otherwise} \end{cases} \quad \text{for } \tau = 1, \ldots, T-1,\ \kappa = 1, \ldots, K. \tag{B.28}$$

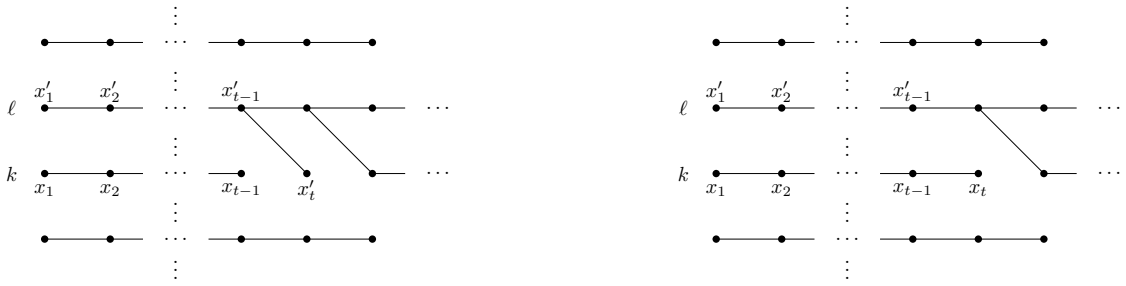


Figure B.2: (Left) particle set $(\bar{z}_{1:T}^{1:K}, \bar{a}_{1:T-1}^{1:K})$ and (right) particle set $(\tilde{z}_{1:T}^{1:K}, \tilde{a}_{1:T-1}^{1:K})$. Lines indicate ancestor indices.

The weights $\bar{w}_\tau^\kappa$ and $\tilde{w}_\tau^\kappa$ for the respective particle sets are identical except when $(\tau, \kappa) = (t, k)$ where

$$\bar{w}_t^k = w_t({x'}_{1:t}), \tag{B.29}$$

$$\tilde{w}_t^k = w_t(z_{1:t}). \tag{B.30}$$

Since $\hat{Z}(\bar{z}_{1:T}^{1:K}, \bar{a}_{1:T-1}^{1:K}) = \hat{Z}(\tilde{z}_{1:T}^{1:K}, \tilde{a}_{1:T-1}^{1:K})$, we have $w_t({x'}_{1:t}) = w_t(z_{1:t})$. As this holds for any arbitrary t and $z_{1:t}$, it follows that $w_t(z_{1:t})$ must be constant with respect to $z_{1:t}$ for all $t = 1, \ldots, T$.

Now, for $z_{1:t}$, consider the implied proposal by rearranging (B.23) and (B.24)

$$q_1(z_1) = \frac{\gamma_1(z_1)}{w_1} \tag{B.31}$$

$$q_t(z_t|z_{1:t-1}) = \frac{\gamma_t(z_{1:t})}{\gamma_{t-1}(z_{1:t-1})w_t} \qquad \text{for } t = 2, \ldots, T, \tag{B.32}$$

where $w_t := w_t(z_{1:t})$ is constant from our previous results. For this to be a normalized density with respect to z_t, we must have

$$w_1 = \int \gamma_1(z_1)\,\mathrm{d}z_1 = Z_1, \tag{B.33}$$

and for $t = 2, \ldots, T$:

$$w_t = \int \frac{\gamma_t(z_{1:t})}{\gamma_{t-1}(z_{1:t-1})}\,\mathrm{d}z_t \tag{B.34}$$

$$= \frac{\int \gamma_t(z_{1:t})\,\mathrm{d}z_t}{\gamma_{t-1}(z_{1:t-1})} \tag{B.35}$$

$$= \frac{Z_t}{Z_{t-1}} \cdot \frac{\int \pi_t(z_{1:t})\,\mathrm{d}z_t}{\pi_{t-1}(z_{1:t-1})}. \tag{B.36}$$

Since $\int \pi_{t+1}(z_{1:t+1})\,\mathrm{d}z_{t+1}$ and $\pi_t(z_{1:t})$ are both normalized densities, we must have $\pi_t(z_{1:t}) = \int \pi_{t+1}(z_{1:t+1})\,\mathrm{d}z_{t+1}$ for all $t = 1, \ldots, T-1$ for all $z_{1:t}$. For a given $t \in \{1, \ldots, T-1\}$ and $z_{1:t}$, applying this repeatedly yields

$$\pi_t(z_{1:t}) = \int \pi_{t+1}(z_{1:t+1})\,\mathrm{d}z_{t+1} = \int\int \pi_{t+2}(z_{1:t+2})\,\mathrm{d}z_{t+2}\,\mathrm{d}z_{t+1} = \cdots = \int \pi_T(z_{1:T})\,\mathrm{d}z_{t+1:T} \tag{B.37}$$

such that each $\pi_t(z_{1:t})$ must be the corresponding marginal of the final target. We also have

$$w_1(z_1) = Z_1, \tag{B.38}$$

$$w_t(z_{1:t}) = \frac{Z_t}{Z_{t-1}}, \qquad t = 2, \ldots, T, \tag{B.39}$$

$$q_1(z_1) = \pi_1(z_1) = \pi_T(z_1), \tag{B.40}$$

$$q_t(z_t|z_{1:t-1}) = \frac{\pi_t(z_{1:t})}{\pi_{t-1}(z_{1:t-1})} = \frac{\pi_T(z_{1:t})}{\pi_T(z_{1:t-1})}, \qquad t = 2, \ldots, T. \tag{B.41}$$

($\Longleftarrow$) To complete the proof, we now simply substitute identities in 1 and 2 of Proposal 4.5 back to the expression of $\hat{Z}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K})$ to obtain $\hat{Z}(z_{1:T}^{1:K}, a_{1:T-1}^{1:K}) = Z$. □

B.2.3 Experiments

B.2.3.1 VRNN

In the following we give the details of our variational recurrent neural network (VRNN) architecture. The generative model is given by:

$$p(z_{1:T}, h_{0:T}, x_{1:T}) = p(h_0) \prod_t p(z_t|h_{t-1}) p(x_t|h_{t-1}, z_t) p(h_t|h_{t-1}, z_t, x_t) \tag{B.42}$$

where

$$\begin{aligned} p(h_0) &= \text{Normal}(h_0; 0, I) \\ p(z_t|h_{t-1}) &= \text{Normal}(z_t; \mu_\theta^z(h_{t-1}), \sigma_\theta^z(h_{t-1})^2) \\ p(x_t|h_{t-1}, z_t) &= \text{Bernoulli}(x_t; \mu_\theta^x(\varphi_\theta^z(z_t), h_{t-1})) \\ p(h_t|h_{t-1}, z_t, x_t) &= \delta_{f(h_{t-1}, \varphi_\theta^z(z_t), \varphi_\theta^x(x_t))}(h_t) \end{aligned} \tag{B.43}$$

and the proposal distribution is given by

$$p(z_t|x_t, h_{t-1}) = \text{Normal}(z_t; \mu_\phi^p(\varphi_\phi^x(x_t), h_{t-1}), \sigma_\phi^{p2}(\varphi_\phi^x(x_t), h_{t-1})) \tag{B.44}$$

The functions μ_θ^z and σ_θ^z are computed by networks with two fully connected layers of size 128 whose first layer is shared. φ_θ^z is one fully connected layer of size 128.

For visual input, the encoding φ_θ^x is a convolutional network with conv-4x4-2-1-32, conv-4x4-2-1-64, conv-4x4-2-1-128 where conv-wxh-s-p-n denotes a convolutional network with n filters of size $w \times h$, stride s, padding p. Between convolutions we use leaky ReLUs with slope 0.2 as nonlinearity and batch norms. The decoding μ_θ^x uses transposed convolutions of the same dimensions but in reversed order, however with stride $s = 1$ and padding $p = 0$ for the first layer.

A Gated Recurrent Unit (GRU) is used as RNN and if not stated otherwise ReLUs are used in between fully connected layers.

For the proposal distribution, the functions μ_ϕ^p and σ_ϕ^p are neural networks with three fully connected layers of size 128 that are sharing the first two layers. Sigmoid and softplus functions are used where values in $(0, 1)$ or $\mathbb{R}^+$ are required. We use a minibatch size of 25.

For the moving agents dataset we use ADAM (ADAM) with a learning rate of 10^{-3}.

A specific feature of the VRNN architecture is that the proposal and the generative model share the component $\varphi^x_{\phi,\theta}$. Consequently, we set $\phi = \theta$ for the parameters belonging to this module and train it using gradients for both θ and ϕ.

B.2.3.2 Moving Agents

In Figure B.3 we investigate the quality of the generative model by comparing visual predictions. We do so for models learned by importance weighted auto-encoder (IWAE) *(top)* and AESMC *(bottom)*. The models were learned using ten particles but for easier visualization we only predict using five particles.

The first row in each graphic shows the ground truth. The second row shows the averaged predictions of all five particles. The next five rows show the predictions made by each particle individually.

The observations (i.e. the top row) up to $t = 19$ are shown to the model. Up to this timestep the latent values $z_{0:19}$ are drawn from the proposal distribution $q(z_t|x_t, h_{t-1})$. From $t = 20$ onwards the latent values $z_{20:37}$ are drawn from the generative model $p(z_t|z_{t-1})$. Consequently, the model predicts the partially occluded, stochastic movement over 17 timesteps into the future.

We note that most particles predict a viable future trajectory. However, the model learned by IWAE is not as consistent in the quality of its predictions, often 'forgetting' the particle. This does not happen in every predicted sequence but the behavior shown here is very typical. Models learned by AESMC are much more consistent in the quality of their predictions.

B.2.3.3 Optimizing Only Proposal Parameters

We have run experiments where we optimize various ELBO objectives with respect to ϕ with θ fixed in order to see how various objectives have an effect on proposal learning. In particular, we train ELBO_{IS} and ELBO_{SMC} with number of particles $K \in \{10, 100, 1000\}$. Once the training is done, we use the trained proposal network to perform inference using both IS and SMC with number of particles $K_{\text{test}} \in \{10, 100, 1000\}$.

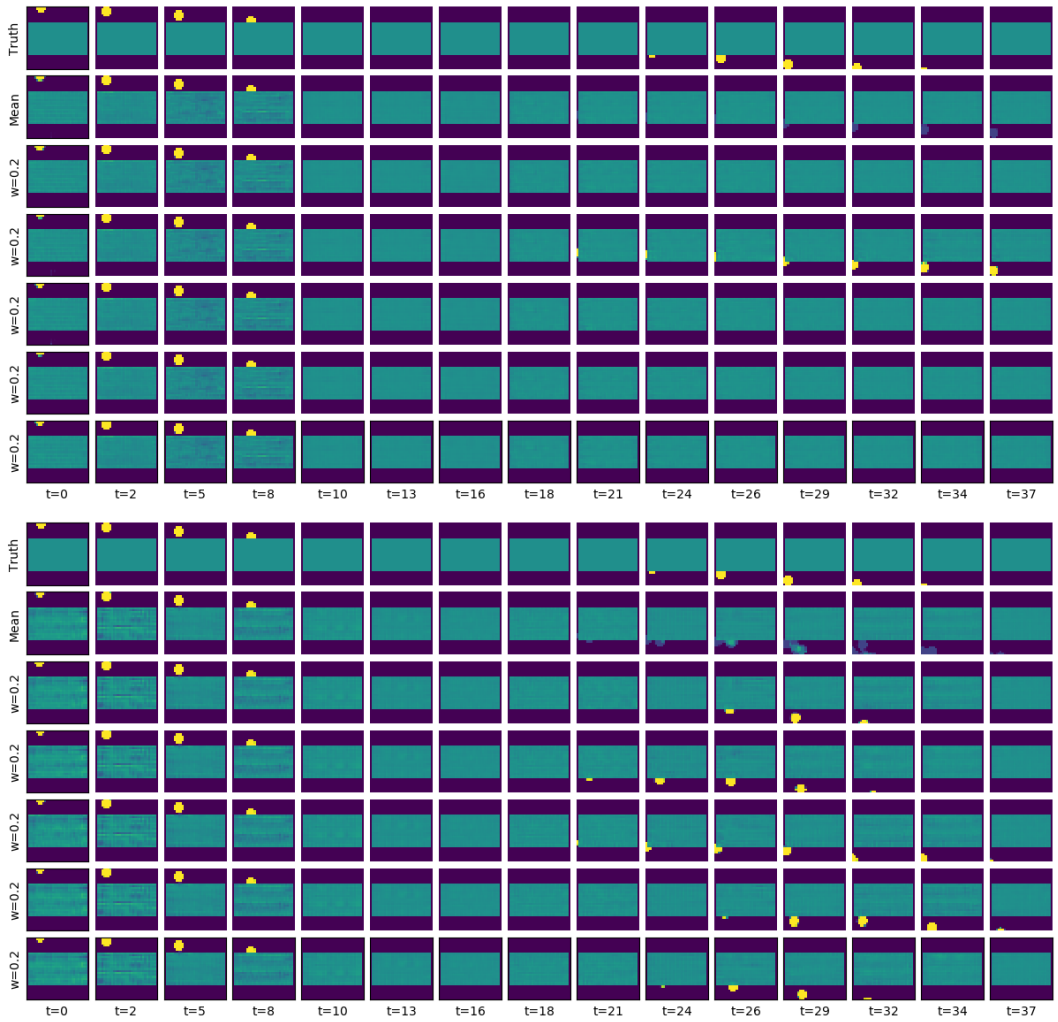


Figure B.3: Visualisation of the learned model. Ground truth observations (top row in each sub figure) are only revealed to the algorithm up until t=19 inclusive. The second row shows the prediction averaged over all particles, all following rows show the prediction made by a single particle. *(Top)* IWAE. *(Bottom)* AESMC.

In Figure B.4, we see experimental results for the LGSSM described in Section 4.2.4.1. We measure the quality of the inference network using a proxy $\sqrt{\sum_{t=1}^{T}(\mu_t^{\text{kalman}} - \mu_t^{\text{approx}})^2}$ where μ_t^{kalman} is the true marginal mean $\mathbb{E}_{p(z_{1:T}|x_{1:T})}[z_t]$ obtained from the Kalman smoothing algorithm and $\mu_t^{\text{approx}} = \left(\sum_{k=1}^{K} w_T^k z_t\right) / \left(\sum_{k=1}^{K} w_T^k\right)$ is an approximate marginal mean obtained from the proposal parameterized by ϕ.

We see that if we train using ELBO_{SMC} with $K_{\text{train}} = 1000$, the performance for inference using SMC (with whichever $K_{\text{test}} \in \{10, 100, 1000\}$) is worse than if we train with ELBO_{IS} with any number of particles $K_{\text{train}} \in \{10, 100, 1000\}$. Examining the other axes of variation:

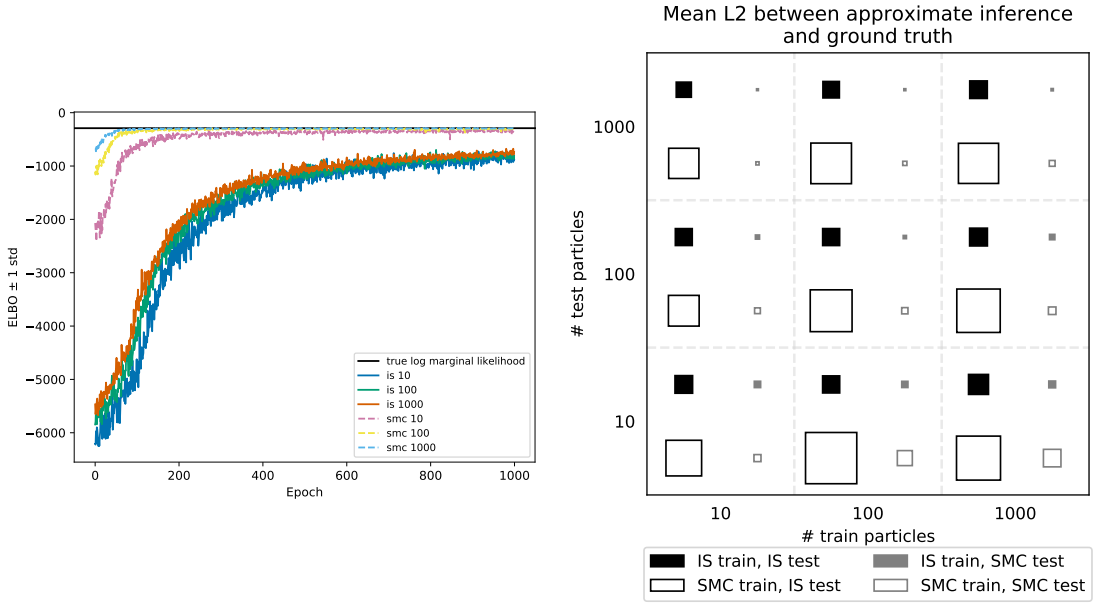


Figure B.4: *(Left)* Optimizing ELBO with respect to ϕ for linear Gaussian state space model (LGSSM). *(Right)* The lengths of the squares are proportional (with a constant factor) to $\sqrt{\sum_{t=1}^{T}(\mu_t^{\text{kalman}} - \mu_t^{\text{approx}})^2}$ which is a proxy for inference quality of ϕ described in the main text. The larger the square, the worse the inference.

- Increasing K_{test} (moving up in Figure B.4 (Right)) improves inference.
- Increasing K_{train} (moving to the right in Figure B.4 (Right)) worsens inference.
- Among different possible combinations of (training algorithm, testing algorithm), (IS, SMC) $\succ$ (SMC, SMC) $\succ$ (IS, IS) $\succ$ (SMC, IS), where we use "$a \succ b$" to denote that the combination a results in better inference than combination b.

B.3 Appendices for Revisiting Reweighted Wake-Sleep

B.3.1 Probabilistic Context Free Grammar

We show the *astronomers* probabilistic context free grammar (PCFG) in Fig. B.5. Figure B.6 shows samples from an inference network trained with variational inference for Monte Carlo objectives (VIMCO) with $K = 20$, conditioned on the sentence $x =$ "astronomers saw stars with telescopes". Figure B.7 shows

production probabilities of the non-terminal NP learned by VIMCO and wake-sleep (WS) with $K = 20$.

$$
\begin{aligned}
\mathrm{S} &\to \mathrm{NP\,VP}\ (1.0) \\
\mathrm{NP} &\to \mathrm{NP\,PP}\ (0.4) | \text{astronomers}\ (0.1) | \text{ears}\ (0.18) | \\
&\quad \text{saw}\ (0.04) | \text{stars}\ (0.18) | \text{telescopes}\ (0.1) \\
\mathrm{VP} &\to \mathrm{V\,NP}\ (0.7) | \mathrm{VP\,PP}\ (0.3) \\
\mathrm{PP} &\to \mathrm{P\,NP}\ (1.0) \\
\mathrm{P} &\to \text{with}\ (1.0) \\
\mathrm{V} &\to \text{saw}\ (1.0).
\end{aligned}
$$

Figure B.5: *The astronomers* PCFG from Manning et al. (1999, Table 11.2). The terminals are $\{\text{astronomers}, \text{ears}, \text{saw}, \text{stars}, \text{telescopes}, \text{with}\}$, the non-terminals are $\{\mathrm{S}, \mathrm{NP}, \mathrm{VP}, \mathrm{PP}, \mathrm{P}, \mathrm{V}\}$ and the start symbol is S. Each row above lists production rules $\{n_i \to \zeta_j\}$ with the corresponding probabilities p_{ij} in the format $n_i \to \zeta_1\,(p_{i1}) | \zeta_2\,(p_{i2}) | \cdots | \zeta_J\,(p_{iJ})$.

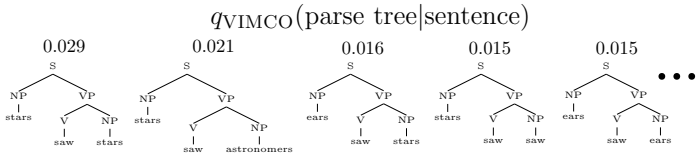


Figure B.6: Samples from the inference network which was trained with VIMCO with $K = 20$.

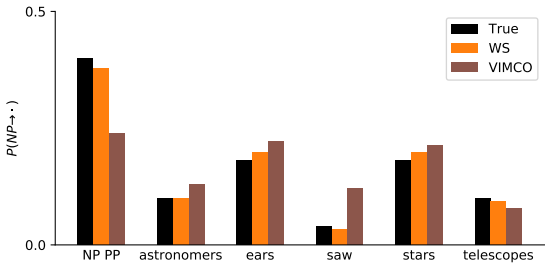


Figure B.7: Production probabilities for the non-terminal NP learned via WS and VIMCO with $K = 20$.

B.3.2 Attend, Infer, Repeat

Attend, Infer, Repeat (AIR) is a complicated model with many components and might be difficult to understand if not described explicitly. Here, we outline details

of our implementation and provide pseudo-code for the inference (Algorithm 5) and generative models (Algorithm 6) in the case of continuous data and Gaussian data likelihood.

Algorithm 5 Inference in AIR

Input: Image $\boldsymbol{x}$, maximum number of inference steps N

$\boldsymbol{h}_0, \boldsymbol{z}_0^{\text{what}}, \boldsymbol{z}_0^{\text{where}} = \text{initialize}()$
for $n \in [1, \ldots, N]$ **do**
  $\boldsymbol{w}_n, \boldsymbol{h}_n = R_\phi\left(\boldsymbol{x}, \boldsymbol{z}_{n-1}^{\text{what}}, \boldsymbol{z}_{n-1}^{\text{where}}, \boldsymbol{h}_{n-1}\right)$
  $p_n \sim \text{Bernoulli}(p \mid \boldsymbol{w}_n)$
  if $p_n = 0$ **then**
    break
  $\boldsymbol{z}_n^{\text{where}} \sim q_\phi^{\text{where}}\left(\boldsymbol{z}^{\text{where}} \mid \boldsymbol{w}_n\right)$
  $\boldsymbol{g}_n = \text{STN}\left(\boldsymbol{x}, \boldsymbol{z}_n^{\text{where}}\right)$
  $\boldsymbol{z}_n^{\text{what}} \sim q_\phi^{\text{what}}\left(\boldsymbol{z}^{\text{what}} \mid \boldsymbol{g}_n\right)$
return $\boldsymbol{z}_{1:n}^{\text{what}}$, $\boldsymbol{z}_{1:n}^{\text{where}}$, n

Algorithm 6 Generation in AIR

Input: $\boldsymbol{z}_{1:n}^{\text{what}}$, $\boldsymbol{z}_{1:n}^{\text{where}}$, n

$\boldsymbol{y}_0 = \boldsymbol{0}$
for $t \in [1, \ldots, n]$ **do**
  $\hat{\boldsymbol{g}}_t = h_\theta^{\text{dec}}\left(\boldsymbol{z}_t^{\text{what}}\right)$
  $\boldsymbol{y}_t = \boldsymbol{y}_{t-1} + \text{STN}^{-1}\left(\hat{\boldsymbol{g}}_t, \boldsymbol{z}_t^{\text{where}}\right)$
$\hat{\boldsymbol{x}} \sim \text{Normal}\left(\boldsymbol{x} \mid \boldsymbol{y}_n, \sigma_x^2 \boldsymbol{I}\right)$
return $\hat{\boldsymbol{x}}$

B.3.3 Gaussian Mixture Model

The reparameterized sampling of Gumbels and conditional Gumbels is described by Tucker et al. (2017, Appendix C) and Grathwohl et al. (2018, Appendix B). In the following, we describe architectures used for the Gaussian mixture model (GMM) experiment (Section 4.3.4.3).

REBAR proposes the following architecture for the control variate $c_\rho(g_{1:K})$:

$$c_\rho^{\text{REBAR}}(g_{1:K}) = \rho_1 \log\left(\frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(\text{softmax}(g_k/e^{\rho_2}), x)}{q_\phi(\text{softmax}(g_k/e^{\rho_2})|x)}\right), \tag{B.45}$$

where $\rho = (\rho_1, \rho_2)$. While the functional form of (B.45) suggests that it will be highly correlated with $\log(\frac{1}{K}\sum_{k=1}^{K} w_k)$, the terms probability mass functions (PMFs) in the fraction are undefined due to the softmax. A straightforward fix is the evaluate "a soft PMF" instead:

$$
\begin{aligned}
p_\theta(\text{softmax}(g_k/e^{\rho_2}), x) &= p_\theta(\text{softmax}(g_k/e^{\rho_2}))p(x|\text{softmax}(g_k/e^{\rho_2})) && \text{(B.46)}\\
&= \text{Categorical}(\text{softmax}(g_k/e^{\rho_2})|\text{softmax}(\theta))\cdot && \text{(B.47)}\\
&\quad \text{Normal}(x|\mu_{\text{softmax}(g_k/e^{\rho_2})}, \sigma^2_{\text{softmax}(g_k/e^{\rho_2})}) && \text{(B.48)}\\
&\approx \text{softmax}(g_k/e^{\rho_2})^\intercal \text{softmax}(\theta) && \text{(B.49)}\\
&\quad \text{Normal}(x|\mu^\intercal \text{softmax}(g_k/e^{\rho_2}), (\sigma^2)^\intercal \text{softmax}(g_k/e^{\rho_2})), && \text{(B.50)}\\
q_\phi(\text{softmax}(g_k/e^{\rho_2})|x) &= \text{Categorical}(\text{softmax}(g_k/e^{\rho_2})|\text{softmax}(\eta_\phi(x))) && \text{(B.51)}\\
&\approx \text{softmax}(g_k/e^{\rho_2})^\intercal \text{softmax}(\eta_\phi(x)). && \text{(B.52)}
\end{aligned}
$$

Optimization of the log-temperature ρ_2 is highly sensitive as low values can make the training unstable.

RELAX proposes using an arbitrary neural network for $c_\rho(g_{1:K})$. Due to the symmetry in the arguments, we pick the following for the GMM experiment:

$$
c_\rho^{\text{RELAX}}(g_{1:K}) = \frac{1}{K}\sum_{k=1}^{K} \text{MLP}_\rho([x, g_k]), \tag{B.53}
$$

where the architecture of the multilayer perceptron (MLP) is $(1 + C)$-16-16-1 (with the tanh nonlinearity between layers) and ρ are the weights are its weights. This architecture is—unlike the one in (B.45)—well-defined for all inputs. A drawback of using such control variate is that it can start out not being very correlated with $\log(\frac{1}{K}\sum_{k=1}^{K} w_k)$.

RELAX also proposes using a summation of the free-form control variate like the one in (B.53) and a more correlated control variate in (B.45).

We have tried all architectures and found that (B.53) leads to the most stable and best training.

Using REBAR/RELAX for more complicated models is possible, however designing an architecture that is highly correlated with the high-variance term and stable to train still remains a challenge.

B.3.4 Discrete VAEs

Rolfe (2016) introduces discrete VAE (DVAE). It combines a prior over binary latent variables with an element-wise spike-and-X smoothing transformation, allowing approximate marginalization of the discrete variables. This results in a continuous relaxation of discrete variables and a low-variance gradient estimator. Vahdat et al. (2018b) replaced the original transformation with an overlapping exponential transformation, leading to a yet lower-variance gradient estimator. While both approaches produce relaxed binary variables, the relaxation is significantly less tight (Vahdat et al. (2018b), Appendix C, Figure 5.) then the CONCRETE of Jang et al. (2017); Maddison et al. (2017b) Both approaches require analytical inverse CDFs of the smoothing transformations, a shortcoming addressed by Vahdat et al. (2018a) — it also leads to a tighter relaxation than its predecessors, however no comparison to CONCRETE is available.

DVAE was designed for undirected binary priors, *e.g.* restricted Boltzmann machines (RBM), and it does not account for the case of categorical latent variables. It is possible to construct a d-dimensional categorical variable from $d-1$ binary variables via stick-breaking construction. This process is slow, however, as it requires $\mathcal{O}(d)$ sequential operations and cannot be parallelized. Moreover, in the case of relaxed variables, the tightness of the derived relaxed categorical variable decreases exponentially with the number of dimensions. This is a major issue in control flows: not only we have to evaluate all branches of the control flow, but the indicator variables that we multiply with outcomes of different branches become exponentially loose with the depth of the flow.

Bibliography

Ryan Adams, Hanna Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and computing*, 18(4):343–373, 2008.

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2014.

Renee Baillargeon, Elizabeth S Spelke, and Stanley Wasserman. Object permanence in five-month-old infants. *Cognition*, 20(3):191–208, 1985.

Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

David Barner and Andrew Scott Baron. *Core Knowledge and Conceptual Change*. Oxford University Press, 2016.

Peter W Battaglia, Jessica B Hamrick, and Joshua B Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.

Atilim Gunes Baydin, Lukas Heinrich, Wahid Bhimji, Bradley Gram-Hansen, Gilles Louppe, Lei Shao, Kyle Cranmer, Frank Wood, et al. Efficient probabilistic inference in the quest for physics beyond the standard model. *arXiv preprint arXiv:1807.07706*, 2018.

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006. ISBN 0387310738.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518): 859–877, 2017.

Taylor L Booth and Richard A Thompson. Applying probability measures to abstract languages. *IEEE transactions on Computers*, 100(5):442–450, 1973.

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. In *International Conference on Learning Representations*, 2015.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *International Conference on Learning Representations*, 2016.

Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based CAPTCHA strengths and weaknesses. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 125–138. ACM, 2011.

Elie Bursztein, Jonathan Aigrain, Angelika Moscicki, and John C Mitchell. The end is nigh: generic solving of text-based CAPTCHAs. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

Susan Carey. Précis of the origin of concepts. *Behavioral and Brain Sciences*, 34(3): 113–124, 2011.

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76 (1), 2017.

Mario Lezcano Casado. Compiled inference with probabilistic programming for large-scale scientific simulations. Master's thesis, University of Oxford, 2017.

Nick Chater and Christopher D Manning. Probabilistic models of language processing and acquisition. *Trends in cognitive sciences*, 10(7):335–344, 2006.

Sourav Chatterjee, Persi Diaconis, et al. The sample size required in importance sampling. *The Annals of Applied Probability*, 28(2):1099–1135, 2018.

Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

Jie Chen and Ronny Luss. Stochastic gradient descent with biased but consistent gradient estimators. *arXiv preprint arXiv:1807.11880*, 2018.

Jian Cheng and Marek J Druzdzel. Ais-bn: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks. *Journal of Artificial Intelligence Research*, 13:155–188, 2000.

Kristy Choi, Mike Wu, Noah Goodman, and Stefano Ermon. Meta-amortized variational inference and learning. *arXiv preprint arXiv:1902.01950*, 2019.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

Kacper Chwialkowski, Heiko Strathmann, and Arthur Gretton. A kernel test of goodness of fit. In *International Conference on Machine Learning*, pages 2606–2615, 2016.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A MATLAB-like environment for machine learning. In *BigLearn, NIPS Workshop*, EPFL-CONF-192376, 2011.

Julien Cornebise. *Adaptive Sequential Monte Carlo Methods*. PhD thesis, University Paris VI Pierre and Marie Curie and Telecom ParisTech, 2009.

Julien Cornebise, Eric Moulines, and Jimmy Olsson. Adaptive sequential monte carlo by means of mixture of experts. *Statistics and Computing*, 24(3):317–337, 2014.

Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. In *Advances in Neural Information Processing Systems*, pages 1647–1655, 2011.

Richard T Cox. Probability, frequency and reasonable expectation. *American journal of physics*, 14(1):1–13, 1946.

Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The Helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

Bruno de Finetti. Sul significato soggettivo della probabilità. *Fundamenta Mathematicae*, 17(1):298–329, 1931.

Bruno de Finetti. La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pages 1–68, 1937.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012.

P Del Moral. Feynman-Kac formulae: genealogical and interacting particle systems with applications. *Probability and its applications*, 2004.

Pierre Del Moral, Arnaud Doucet, Ajay Jasra, et al. On adaptive resampling strategies for sequential Monte Carlo methods. *Bernoulli*, 18(1):252–278, 2012.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

Adji Bousso Dieng, Dustin Tran, Rajesh Ranganath, John Paisley, and David Blei. Variational inference via χ upper bound minimization. In *Advances in Neural Information Processing Systems*, pages 2732–2741, 2017.

Randal Douc and Olivier Cappé. Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64–69. IEEE, 2005.

Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12(656–704):3, 2009.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001.

James Durbin and Siem Jan Koopman. *Time series analysis by state space methods*. Oxford university press, 2012.

David Duvenaud, James R. Lloyd, Roger Grosse, Josh B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *ICML 2013: Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *JLMR Proceedings*, pages 1166–1174, 2013.

Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems*, pages 6062–6071, 2018.

S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, 2016.

Owain Evans, Andreas Stuhlmüller, John Salvatier, and Daniel Filan. Modeling agents with probabilistic programs. http://agentmodels.org, 2017. Accessed: July 4th, 2018.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The Pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

Haichang Gao, Wei Wang, Jiao Qi, Xuqin Wang, Xiyang Liu, and Jeff Yan. The robustness of hollow CAPTCHAs. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 1075–1086. ACM, 2013.

Haichang Gao, Wei Wang, Ye Fan, Jiao Qi, and Xiyang Liu. The robustness of "connecting characters together" CAPTCHAs. *Journal of Information Science and Engineering*, 30(2):347–369, 2014.

Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: Composable inference for probabilistic programming. In *AISTATS*, pages 1682–1690, 2018.

Timon Gehr, Sasa Misailovic, and Martin Vechev. Psi: Exact symbolic inference for probabilistic programs. In *International Conference on Computer Aided Verification*, pages 62–83. Springer, 2016.

Andrew Gelman and Cosma Rohilla Shalizi. Philosophy and the practice of Bayesian statistics. *British Journal of Mathematical and Statistical Psychology*, 66(1):8–38, 2013.

Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

György Gergely, Zoltán Nádasdy, Gergely Csibra, and Szilvia Bíró. Taking the intentional stance at 12 months of age. *Cognition*, 56(2):165–193, 1995.

Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 36, 2014.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452, 2015.

WR Gilks, GO Roberts, and SK Sahu. Adaptive Markov chain Monte Carlo. *J. Am. Stat. Assoc*, 93(1):45–1, 1998.

Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

Adam Golinski, Yee Whye Teh, Frank Wood, and Tom Rainforth. Amortized Monte Carlo integration. *Symposium on Advances in Approximate Bayesian Inference*, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014a.

Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *International Conference on Learning Representations*, 2014b.

Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014. Accessed: 2019-1-12.

Noah D Goodman, Vikash K Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 220–229, 2008.

Noah D Goodman, Joshua B Tenenbaum, and Tobias Gerstenberg. Concepts in a probabilistic language of thought. *To appear in The Conceptual Mind: New Directions in the Study of Concepts*, 2014.

Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Future of Software Engineering, FOSE 2014*, pages 167–181. ACM, 2014.

Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F-radar and signal processing*, volume 140, pages 107–113. IET, 1993.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations*, 2018.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1828–1836, 2015.

Shixiang Gu, Zoubin Ghahramani, and Richard E Turner. Neural adaptive sequential monte carlo. In *Advances in Neural Information Processing Systems*, pages 2611–2619, 2015.

Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. In *International Conference on Learning Representations*, 2016.

Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic data for text localisation in natural images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2315–2324, 2016.

Heikki Haario, Marko Laine, Antonietta Mira, and Eero Saksman. Dram: efficient adaptive mcmc. *Statistics and computing*, 16(4):339–354, 2006.

Jun Han and Qiang Liu. Stein variational adaptive importance sampling. In *Uncertainty in Artificial Intelligence (UAI)*, 2017.

Luis D Hernández, Serafin Moral, and Antonio Salmerón. A monte carlo algorithm for probabilistic propagation in belief networks based on importance sampling and stratified simulation techniques. *International Journal of Approximate Reasoning*, 18(1-2):53–91, 1998.

José Miguel Hernández-Lobato, Yingzhen Li, Mark Rowland, Daniel Hernández-Lobato, Thang Bui, and Richard Eric Turner. Black-box α-divergence minimization. 2016.

Tim Hesterberg. Weighted average importance sampling and defensive mixture distributions. *Technometrics*, 37(2):185–194, 1995.

Rich Hickey. The Clojure programming language. In *Proceedings of the 2008 symposium on Dynamic languages*. ACM New York, NY, USA, 2008.

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

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

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

Jonathan H Huggins and Daniel M Roy. Sequential Monte Carlo as approximate sampling: bounds, adaptive resampling via ∞-ESS, and an application to particle Gibbs. *arXiv preprint arXiv:1503.00966*, 2015.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*, 2018.

Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. In *Workshop on Deep Learning, NIPS*, 2014.

Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Reading text in the wild with convolutional neural networks. *International Journal of Computer Vision*, 116(1):1–20, 2016. doi: 10.1007/s11263-015-0823-z.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Edwin Thompson Jaynes. *Probability theory: the logic of science*. Cambridge University press, 2003.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Biing Hwang Juang and Laurence R Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

Herman Kahn. Random sampling (Monte Carlo) techniques in neutron attenuation problems, I. *Nucleonics*, 6(5):27–37, 1950a.

Herman Kahn. Random sampling (Monte Carlo) techniques in neutron attenuation problems, II. *Nucleonics*, 6(6):60–65, 1950b.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. 2006.

Angelika Kimmig, Bart Demoen, Luc De Raedt, Vitor Santos Costa, and Ricardo Rocha. On the implementation of the probabilistic logic programming language problog. *Theory and Practice of Logic Programming*, 11(2-3):235–262, 2011.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

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

Dan Klein and Christopher D Manning. A parsing: fast exact viterbi parse selection. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 40–47. Association for Computational Linguistics, 2003.

Adam R. Kosiorek, Hyunjik Kim, Ingmar Posner, and Yee Whye Teh. Sequential attend, infer, repeat: Generative modelling of moving objects. In *NeurIPS*, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 2017.

Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4390–4399, 2015.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266): 1332–1338, 2015.

Brenden M Lake, Neil D Lawrence, and Joshua B Tenenbaum. The emergence of organizing structure in conceptual representation. *Cognitive science*, 2018.

Karim Lari and Steve J Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech & language*, 4(1):35–56, 1990.

Dieterich Lawson, George Tucker, Christian A Naesseth, Chris J Maddison, Ryan P Adams, and Yee Whye Teh. Twisted variational sequential monte carlo. *NeurIPS Workshop on Bayesian Deep Learning*, 2018.

Tuan Anh Le. Improving inference in probabilistic programming systems, 2015. Masters' thesis.

Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 1338–1348, 2017a.

Tuan Anh Le, Atılım Güneş Baydin, Robert Zinkov, and Frank Wood. Using synthetic data to train neural networks is model-based reasoning. In *30th International Joint Conference on Neural Networks*, pages 3514–3521. IEEE, 2017b.

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

Tuan Anh Le, Adam R. Kosiorek, N. Siddharth, Yee Whye Teh, and Frank Wood. Revisiting reweighted wake-sleep. *arXiv preprint 1805.10469*, 2018b.

Victor Lempitsky and Andrew Zisserman. Learning to count objects in images. In *Advances in Neural Information Processing Systems*, pages 1324–1332, 2010.

Yingzhen Li and Richard E Turner. Rényi divergence variational inference. In *Advances in Neural Information Processing Systems*, pages 1073–1081, 2016.

Yingzhen Li, Richard E Turner, and Qiang Liu. Approximate inference with amortised MCMC. *arXiv preprint arXiv:1702.08343*, 2017.

Jun S Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044, 1998.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose Bayesian inference algorithm. In *Advances In Neural Information Processing Systems*, pages 2378–2386, 2016.

Qiang Liu, Jason Lee, and Michael Jordan. A kernelized stein discrepancy for goodness-of-fit tests. In *International Conference on Machine Learning*, pages 276–284, 2016.

David J Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. WinBUGS-a Bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing*, 10(4):325–337, 2000.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Chris J Maddison, John Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Teh. Filtering variational objectives. In *Advances in Neural Information Processing Systems*, pages 6576–6586, 2017a.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017b.

Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

Vikash Mansinghka, Tejas D Kulkarni, Yura N Perov, and Josh Tenenbaum. Approximate Bayesian image interpretation using generative probabilistic graphics programs. In *Advances in Neural Information Processing Systems*, pages 1520–1528, 2013.

Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

Vikash Mansinghka, Richard Tibbetts, Jay Baxter, Pat Shafto, and Baxter Eaves. Bayesdb: A probabilistic programming system for querying the probable implications of data. *arXiv preprint arXiv:1512.05006*, 2015.

Dimitris Margaritis. Learning bayesian network model structure from data. Technical report, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science, 2003.

Andrew McCallum, Karl Schultz, and Sameer Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems*, pages 1249–1257, 2009.

Brian Milch, Bhaskara Marthi, and Stuart Russell. Blog: Relational modeling with unknown objects. In *ICML 2004 workshop on statistical relational learning and its connections to other fields*, pages 67–73, 2004.

T. Minka, J.M. Winn, J.P. Guiver, Y. Zaykov, D. Fabian, and J. Bronskill. Infer.NET 2.7, 2018. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

Tom Minka et al. Divergence measures and message passing. Technical report, Technical report, Microsoft Research, 2005.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pages 1791–1799, 2014.

Andriy Mnih and Danilo Rezende. Variational inference for Monte Carlo objectives. In *International Conference on Machine Learning*, pages 2188–2196, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Shakir Mohamed and Balaji Lakshminarayanan. Learning in implicit generative models. *arXiv preprint arXiv:1610.03483*, 2016.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

Lawrence M Murray. Bayesian state-space modelling on high-performance hardware using libbi. *arXiv preprint arXiv:1306.3277*, 2013.

Lawrence M Murray and Thomas B Schön. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 2018.

Christian Naesseth, Scott Linderman, Rajesh Ranganath, and David Blei. Variational sequential Monte Carlo. In *International Conference on Artificial Intelligence and Statistics*, 2018.

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in hakaru (system description). In *International Symposium on Functional and Logic Programming*, pages 62–79. Springer, 2016.

Radford M Neal. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113, 1992.

Willie Neiswanger, Frank Wood, and Eric Xing. The dependent Dirichlet process mixture of objects for detection-free tracking and object modeling. In *Artificial Intelligence and Statistics*, pages 660–668, 2014.

Robert Nishihara, Thomas Minka, and Daniel Tarlow. Detecting parameter symmetries in probabilistic models. *arXiv preprint arXiv:1312.5386*, 2013.

Aditya Nori, Chung-Kil Hur, Sriram Rajamani, and Selva Samuel. R2: An efficient mcmc sampler for probabilistic programs. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Man-Suk Oh and James O Berger. Adaptive importance sampling in monte carlo integration. *Journal of Statistical Computation and Simulation*, 41(3-4):143–168, 1992.

Peter Ondrúška and Ingmar Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Luis E Ortiz and Leslie Pack Kaelbling. Adaptive importance sampling for estimation in structured domains. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 446–454. Morgan Kaufmann Publishers Inc., 2000.

Art B. Owen. *Monte Carlo theory, methods and examples.* 2013.

Brooks Paige and Frank Wood. A compilation target for probabilistic programming languages. In *International Conference on Machine Learning*, pages 1935–1943, 2014.

Brooks Paige and Frank Wood. Inference networks for sequential Monte Carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049, 2016.

Brooks Paige, Frank Wood, Arnaud Doucet, and Yee Whye Teh. Asynchronous anytime sequential monte carlo. In *Advances in Neural Information Processing Systems*, pages 3410–3418, 2014.

John Paisley, David M Blei, and Michael I Jordan. Variational bayesian inference with stochastic search. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1363–1370. Omnipress, 2012.

Judea Pearl. Probabilistic reasoning in intelligent systems: networks of plausible inference. 1988.

Judea Pearl. *Causality.* Cambridge university press, 2009.

Yura Perov and Frank Wood. Automatic sampler discovery via probabilistic programming and approximate bayesian computation. In *Artificial General Intelligence*, pages 262–273. Springer, 2016.

Yura Perov, Tuan Anh Le, and Frank Wood. Data-driven sequential Monte Carlo in probabilistic programming. In *NIPS Workshop on Black Box Learning and Inference*, 2015.

Carsten Peterson and James R Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

Avi Pfeffer. Ibal: A probabilistic rational programming language. In *IJCAI*, pages 733–740. Citeseer, 2001.

Avi Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 137:96, 2009.

Martyn Plummer et al. Jags: A program for analysis of bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, volume 124. Vienna, Austria., 2003.

Lutz Prechelt. Early stopping — but when? In *Neural Networks: Tricks of the Trade*, pages 55–69. Springer, 1998.

Tom Rainforth. Nesting probabilistic programs. In *Uncertainty in Artificial Intelligence*, 2018.

Tom Rainforth, Tuan Anh Le, Jan-Willem van de Meent, Michael A Osborne, and Frank Wood. Bayesian Optimization for Probabilistic Programs. In *Advances in Neural Information Processing Systems*, pages 280–288, 2016a.

Tom Rainforth, Christian Naesseth, Fredrik Lindsten, Brooks Paige, Jan-Willem van de Meent, Arnaud Doucet, and Frank Wood. Interacting particle Markov chain Monte Carlo. In *International Conference on Machine Learning*, pages 2616–2625, 2016b.

Tom Rainforth, Tuan Anh Le, Maximilian Igl, Chris J Maddison, Yee Whye Teh, and Frank Wood. Tighter variational bounds are not necessarily better. *NIPS Workshop on Bayesian Deep Learning*, 2017.

Tom Rainforth, Adam R Kosiorek, Tuan Anh Le, Chris J Maddison, Maximilian Igl, Frank Wood, and Yee Whye Teh. Tighter variational bounds are not necessarily better. In *International Conference on Machine Learning*, 2018a.

Tom Rainforth, Yuan Zhou, Xiaoyu Lu, Yee Whye Teh, Frank Wood, Hongseok Yang, and Jan-Willem van de Meent. Inference trees: Adaptive inference with exploration. *arXiv preprint arXiv:1806.09550*, 2018b.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822, 2014.

Carl Edward Rasmussen. The infinite Gaussian mixture model. In *Advances in neural information processing systems*, pages 554–560, 2000.

Scott Reed and Nando de Freitas. Neural programmer-interpreters. In *International Conference on Learning Representations*, 2016.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, 2014.

Daniel Ritchie, Andreas Stuhlmüller, and Noah Goodman. C3: Lightweight incrementalized mcmc for probabilistic programs using continuations and callsite caching. In *Artificial Intelligence and Statistics*, pages 28–37, 2016a.

Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah Goodman. Neurally-guided procedural models: Amortized inference for procedural graphics programs using neural networks. In *Advances in neural information processing systems*, pages 622–630, 2016b.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Gareth O Roberts and Jeffrey S Rosenthal. Examples of adaptive mcmc. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009.

Jason Tyler Rolfe. Discrete variational autoencoders. In *NIPS*, volume abs/1609.02200, 2016.

Ruslan Salakhutdinov, Joshua Tenenbaum, and Antonio Torralba. One-shot learning with a hierarchical nonparametric bayesian model. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 195–206, 2012.

Antonio Salmerón, Andrés Cano, and Serafın Moral. Importance sampling in bayesian networks using probability trees. *Computational Statistics & Data Analysis*, 34(4):387–413, 2000.

Taisuke Sato and Yoshitaka Kameya. Prism: a language for symbolic-statistical modeling. In *IJCAI*, volume 97, pages 1330–1339, 1997.

Adam Ścibior, Zoubin Ghahramani, and Andrew D Gordon. Practical probabilistic programming with monads. In *ACM SIGPLAN Notices*, volume 50, pages 165–176. ACM, 2015.

Iris Rubi Seaman, Jan-Willem van de Meent, and David Wingate. Modeling theory of mind for autonomous agents with probabilistic programs. *arXiv preprint arXiv:1812.01569*, 2018.

Ross D Shachter and Mark Alan Peot. Simulation approaches to general probabilistic inference on belief networks. *arXiv preprint arXiv:1304.1526*, 2013.

Cosma Rohilla Shalizi et al. Dynamics of Bayesian updating with dependent data and misspecified models. *Electronic Journal of Statistics*, 3:1039–1074, 2009.

N Siddharth, T Brooks Paige, Jan-Willem Van de Meent, Alban Desmaison, Noah Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr. Learning disentangled representations with semi-supervised deep generative models. In *Advances in Neural Information Processing Systems*, pages 5925–5935, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition – Volume 2*, ICDAR '03, pages 958–962. IEEE Computer Society, 2003.

Scott A Sisson, Yanan Fan, and Mark Beaumont. *Handbook of Approximate Bayesian Computation*. Chapman and Hall/CRC, 2018.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.

Armando Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, UNIVERSITY OF CALIFORNIA, BERKELEY, 2008.

Elizabeth S Spelke. Core knowledge, language, and number. *Language Learning and Development*, 13(2):147–170, 2017.

F. Stark, C. Hazırbaş, R. Triebel, and D. Cremers. Captcha recognition with active deep learning. In *GCPR Workshop on New Challenges in Neural Computation*, 2015.

Oleg Starostenko, Claudia Cruz-Perez, Fernando Uceda-Ponga, and Vicente Alarcon-Aquino. Breaking text-based CAPTCHAs with variable word and character orientation. *Pattern Recognition*, 48(4):1101–1112, 2015.

Charles Stein. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, Volume 2: Probability Theory*. The Regents of the University of California, 1972.

Leland Stewart and Perry McCarty. Use of bayesian belief networks to fuse continuous and discrete information for target recognition, tracking, and situation assessment. In *Signal Processing, Sensor Fusion, and Target Recognition*, volume 1699, pages 177–186. International Society for Optics and Photonics, 1992.

Andreas Stuhlmüller and Noah D Goodman. Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs. *Cognitive Systems Research*, 28:80–99, 2014.

Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. Learning stochastic inverses. In *Advances in neural information processing systems*, pages 3048–3056, 2013.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Yee W Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Sharing clusters among related groups: Hierarchical dirichlet processes. In *Advances in neural information processing systems*, pages 1385–1392, 2005.

Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022): 1279–1285, 2011.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

Adrien Todeschini, François Caron, Marc Fuentes, Pierrick Legrand, and Pierre Del Moral. Biips: software for bayesian inference with interacting particle systems. *arXiv preprint arXiv:1412.3779*, 2014.

Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. Deep probabilistic programming. In *International Conference on Learning Representations*, 2017.

Jean-Baptiste Tristan, Daniel Huang, Joseph Tassarotti, Adam C Pocock, Stephen Green, and Guy L Steele. Augur: Data-parallel probabilistic modeling. In *Advances in Neural Information Processing Systems*, pages 2600–2608, 2014.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pages 2624–2633, 2017.

George Tucker, Dieterich Lawson, Shixiang Gu, and Chris J. Maddison. Doubly reparameterized gradient estimators for monte carlo objectives. In *International Conference on Learning Representations*, 2019.

RE Turner, P Berkes, M Sahani, and DJC MacKay. Counterexamples to variational free energy compactness folk theorems. Technical report, 2008.

Richard E. Turner and Maneesh Sahani. Two problems with variational expectation maximisation for time-series models. Cambridge University Press, 2011.

Arash Vahdat, Evgeny Andriyash, and William G. Macready. Dvae#: Discrete variational autoencoders with relaxed boltzmann priors. In *NeurIPS*, 2018a.

Arash Vahdat, William G. Macready, Zhengbing Bian, and Amir Khoshaman. Dvae++: Discrete variational autoencoders with overlapping transformations. In *ICML*, 2018b.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6306–6315. Curran Associates, Inc., 2017.

T Varga and H Bunke. Generation of synthetic training data for an hmm-based handwriting recognition system. In *Seventh International Conference on Document Analysis and Recognition, 2003*, pages 618–622. IEEE, 2003.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for MATLAB. In *Proceeding of the ACM International Conference on Multimedia*, 2015.

Subhashini Venugopalan, Huijuan Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, and Kate Saenko. Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729*, 2014.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

Luis von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. reCAPTCHA: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008. doi: 10.1126/science.1160379.

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2): 1–305, 2008.

Dilin Wang, Hao Liu, and Qiang Liu. Variational inference with tail-adaptive f-divergence. In *Advances in Neural Information Processing Systems 31*, pages 5742–5752. 2018a.

Tongzhou Wang, Yi Wu, Dave Moore, and Stuart J Russell. Meta-learning mcmc proposals. In *Advances in Neural Information Processing Systems*, pages 4150–4160, 2018b.

Zhangyang Wang, Jianchao Yang, Hailin Jin, Eli Shechtman, Aseem Agarwala, Jonathan Brandt, and Thomas S Huang. Deepfont: Identify your font from an image. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 451–459. ACM, 2015.

Stefan Webb, Adam Golinski, Rob Zinkov, Siddharth Narayanaswamy, Tom Rainforth, Yee Whye Teh, and Frank Wood. Faithful inversion of generative models for effective amortized inference. In *Advances in Neural Information Processing Systems*, pages 3074–3084, 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299*, 2013.

David Wingate, Andreas Stuhlmüller, and Noah Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 770–778, 2011.

Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032, 2014.

Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, Bill Freeman, and Josh Tenenbaum. Marrnet: 3D shape reconstruction via 2.5D sketches. In *Advances in neural information processing systems*, pages 540–550, 2017.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.

Daniel H Younger. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208, 1967.

Haohai Yu and Robert A Van Engelen. Refractor importance sampling. *arXiv preprint arXiv:1206.3295*, 2012.

Changhe Yuan and Marek J Druzdzel. Importance sampling in bayesian networks: An influence-based approximation strategy for importance functions. *arXiv preprint arXiv:1207.1422*, 2012.

X. Zhang, Y. Fu, S. Jiang, L. Sigal, and G. Agam. Learning from synthetic data using a stacked multichannel autoencoder. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 461–464, Dec 2015. doi: 10.1109/ICMLA.2015.199.