Joint Stochastic Approximation and Its Application to Learning Discrete Latent Variable Models

Zhijian Ou
Electronic Engineering Department
Tsinghua University, Beijing, China

Yunfu Song
Electronic Engineering Department
Tsinghua University, Beijing, China

Abstract

Although with progress in introducing auxiliary amortized inference models, learning discrete latent variable models is still challenging. In this paper, we show that the annoying difficulty of obtaining reliable stochastic gradients for the inference model and the drawback of indirectly optimizing the target log-likelihood can be gracefully addressed in a new method based on stochastic approximation (SA) theory of the Robbins-Monro type. Specifically, we propose to directly maximize the target log-likelihood and simultaneously minimize the inclusive divergence between the posterior and the inference model. The resulting learning algorithm is called joint SA (JSA). To the best of our knowledge, JSA represents the first method that couples an SA version of the EM (expectation-maximization) algorithm (SAEM) with an adaptive MCMC procedure. Experiments on several benchmark generative modeling and structured prediction tasks show that JSA consistently outperforms recent competitive algorithms, with faster convergence, better final likelihoods, and lower variance of gradient estimates.

1 INTRODUCTION

A wide range of machine learning tasks involves observed and unobserved data. Latent variable models explain observations as part of a partially observed system and usually express a joint distribution $p_\theta(x, h)$ over observation x and its unobserved counterpart h, with parameter θ. Models with discrete latent variables are broadly applied, including mixture modeling, unsupervised learning (Neal, 1992; Mnih and Gregor, 2014), structured output prediction (Raiko et al., 2014; Mnih and Rezende, 2016) and so on.

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

Currently variational methods are widely used for learning latent variable models, especially those parameterized using neural networks. In such methods, an auxiliary amortized inference model $q_\phi(h|x)$ with parameter ϕ is introduced to approximate the posterior $p_\theta(h|x)$ (Kingma and Welling, 2014; Rezende et al., 2014), and some bound of the marginal log-likelihood, used as a surrogate objective, is optimized over both θ and ϕ. Two well-known bounds are the evidence lower bound (ELBO) (Jordan et al., 1999) and the multi-sample importance-weighted (IW) lower bound (Burda et al., 2015). Though with progress (as reviewed in Section 4), a difficulty in variational learning of discrete latent variable models is to obtain reliable (unbiased, low-variance) Monte-Carlo estimates of the gradient of the bound with respect to (w.r.t.) ϕ[1].

Additionally, a common drawback in many existing methods for learning latent-variable models is that they indirectly optimize some bound of the target marginal log-likelihood. This leaves an uncontrolled gap between the marginal log-likelihood and the bound, depending on the expressiveness of the inference model. There are hard efforts to develop more expressive but increasingly complex inference models to reduce the gap (Salimans et al., 2014; Rezende and Mohamed, 2015; Maaløe et al., 2016; Kingma et al., 2017)[2]. But it is highly desirable that we can eliminate the effect of the gap on model learning or ideally directly optimize the marginal log-likelihood, without increasing the complexity of the inference model.

In this paper, we show that the annoying difficulty of obtaining reliable stochastic gradients for the inference model and the drawback of indirectly optimizing the target log-likelihood can be gracefully addressed in a new method based on stochastic approximation (SA) theory

[1] since the gradient of the ELBO w.r.t. θ usually can be reliably estimated.

[2] Notably, some methods are mainly applied to continuous latent variables, e.g. it is challenging to directly apply normalizing flows to discrete random variables, though recently there are some effort (Ziegler and Rush, 2019).

of the Robbins-Monro type (Robbins and Monro, 1951). These two seemingly unrelated issues are inherently related to our choice that we optimizes the ELBO or the similar IW lower bound.

Specifically, we propose to directly maximize w.r.t. θ the marginal log-likelihood[3] and simultaneously minimize w.r.t. ϕ the *inclusive* divergence[4] $KL[p_\theta(h|x)||q_\phi(h|x)]$ between the posterior and the inference model, and fortunately, we can use the SA framework to solve the joint optimization problem. The key is to recast the two gradients as expectations and equate them to zero; then the equations can be solved by applying the SA algorithm, in which the inference model serves as an adaptive proposal for constructing the Markov Chain Monte Carlo (MCMC) sampler. The resulting learning algorithm is called joint SA (JSA), as it couples SA-based model learning and SA-based adaptive MCMC and jointly finds the two sets of unknown parameters (θ and ϕ).

It is worthwhile to recall that there is an another class of methods in learning latent variable models for maximum likelihood (ML), even prior to the recent development of variational learning for approximate ML, which consists of the expectation-maximization (EM) algorithm (Dempster et al., 1977) and its extensions. Interestingly, we show that the JSA method amounts to coupling an SA version of EM (SAEM) (Delyon et al., 1999; Kuhn and Lavielle, 2004) with an adaptive MCMC procedure. This represents a new extension among the various stochastic versions of EM in the literature. This revealing of the connection between JSA and SAEM is important for us to appreciate the new JSA method.

The JSA learning algorithm can handle both continuous and discrete latent variables. The application of JSA in the continuous case is not pursued in the present work, and we leave it as an avenue of future exploration. In this paper, we mainly present experimental results for learning discrete latent variable models with Bernoulli and categorical variables, consisting of stochastic layers or neural network layers. Our results on several benchmark generative modeling and structured prediction tasks demonstrate that JSA consistently outperforms recent competitive algorithms, with faster convergence, better final likelihoods, and lower variance of gradient estimates.

2 PRELIMINARIES

2.1 STOCHASTIC APPROXIMATION (SA)

Stochastic approximation methods are an important family of iterative stochastic optimization algorithms, introduced in (Robbins and Monro, 1951) and extensively studied (Benveniste et al., 1990; Chen, 2002). Basically, stochastic approximation provides a mathematical framework for stochastically solving a root finding problem, which has the form of expectations being equal to zeros. Suppose that the objective is to find the solution λ^* of $f(\lambda) = 0$ with

$$f(\lambda) = E_{z\sim p_\lambda(\cdot)}[F_\lambda(z)], \tag{1}$$

where λ is a d-dimensional parameter vector, and z is an observation from a probability distribution $p_\lambda(\cdot)$ depending on λ. $F_\lambda(z) \in R^d$ is a function of z, providing d-dimensional stochastic measurements of the so-called mean-field function $f(\lambda)$. Intuitively, we solve a system of simultaneous equations, $f(\lambda) = 0$, which consists of d constraints, for determining d-dimensional λ.

Given some initialization $\lambda^{(0)}$ and $z^{(0)}$, a general SA algorithm iterates Monte Carlo sampling and parameter updating, as shown in Algorithm 1. The convergence of SA has been established under conditions (Benveniste et al., 1990; Andrieu et al., 2005; Song et al., 2014), including a few technical requirements for the mean-field function $f(\lambda)$, the Markov transition kernel $K_{\lambda^{(t-1)}}(z^{(t-1)}, \cdot)$ and the learning rates. Particularly, when $f(\lambda)$ corresponds to the gradient of some objective function, then $\lambda^{(t)}$ will converge to local optimum, driven by stochastic gradients $F_\lambda(z)$. To speed up convergence, during each SA iteration, it is possible to generate a set of multiple observations z by performing the Markov transition repeatedly and then use the average of the corresponding values of $F_\lambda(z)$ for updating λ, which is known as SA with multiple moves (Wang et al., 2018), as shown in Algorithm 3 in Appendix.

Remarkably, Algorithm 1 shows stochastic approximation with Markovian perturbations (Benveniste et al., 1990). It is more general than the non-Markovian SA which requires exact sampling $z^{(t)} \sim p_{\lambda^{(t-1)}}(\cdot)$ at each iteration and in some tasks can hardly be realized. In non-Markovian SA, we check that $F_\lambda(z)$ is unbiased estimates of $f(\lambda)$, while in SA with Markovian perturbations, we check the ergodicity property of the Markov transition kernel.

Algorithm 1 The general stochastic approximation (SA) algorithm

for $t = 1, 2, \cdots$ **do**
  Monte Carlo sampling: Draw a sample $z^{(t)}$ with a Markov transition kernel $K_{\lambda^{(t-1)}}(z^{(t-1)}, \cdot)$, which starts with $z^{(t-1)}$ and admits $p_{\lambda^{(t-1)}}(\cdot)$ as the invariant distribution.
  SA updating: Set $\lambda^{(t)} = \lambda^{(t-1)} + \gamma_t F_{\lambda^{(t-1)}}(z^{(t)})$, where γ_t is the learning rate.
end for

[3] "directly" is in the sense that we set the marginal log-likelihood as the objective function in stochastic optimization.

[4] $KL[p_\theta||q_\phi] \triangleq \int p_\theta \log\left(p_\theta/q_\phi\right)$

2.2 VARIATIONAL LEARNING METHODS

Here we briefly review the variational methods, recently developed for learning latent variable models (Kingma and Welling, 2014; Rezende et al., 2014). Consider a latent variable model $p_\theta(x, h)$ for observation x and latent variable h, with parameter θ. Instead of directly maximizing the marginal log-likelihood $\log p_\theta(x)$ for the above latent variable model, variational methods maximize the variational lower bound (also known as ELBO), after introducing an auxiliary amortized inference[5] model $q_\phi(h|x)$:

$$\begin{aligned} ELBO(\theta, \phi; x) &\triangleq E_{q_\phi(h|x)} \log \frac{p_\theta(x, h)}{q_\phi(h|x)} \\ &= \log p_\theta(x) - KL\left[q_\phi(h|x)||p_\theta(h|x)\right] \end{aligned}$$

It is known that maximizing the ELBO w.r.t. ϕ amounts to minimize the exclusive KL-divergence $KL[q_\phi(h|x)||p_\theta(h|x)]$, which has the annoying effect of high variance in estimating gradients mentioned before.

3 METHOD

3.1 DEVELOPING JSA

Consider a latent variable model $p_\theta(x, h)$ for observation x and latent variable h, with parameter θ. Like in variational methods, we also jointly train the target model $p_\theta(x, h)$ together with an auxiliary amortized inference model $q_\phi(h|x)$. The difference is that we propose to directly maximize w.r.t. θ the marginal log-likelihood and simultaneously minimizes w.r.t. ϕ the inclusive KL divergence $KL(p_\theta(h|x)||q_\phi(h|x))$ between the posterior and the inference model, pooled over the empirical dataset:

$$\begin{cases} \min\limits_\theta KL\left[\tilde{p}(x)||p_\theta(x)\right] \\ \min\limits_\phi E_{\tilde{p}(x)} KL\left[p_\theta(h|x)||q_\phi(h|x)\right] \end{cases} \tag{2}$$

where $\tilde{p}(x) \triangleq \frac{1}{n}\sum_{i=1}^n \delta(x - x_i)$ denotes the empirical distribution for a training dataset consisting of n independent and identically distributed (IID) data points $\{x_1, \cdots, x_n\}$. In such a way, we pursue direct maximum likelihood estimation of θ and at the same time avoid the high-variance gradient estimate w.r.t. ϕ.

Fortunately, we can use the SA framework to solve the joint optimization problem Eq.(2) as described below. First, it can be shown that the gradients for optimizing the two objectives in Eq. (2) can be derived as follows[6]:

$$\begin{cases} g_\theta \triangleq -\nabla_\theta KL[\tilde{p}(x)||p_\theta(x)] \\ \qquad\qquad = E_{\tilde{p}(x)p_\theta(h|x)}\left[\nabla_\theta log p_\theta(x, h)\right] \\ g_\phi \triangleq -\nabla_\phi E_{\tilde{p}(x)} KL\left[p_\theta(h|x)||q_\phi(h|x)\right] \\ \qquad\qquad = E_{\tilde{p}(x)p_\theta(h|x)}\left[\nabla_\phi log q_\phi(h|x)\right] \end{cases} \tag{3}$$

By the following Proposition 1, Eq.(3) can be recast in the expectation form of Eq.(1). The optimization problem can then be solved by setting the gradients to zeros and applying the SA algorithm to finding the root for the resulting system of simultaneous equations.

Proposition 1. *The gradients w.r.t. θ and ϕ as in Eq.(3) can be recast in the expectation form of Eq.(1) (i.e. as expectation of stochastic gradients), by letting $\lambda \triangleq (\theta, \phi)^T$, $z \triangleq (\kappa, h_1, \cdots, h_n)^T$, $p_\lambda(z) \triangleq \frac{1}{n}\prod_{i=1}^n p_\theta(h_i|x_i)$, $f(\lambda) \triangleq (g_\theta, g_\phi)^T$, and*

$$F_\lambda(z) \triangleq \begin{pmatrix} \sum_{i=1}^n \delta(\kappa = i)\nabla_\theta log p_\theta(x_i, h_i) \\ \sum_{i=1}^n \delta(\kappa = i)\nabla_\phi log q_\phi(h_i|x_i) \end{pmatrix}.$$

In order to avoid visiting all $h_1, \cdots, h_n$ at every SA iteration (to be explained below), we introduce an index variable κ which is uniformly distributed over $1, \cdots, n$. $\delta(\kappa = i)$ denotes the indicator of κ taking i.

Proof. This can be readily seen by rewriting Eq.(3) as:

$$\begin{cases} g_\theta = \frac{1}{n}\sum\limits_{i=1}^n E_{p_\theta(h_i|x_i)}\left[\nabla_\theta log p_\theta(x_i, h_i)\right] \\ g_\phi = \frac{1}{n}\sum\limits_{i=1}^n E_{p_\theta(h_i|x_i)}\left[\nabla_\phi log q_\phi(h_i|x_i)\right] \end{cases} \tag{4}$$

and applying $E[\delta(\kappa = i)] = \frac{1}{n}$ and the independence between $\kappa, h_1, \cdots, h_n$. □

Recall that as defined in Proposition 1, z consists of $n + 1$ independent components, κ, and $h_1, \cdots, h_n$. At iteration t, the SA algorithm need to draw sample $z^{(t)} \sim p_{\lambda^{(t-1)}}(z)$ either directly or through a Markov transition kernel, and then to update parameters based on stochastic gradients $F_{\lambda^{(t-1)}}(z^{(t)})$ calculated using this sample $z^{(t)}$. The introduction of κ allows us to calculate the stochastic gradients for only one h_i, indexed by κ, instead of calculating over the full batch of all n data points (as originally suggested by Eq.(4)). Minibatching (i.e. drawing a subset of data points) can be achieved by running SA with multiple moves, once the SA procedure with only one h_i is established (see Appendix C for details).

[5] $q_\phi(h|x)$ uses a single, global set of parameters ϕ over the entire training set, which is called amortized inference.

[6] The first equation in Eq.(1) directly follows from the Fisher identity Eq.(5).

Algorithm 2 The JSA algorithm

repeat

Monte Carlo sampling:

Draw κ over $1, \cdots, n$, pick the data point x_κ along with the cached $h_\kappa^{(old)}$, and use MIS to draw h_κ;

SA updating:

Update θ by ascending: $\nabla_\theta log p_\theta(x_\kappa, h_\kappa)$;

Update ϕ by ascending: $\nabla_\phi log q_\phi(h_\kappa|x_\kappa)$;

until convergence

The introduction of κ also allows us to avoid drawing a new full set of $h_1, \cdots, h_n$ in sampling $z^{(t)} \sim p_{\lambda^{(t-1)}}(z)$ at every SA iteration. Suppose that we can construct a base transition kernel for each h_i, denoted by $B_{\lambda,i}(h_i^{(t-1)}, \cdot)$, which admits $p_\theta(h_i|x_i)$ as the invariant distribution, $i = 1, \cdots, n$. Specifically, we can first draw κ uniformly over $1, \cdots, n$, and then draw h_κ by applying $B_{\lambda,\kappa}(h_\kappa^{(t-1)}, \cdot)$ and leave the remaining components $h_1, \cdots, h_{\kappa-1}, h_{\kappa+1}, \cdots, h_n$ unchanged. This is a special instance[7] of random-scan Metropolis-within-Gibbs sampler (Fort et al., 2003).

For designing the base transition kernel $B_{\lambda,i}(h_i^{(t-1)}, \cdot)$, $i = 1, \cdots, n$, we propose to utilize the auxiliary inference model $q_\phi(h_i|x_i)$ as a proposal and use the Metropolis independence sampler (MIS) (Liu, 2008), targeting $p_\theta(h_i|x_i)$. Given current sample $h_i^{(t-1)}$, MIS works as follows:

1. Propose $h_i \sim q_\phi(h_i|x_i)$;
2. Accept $h_i^{(t)} = h_i$ with prob. $min\left\{1, \frac{w(h_i)}{w(h_i^{(t-1)})}\right\}$,

where $w(h_i) = \frac{p_\theta(h_i|x_i)}{q_\phi(h_i|x_i)}$ is the importance sampling weight. Remarkably, we can cancel out the intractable $p_\theta(x_i)$ which appears in both the numerator $w(h_i)$ and denominator $w(h_i^{(t-1)})$. The Metropolis-Hastings (MH) ratio is then calculated as $\frac{p_\theta(h_i,x_i)}{q_\phi(h_i|x_i)} \Big/ \frac{p_\theta(h_i^{(t-1)},x_i)}{q_\phi(h_i^{(t-1)}|x_i)}$.

Finally, the JSA algorithm is summarized in Algorithm 2.

3.2 CONNECTING JSA TO MCMC-SAEM

In this section, we reveal that the JSA algorithm amounts to coupling an SA version of EM (SAEM) (Delyon et al., 1999; Kuhn and Lavielle, 2004) with an adaptive MCMC procedure. To simplify discussion, we consider learning with a single training data point. Learning with a set of IID training data points can be similarly analyzed.

[7] We just sample according to the marginal distribution of $h_i, i = 1, \cdots, n$, instead of the conditional distribution of h_i, since they are independent.

The EM algorithm is an iterative method to find maximum likelihood estimates of parameters for latent variable models (or in statistical terminology, from incomplete data). At iteration t, the E-step calculates the Q-function $Q(\theta|\theta^{(t-1)}) = E_{p_{\theta^{(t-1)}}(h|x)}\left[\nabla_\theta log p_\theta(x,h)\right]$ and the M-step updates θ by maximizing $Q(\theta|\theta^{(t-1)})$ over θ or performing gradient ascent over θ when a closed-form solution is not available. In the E-step, when the expectation in $Q(\theta|\theta^{(t-1)})$ cannot be tractably evaluated, SAEM has been developed (Delyon et al., 1999).

SAEM can be readily obtained from the Fisher identity (see Appendix B for proof),

$$\nabla_\theta log p_\theta(x) = E_{p_\theta(h|x)}\left[\nabla_\theta log p_\theta(x,h)\right], \qquad (5)$$

which again can be viewed as in the expectation form of Eq.(1) (i.e. as expectation of stochastic gradients). So we can apply the SA algorithm, which yields SAEM. It can be seen that the Monte Carlo sampling step in SA fills the missing values for latent variables through sampling $h' \sim p_\theta(h|x)$, which is analogous to the E-step in EM. The SA updating step performs gradient ascent over θ using $\nabla_\theta log p_\theta(x,h')$, analogous to the M-step in EM.

When exact sampling from $p_{\theta^{(t-1)}}(h|x)$ is difficult, an MCMC-SAEM algorithm has been developed (Kuhn and Lavielle, 2004). MCMC-SAEM draws a sample of the latent h by applying a Markov transition kernel which admits $p_{\theta^{(t-1)}}(h|x)$ as the invariant distribution. Given $\theta^{(t-1)}$, the MCMC step in classic MCMC-SAEM is non-adaptive in the sense that the proposal of the transition kernel is fixed. In contrast, in JSA, the auxiliary amortized inference model $q_\phi(h|x)$ acts like an adaptive proposal, adjusted from past realizations of the Markov chain, so that the Markov transition kernel is adapted.

4 RELATED WORK

Novelty. MCMC-SAEM (Kuhn and Lavielle, 2004) and adaptive MCMC (Andrieu and Moulines, 2006) have been separately developed in the SA framework. To the best of our knowledge, JSA represents the first method that couples MCMC-SAEM with adaptive MCMC, and encapsulate them through a joint SA procedure. The model learning of θ and the proposal tuning through ϕ are coupled, evolving together to converge (see Appendix A for convergence of JSA). This coupling has important implications for reducing computational complexity (see below) and improving the performance of MCMC-SAEM with adaptive proposals.

Depending on the objective functions used in joint training of the latent variable model p_θ and the auxiliary inference model q_ϕ, JSA is also distinctive among existing methods. A class of methods relies on a single objective

function, either the variational lower bound (ELBO) or the IW lower bound, for optimizing both p_θ and q_ϕ, but suffers from seeking reliable stochastic gradients for the inference model. The reweighted wake-sleep (RWS) algorithm uses the IW lower bound for optimizing p_θ and the inclusive divergence for optimizing q_ϕ (Bornschein and Bengio, 2014). In contrast, JSA directly optimizes the marginal log-likelihood for p_θ. Though both RWS and JSA use the inclusive divergence for optimizing q_ϕ, the proposed samples from $q_\phi(h|x)$ are always accepted in RWS, which clearly lacks an accept/reject mechanism by MCMC for convergence guarantees.

Compared to the earlier work (Xu and Ou, 2016), this paper is a new development by introducing the random-scan sampler and a formal proof of convergence (Proposition 1 and Appendix A). A similar independent work pointed out by one of the reviewers is called Markov score climbing (MSC) (Naesseth et al., 2020) (see Appendix C for differences between JSA and MSC).

Computational complexity. It should be stressed that though we use MCMC to draw samples from the posterior $p_\theta(h|x)$, we do not need to run the Markov chain for sufficiently long time to converge within one SA iteration[8]. This is unlike in applications of MCMC solely for inference. In learning latent variable models, JSA only runs a few steps of transitions (the same as the number of samples drawn in variational methods) per parameter update and still have parameter convergence. Thus the training time complexity of JSA is close to that of variational methods.

One potential problem with JSA is that we need to cache the sample for latent h per training data point in order to make a persistent Markov Chain. Thus JSA trades storage for quality of model learning. This might not be restrictive given today's increasingly large and cheap storage.

Minimizing inclusive divergence $KL[p_\theta(h|x)||q_\phi(h|x)]$ w.r.t. ϕ ensures that $q_\phi(h|x) > 0$ wherever $p_\theta(h|x) > 0$, which is a basic restriction that makes $q_\phi(h|x)$ a valid proposal for MH sampling of $p_\theta(h|x)$. The exclusive divergence is unsuitable for this restriction. Inclusive minimization also avoid the annoying difficulty of obtaining reliable stochastic gradients for ϕ, which is suffered by minimizing the exclusive divergence.

Learning discrete latent variable models. In order to obtain reliable (unbiased, low-variance) Monte-Carlo estimates of the gradient of the bound w.r.t. ϕ, two well-known possibilities are continuous relaxation of discrete variable (Maddison et al., 2016; Jang et al., 2017), which gives low-variance but biased gradient estimates, and the REINFORCE method (Paisley et al., 2012), which yields unbiased but high-variance gradient estimates. Different control variates - NVIL (Mnih and Gregor, 2014), VIMCO (Mnih and Rezende, 2016), MuProp (Gu et al., 2015), REBAR (Tucker et al., 2017), RELAX (Grathwohl et al., 2018), are developed to reduce the variance of REINFORCE. Other recent possibilities include a Rao-Blackwellization procedure (Liu et al., 2018), a finite difference approximation (Lorberbom et al., 2018), a gradient reparameterization via Dirichlet noise (Yin et al., 2019) or sampling without replacement (Kool et al., 2020).

The gap between the marginal log-likelihood and the ELBO is related to the mismatch between the inference model and the true posterior. Some studies develop more expressive but increasingly complex inference models, which can also be used in JSA. To reduce the gap without introducing any additional complexity to the inference model, there are methods to seek tighter bound, e.g. the IW bound in (Burda et al., 2015). But it is shown in (Rainforth et al., 2018) that using tighter bound of this form is detrimental to the process of learning the inference model. There are efforts to incorporate MCMC into variational inference to reduce the gap. (Salimans et al., 2014) introduces some auxiliary variables from a Markov chain in defining a tighter ELBO, but with additional neural networks for forward and backward transition operators. (Hoffman, 2017) seeks ML estimate of θ by performing additional Hamiltonian Monte Carlo (HMC) steps from the proposal given by q_ϕ, but still estimate ϕ by maximizing ELBO. This method only works with continuous latent variables by using HMC and would be limited since minimizing exclusive divergence encourages low entropy inference models which are not good for proposals. In contrast, JSA is not severely suffered by the mismatch, since although the mismatch affects the sampling efficiency of MIS and SA convergence rate, we still have theoretical convergence to ML estimate.

Adaptive MCMC is an active research area. A classic example is adaptive scaling of the variance of the Gaussian proposal in random-walk Metropolis (Roberts and Rosenthal, 2009). Recently, some adaptive MCMC algorithms are analyzed as SA procedures to study their ergodicity properties (Andrieu and Moulines, 2006). These algorithms minimize the inclusive divergence between the target distribution and the proposal, but use a mixture of distributions from an exponential family as the adaptive proposal. The L2HMC (Levy et al., 2018) learns a parametric leapfrog operator to extend HMC mainly in continuous space, by maximizing expected squared jumped distance. The auxiliary variational MCMC (Habib and Barber, 2019) optimizes the proposal by minimizing ex-

[8]To understand this intuitively, first, note that the inference model is adapted to chase the posterior on the fly, so the proposed samples from $q_\phi(h|x)$ are already good approximate samples for the posterior. Second, the chain could be viewed as running continuously across iterations and dynamically close to the slowly-changing stationary distribution.

clusive divergence.

5 EXPERIMENTS

In this section, we evaluate the effectiveness of the proposed JSA method in training models for generative modeling with Bernoulli and categorical variables and structured output prediction with Bernoulli variables.

5.1 BERNOULLI LATENT VARIABLES

We start by applying various methods to training generative models with Bernoulli (or binary) latent variables, comparing JSA to REINFORCE (Williams, 1992), NVIL (Mnih and Gregor, 2014), RWS (Bornschein and Bengio, 2014), Concrete (Maddison et al., 2016)/Gumbel-Softmax (Jang et al., 2017), REBAR (Tucker et al., 2017), VIMCO (Mnih and Rezende, 2016), and ARM[9] (Yin and Zhou, 2019). For JSA, RWS and VIMCO, we use *particle-number* $= 2$ (i.e. computing gradient with 2 Monte Carlo samples during training), which yields their theoretical time complexity comparable to ARM.

We follow the model setup in (Maddison et al., 2016), which is also used in (Yin and Zhou, 2019). For $q_\phi(h|x)$ and $p_\theta(x|h)$, three different network architectures are examined, as summarized in Table 4 in Appendix, including "Nonlinear" that has one stochastic but two Leaky-ReLU deterministic hidden layers, "Linear" that has one stochastic hidden layer, and "Linear two layers" that has two stochastic hidden layers. We use the widely used binarized MNIST (Salakhutdinov and Murray, 2008) with the standard training/validation/testing partition (50000/10000/10000), making our experimental results directly comparable to previous results in (Bornschein and Bengio, 2014; Yin and Zhou, 2019).

During training, we use the Adam optimizer with learning rate 0.0003 and minibatch size 50; during testing, we use 1,000 proposal samples to estimate the negative log-likelihood (NLL) for each data point in the test set, as in (Mnih and Rezende, 2016; Yin and Zhou, 2019). This setup is also used in section 5.2 but with minibatch size 200. In all experiments in section 5.1, 5.2 and 5.3, we run different methods on the training set, calculate the validation NLL for every 5 epochs, and report the test NLL when the validation NLL reaches its minimum within a certain number of epochs[10] (i.e. we apply early-stopping).

For training with JSA, theoretically we need to cache a latent h-sample for each training data point. Practically, our JSA method runs in two stages. In stage I, we run without caching, i.e. at each iteration, we accept the first proposed sample from $q_\phi(h|x)$ as an initialization and then run MIS with multiple moves. After stage I, we switch to running JSA in its standard manner. The idea is that when initially the estimates of θ and ϕ are far from the root of Eq.(2), the sample may not be valuable enough to be cached and large randomness could force the estimates moving fast to a neighborhood around the root. This two-stage scheme yields fast learning while ensuring convergence. In this experiment, we use the first 600 epochs as stage I and the remaining 400 epochs as stage II. See (Gu and Zhu, 2001; Tan, 2017) for similar two-stage SA algorithms and more discussions.

Table 1: Test NLL of different methods with three different network architectures on generative modeling with Bernoulli variables on MNIST, where $*$ denotes the results reported in (Yin and Zhou, 2019), and the others are obtained based on our implementation[12]. The mean and standard deviation results are computed over five independent trials with different random seeds.

Method	Linear	Nonlinear	Two layers
REINFORCE*	170.1	114.1	159.2
RWS	108.0 ± 0.3	99.2 ± 0.2	96.5 ± 0.2
NVIL*	113.1	102.2	99.8
Concrete*	107.2	99.6	95.6
REBAR*	107.7	100.5	95.5
VIMCO	107.5 ± 0.3	100.6 ± 0.3	95.8 ± 0.1
ARM*	107.2 ± 0.1	98.4 ± 0.3	96.7 ± 0.3
JSA	105.5 ± 0.1	98.2 ± 0.4	95.3 ± 0.1

Table 1 lists the testing NLL results and Figure 1 plots training and testing NLL against training epochs and time. We observe that JSA significantly outperforms other methods with faster convergence and lower NLL for both training and test set. The training time (or complexity) for JSA is comparable to other methods. Notably, the NLL of JSA drops around epoch 600 when JSA switches from stage I (no caching) and stage II (caching). This drop indicates the effectiveness of our two-stage scheme and the empirical benefit of our theoretical development (keeping a persistent chain).

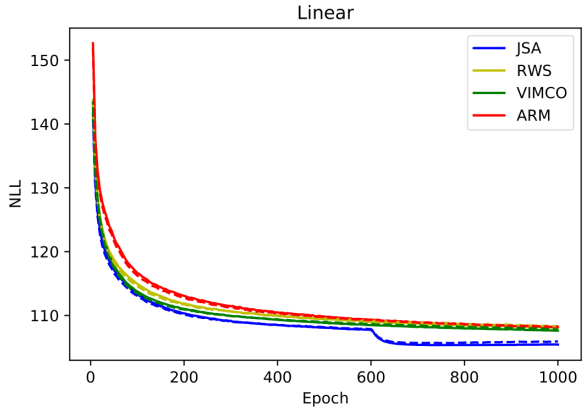
5.2 CATEGORICAL LATENT VARIABLES

In this experiment, we evaluate various methods for training generative models with categorical latent variables. Following the setting of (Yin et al., 2019), there are 20 latent variables, each with 10 categories, and the archi-

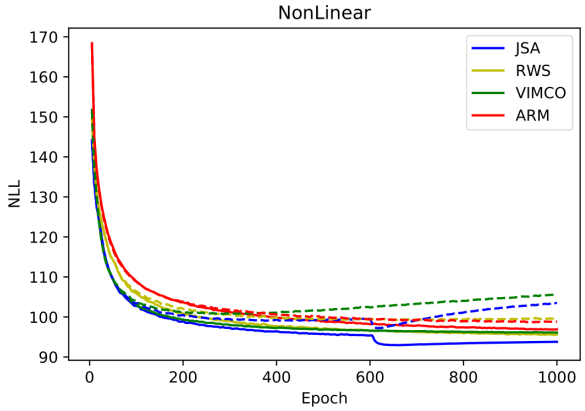
[9] ARSM with Bernoulli latent variables reduces to ARM.

[10] Specifically, 1,000, 500 and 200 epochs are used for experiments in section 5.1, 5.2 and 5.3 respectively.

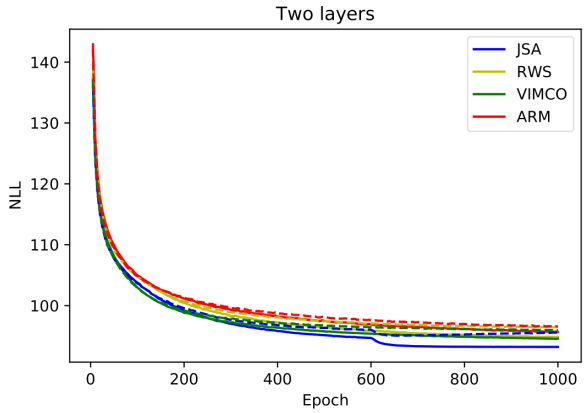
[12] Our Pytorch code and hyperparameters follow (Yin and Zhou, 2019). Some differences are: (Yin and Zhou, 2019) uses TensorFlow, runs up to 1,200 epochs, and monitors the validation NLL every one epoch. Our run saves time and should be no worse under their conditions.



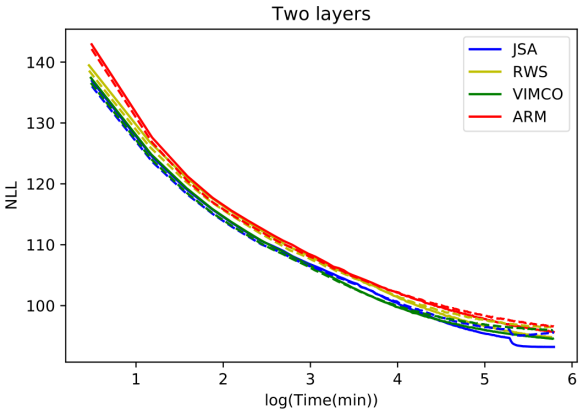
(a) NLL vs training epochs for "Linear"



(b) NLL vs training epochs for "Nonlinear"



(c) NLL vs training epochs for "Two layers"



(d) NLL vs training time for "Two layers"

Figure 1: Plots of training and testing NLL curves for training models with Bernoulli variables on MNIST. (a)(b)(c) are NLL against training epochs for three different network architectures, and (d) is NLL against training wall-clock times for the "Two layers" architecture (See Table 4 in Appendix for the other two architectures). The solid and dash lines correspond to the training and testing NLL respectively. For completeness, we include the testing NLLs against prolonged training epochs in Figure 1 and 2, which show overfitting (see Appendix C for comments).

tectures are listed in Table 5 in Appendix. We compare JSA with other methods that can handle categorical latent variables, including VIMCO (Mnih and Rezende, 2016), straight through Gumbel-Softmax (ST Gumbel-S.) (Jang et al., 2017) and ARSM (Yin et al., 2019). For JSA, ST Gumbel-Softmax and VIMCO, we use *particle-number* $= 20$, which yields their theoretical time complexity close to ARSM. JSA runs with the two-stage scheme, using the first 300 epochs as stage I and the remaining 200 epochs as stage II.

We use the code from (Yin et al., 2019), and implement JSA and VIMCO, making the results directly comparable. We use a binarized MNIST dataset by thresholding each pixel value at 0.5, the same as in (Yin et al., 2019).

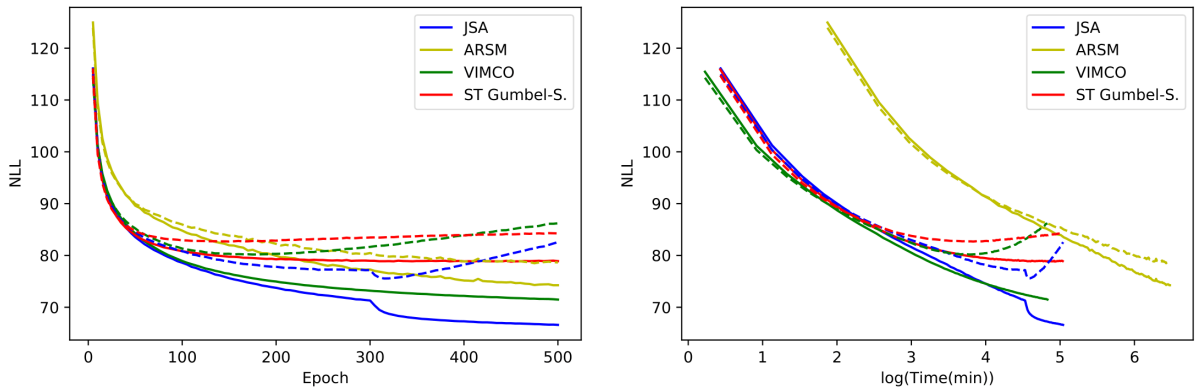
Table 2 lists the test NLL results and Figure 2 plots training and testing NLL against training epochs and time. Similar to results with Bernoulli variables, JSA significantly outperforms other competitive methods. The training time (or complexity) for JSA is comparable to VIMCO and ST Gumbel-Softmax, and ARSM costs much longer time. Also, when JSA switches to stage II, a significant decrease of NLL is observed, which clearly shows the benefit of JSA with caching. Figure 3 plots the gradient variances of different methods. The gradient variances w.r.t. θ for different methods are close and generally smaller than the variances w.r.t. ϕ. Notably, the gradient variance w.r.t. ϕ from JSA is the smallest, which clearly validates the superiority of JSA.

5.3 STRUCTURED OUTPUT PREDICTION

Structured prediction is another common benchmark task for latent variable models. The task is to model a complex

Table 2: Test NLL of different methods on generative modeling of categorical variables on MNIST. The mean and standard deviation results are computed over five independent trials with different random seeds.

Method	VIMCO	ST Gumbel-S.	ARSM	JSA
NLL	79.3 ± 0.5	82.8 ± 0.2	78.7 ± 0.2	75.3 ± 0.3



(a) NLL vs training epochs

(b) NLL vs training time

Figure 2: Plots of training and testing NLL curves for training with categorical variables on MNIST, against (a) training epochs, (b) training wall-clock times. The solid and dash lines correspond to the training and testing NLL respectively.

Table 3: Test NLL of different methods on structured output prediction with Bernoulli variables on MNIST, where "n" denotes *particle-number*. The mean and standard deviation results are computed over five independent trials with different random seeds.

Method	RWS	VIMCO	JSA
n=5	46.2 ± 0.4	46.3 ± 0.1	45.2 ± 0.4
n=10	44.7 ± 0.1	46.2 ± 0.2	44.2 ± 0.1
n=20	43.8 ± 0.1	46.2 ± 0.2	43.2 ± 0.2
n=40	43.2 ± 0.1	46.1 ± 0.2	42.9 ± 0.1
n=80	42.8 ± 0.0	46.0 ± 0.1	42.6 ± 0.1

observation x given a context c, i.e. model the conditional distribution $p(x|c)$. We use a conditional latent variable model $p_\theta(x, h|c) = p_\theta(x|h, c)p_\theta(h|c)$, whose model part is similar to conditional VAE (Sohn et al., 2015).

Specifically, we examine the standard task of modeling the bottom half of a binarized MNIST digit (as x) from the top half (as c), based on the widely used binarized MNIST (Salakhutdinov and Murray, 2008). The latent h consists of 50 Bernoulli variables and the conditional prior $p_\theta(h|c)$ feeds c to two deterministic layers of 200 tanh units to parameterize factorized Bernoulli distributions of h. For $p_\theta(x|h, c)$, we concatenate h with c and feed it to two deterministic layers of 200 tanh units to parameterize factorized Bernoulli outputs. For $q_\phi(h|x, c)$, we feed the whole MNIST digit to two deterministic layers of 200 tanh units to parameterize factorized Bernoulli distributions of h.

RWS, VIMCO, and JSA are conducted with the same models and training setting. During training, we use Adam optimizer with learning rate 0.0003 and minibatch size 100. During testing, we use 1,000 samples from $q_\phi(h|x, c)$ to estimate $-log p_\theta(x|c)$ (NLL) by importance sampling for each data point. JSA runs with the two-stage scheme, using the first 60 epochs as stage I and afterwards as stage II.

Table 3 lists test NLL results against different number of particles (i.e. the number of Monte Carlo samples used to compute gradients during training). JSA performs the best consistently under different number of particles. Both JSA and RWS clearly benefit from using increasing number of particles; but VIMCO not[13].

6 CONCLUSION

We introduce a new class of algorithms for learning latent variable models - JSA. It directly maximizes the marginal likelihood and simultaneously minimizes the inclusive divergence between the posterior and the inference model. JSA couples SA-based model learning and SA-based adap-

[13]Our result of VIMCO here is accordance with the analysis and result of IWAE in (Rainforth et al., 2018), which show that using the IW bound with increasing number of particles actually hurt model learning. IWAE and VIMCO are two such examples, with continuous and discrete latent variables respectively.

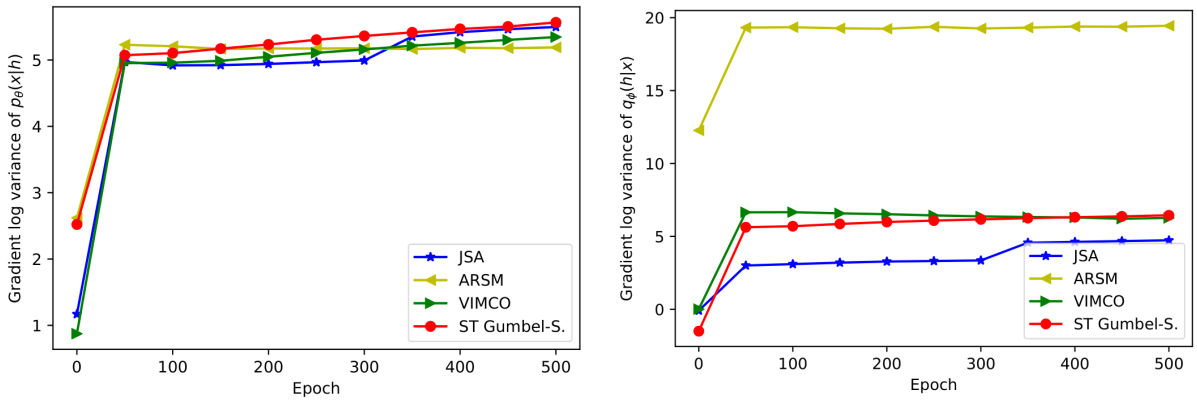


Figure 3: Log variance of gradient of different methods on generative modeling with categorical variables on MNIST, estimated every 50 (out of 500) epochs during training. Each method is computed 1000 times with different latent samples for the first minibatch (the first 200 records) in the epoch. The left and right are variances of gradients w.r.t. θ and ϕ respectively. We report (the logarithm of) the sum of the variances per parameter.

tive MCMC and jointly finds the two sets of unknown parameters for the target latent variable model and the auxiliary inference model. The inference model serves as an adaptive proposal for constructing the MCMC sampler. To our knowledge, JSA represents the first method that couples MCMC-SAEM with adaptive MCMC, and encapsulate them through a joint SA procedure.

JSA provides a simple and principled way to handle both discrete and continuous latent variable models. In this paper, we mainly present experimental results for learning discrete latent variable models with Bernoulli and categorical variables, consisting of stochastic layers or neural network layers. Our results on several benchmark generative modeling and structured prediction tasks demonstrate that JSA consistently outperforms recent competitive algorithms, with faster convergence, better final likelihoods, and lower variance of gradient estimates.

JSA has wide applicability and is easy to use without any additional parameters, once the target latent variable model and the auxiliary inference model are defined. The code for reproducing the results in this work is released at `https://github.com/thu-spmi/JSA`

Acknowledgements

Z. Ou is also affiliated with Beijing National Research Center for Information Science and Technology. This work was supported by NSFC Grant 61976122, China MOE-Mobile Grant MCM20170301. The authors would like to thank Zhiqiang Tan for helpful discussions.

References

Andrieu, C. and Moulines, É. (2006). On the ergodicity properties of some adaptive MCMC algorithms. *The Annals of Applied Probability*, 16(3):1462–1505.

Andrieu, C., Moulines, É., and Priouret, P. (2005). Stability of stochastic approximation under verifiable conditions. *SIAM Journal on control and optimization*, 44(1):283–312.

Benveniste, A., Métivier, M., and Priouret, P. (1990). *Adaptive algorithms and stochastic approximations*. New York: Springer.

Bornschein, J. and Bengio, Y. (2014). Reweighted wake-sleep. *arXiv:1406.2751*.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv:1509.00519*.

Chen, H. (2002). *Stochastic approximation and its applications*. Springer Science & Business Media.

Delyon, B., Lavielle, M., and Moulines, E. (1999). Convergence of a stochastic approximation version of the EM algorithm. *Annals of statistics*, pages 94–128.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39.

Fort, G., Moulines, E., Roberts, G. O., and Rosenthal, J. S. (2003). On the geometric ergodicity of hybrid samplers. *The Annals of Applied Probability*.

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. (2018). Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *ICLR*.

Gu, M. G. and Zhu, H. (2001). Maximum likelihood estimation for spatial models by Markov chain Monte Carlo stochastic approximation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):339–355.

Gu, S., Levine, S., Sutskever, I., and Mnih, A. (2015). Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv:1511.05176*.

Habib, R. and Barber, D. (2019). Auxiliary variational MCMC. In *ICLR*.

Hoffman, M. D. (2017). Learning deep latent Gaussian models with Markov chain Monte Carlo. In *ICML*.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparametrization with Gumble-softmax. In *ICLR*.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233.

Kingma, D. P., Salimans, T., and Welling, M. (2017). Improved variational inference with inverse autoregressive flow. *ArXiv:1606.04934*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. *ICLR*.

Kool, W., van Hoof, H., and Welling, M. (2020). Estimating gradients for discrete random variables by sampling without replacement. In *ICLR*.

Kuhn, E. and Lavielle, M. (2004). Coupling a stochastic approximation version of EM with an MCMC procedure. *ESAIM: Probability and Statistics*, 8:115–131.

Levy, D., Hoffman, M. D., and Sohl-Dickstein, J. (2018). Generalizing Hamiltonian Monte Carlo with neural networks. In *ICLR*.

Liu, J. S. (2008). *Monte Carlo strategies in scientific computing*. Springer Science & Business Media.

Liu, R., Regier, J., Tripuraneni, N., Jordan, M. I., and McAuliffe, J. (2018). Rao-blackwellized stochastic gradients for discrete distributions. *ArXiv:1810.04777*.

Lorberbom, G., Gane, A., Jaakkola, T. S., and Hazan, T. (2018). Direct optimization through arg max for discrete variational auto-encoder. *ArXiv:1806.02867*.

Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2016). Auxiliary deep generative models. *ArXiv:1602.05473*.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv:1611.00712*.

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. *arXiv:1402.0030*.

Mnih, A. and Rezende, D. J. (2016). Variational inference for Monte Carlo objectives. In *ICML*.

Naesseth, C. A., Lindsten, F., and Blei, D. (2020). Markovian score climbing: Variational inference with KL (p||q). *ArXiv:2003.10374*.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113.

Paisley, J., Blei, D., and Jordan, M. (2012). Variational Bayesian inference with stochastic search. *arXiv:1206.6430*.

Raiko, T., Berglund, M., Alain, G., and Dinh, L. (2014). Techniques for learning binary stochastic feedforward neural networks. In *ICLR*.

Rainforth, T., Kosiorek, A. R., Le, T. A., Maddison, C. J., Igl, M., Wood, F., and Teh, Y. W. (2018). Tighter variational bounds are not necessarily better. In *ICML*.

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. *ArXiv:1505.05770*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *ICML*.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407.

Roberts, G. O. and Rosenthal, J. S. (2009). Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367.

Salakhutdinov, R. and Murray, I. (2008). On the quantitative analysis of deep belief networks. In *ICML*, pages 872–879.

Salimans, T., Kingma, D. P., and Welling, M. (2014). Markov chain Monte Carlo and variational inference: Bridging the gap. In *ICML*.

Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In *NIPS*.

Song, Q., Wu, M., and Liang, F. (2014). Weak convergence rates of population versus single-chain stochastic approximation MCMC algorithms. *Advances in Applied Probability*, 46(4):1059–1083.

Tan, Z. (2017). Optimally adjusted mixture sampling and locally weighted histogram analysis. *Journal of Computational and Graphical Statistics*, 26(1):54–65.

Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., and Sohl-Dickstein, J. (2017). Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *NIPS*.

Wang, B., Ou, Z., and Tan, Z. (2018). Learning trans-dimensional random fields with applications to language modeling. *PAMI*, 40(4):876–890.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Xu, H. and Ou, Z. (2016). Joint stochastic approximation learning of Helmholtz machines. In *ICLR Workshop Track*.

Yin, M., Yue, Y., and Zhou, M. (2019). ARSM: Augment-reinforce-swap-merge estimator for gradient backpropagation through categorical variables. In *ICML*.

Yin, M. and Zhou, M. (2019). ARM: Augment-reinforce-merge gradient for stochastic binary networks. In *ICLR*.

Ziegler, Z. M. and Rush, A. M. (2019). Latent normalizing flows for discrete sequences. *ArXiv:1901.10548*.

A On convergence of JSA

The convergence of SA has been established under conditions (Benveniste et al., 1990; Andrieu et al., 2005; Song et al., 2014), including a few technical requirements for the mean-field function $f(\lambda)$, the transition kernel $K_{\lambda^{(t-1)}}(z^{(t-1)}, \cdot)$ and the learning rates. We mainly rely on Theorem 5.5 in (Andrieu et al., 2005) to show the convergence of JSA.

For the transition kernel in JSA, it is shown in (Fort et al., 2003) that the random-scan Metropolis-within-Gibbs sampler satisfies the V-uniform ergodicity under some mild conditions. The V-uniform ergodicity of the transition kernel is the key property for the transition kernel K_λ to satisfy the drift condition, which is an important condition used in (Andrieu et al., 2005) to establish the convergence of the SA algorithm.

Specifically, we can apply Theorem 5.5 in Andrieu et al. (2005) to verify the conditions (A1) to (A4) to show JSA convergence. (A1) is the Lyapunov condition on $f(\lambda)$, which typically holds for stochastic optimization problems like in this paper, in which $f(\lambda)$ is a gradient field for some bounded, real-valued and differentiable objective functions. (A2) and (A3) hold under the drift condition, which is satisfied by the transition kernel in JSA as outlined above. (A4) gives conditions on the learning rates, e.g. satisfying that $\sum_{t=0}^{\infty} \gamma_t = \infty$ and $\sum_{t=0}^{\infty} \gamma_t^2 < \infty$.

B Proof of the Fisher identity

Note that $E_{p_\theta(h|x)}\left[\nabla_\theta log p_\theta(h|x)\right] = 0$, so we have $E_{p_\theta(h|x)}\left[\nabla_\theta log p_\theta(x,h)\right] = E_{p_\theta(h|x)}\left[\nabla_\theta log p_\theta(x) + log p_\theta(h|x)\right] = \nabla_\theta log p_\theta(x)$.

C Additional comments about JSA

Minibatching in JSA. In our experiments, we run JSA with multiple moves (Wang et al., 2018) to realize minibatching. Specifically, we draw a minibatch of data points, say, $x_{\kappa_1}, \cdots, x_{\kappa_m}$, and for each $x_{\kappa_j}, j = 1, \cdots, m$, we generate multiple h-samples $h_{\kappa_j,k}, k = 1, \cdots, \textit{particle-number}$. In our pytorch implementation, firstly, we generate proposals for all $h_{\kappa_j,k}, j = 1, \cdots, m, k = 1, \cdots, \textit{particle-number}$, according to q_ϕ and calculate their importance weights. This can be easily organized in tensor operations. Then we perform accept/reject to obtain the h-samples, which takes negligible computation. In this way, JSA can efficiently utilize modern tensor libraries.

Comparison with (Naesseth et al., 2020). A similar independent work pointed out by one of the reviewers is called Markov score climbing (MSC) (Naesseth et al., 2020). It is interesting that MSC uses the conditional importance sampler (CIS), whereas JSA uses the random-scan sampler. We see two further differences. First, the random-scan sampler in JSA satisfies the V-uniform ergodicity, which ensures that the transition kernel satisfies the drift condition for the SA convergence. It is not clear if the CIS sampler satisfies the assumptions listed in (Naesseth et al., 2020). Second, the model setups in (Naesseth et al., 2020) and our work are different for large-scale data. By using the random-scan sampler, JSA can support minibatching in our setup. Minibatching of MSC in the setup of (Naesseth et al., 2020) leads to systematic errors.

Overfitting observed in Figure 1 and 2. Similar to previous studies in optimizing discrete latent variable models (e.g. Tucker et al. (2017); Kool et al. (2020)), we observe overfitting in Figure 1 and 2, when the training process was deliberately prolonged in order to show the effects of different optimization methods for decreasing the training NLLs. A common approach (which is also used in our experiments) to address overfitting is monitoring NLL on validation data and apply early-stopping. Thus, for example in Figure 2a, the testing NLL for JSA should be read from the early-stopping point shortly after epoch 300. Different methods early-stop at different epochs, by monitoring the validation NLLs as described in the 3rd paragraph in section 5.1. For example, in training with categorical latent variables (Figure 2), JSA, ARSM and VIMCO early-stop at epoch 320, 485 and 180 respectively. These early-stopping results are precisely the testing NLLs reported in Table 1 and 2, from which we can see that JSA significantly outperforms other competitive methods across different latent variable models.

D Additional tables and figures

Algorithm 3 SA with multiple moves

for $t = 1, 2, \cdots$ **do**

1. Set $z^{(t,0)} = z^{(t-1,K)}$. For k from 1 to K, generate $z^{(t,k)} \sim K_{\lambda^{(t-1)}}(z^{(t,k-1)}, \cdot)$, where $K_{\lambda^{(t-1)}}(z^{(t,k-1)}, \cdot)$ is a Markov transition kernel that admits $p_{\lambda^{(t-1)}}(\cdot)$ as the invariant distribution.

2. Set $\lambda^{(t)} = \lambda^{(t-1)} + \gamma_t \{\frac{1}{K} \sum_{z \in B^{(t)}} F_{\lambda^{(t-1)}}(z)\}$, where $B^{(t)} = \{z^{(t,k)} | k = 1, \cdots, K\}$.

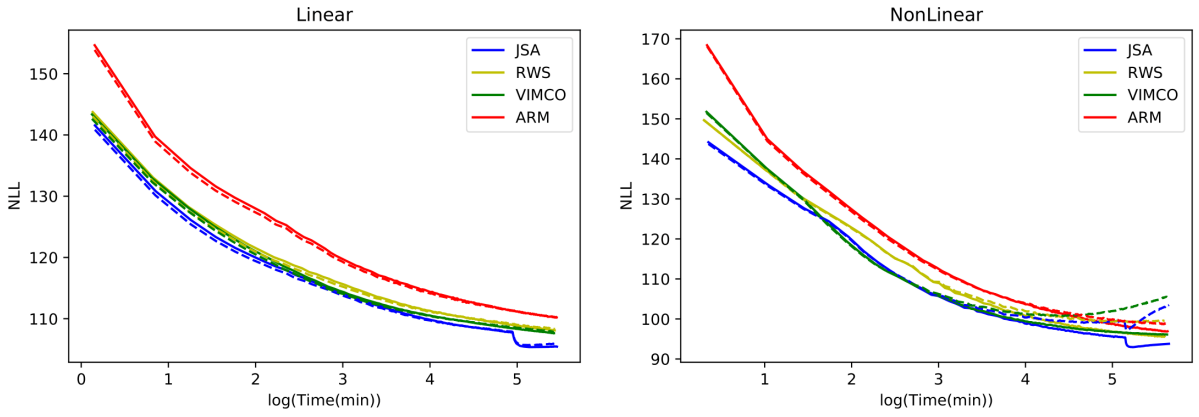
end for

Table 4: The three different network architectures in generative modeling with Bernoulli variables on MNIST, which are the same as Table 1 in (Yin and Zhou, 2019). The following symbols "$\rightarrow$", "]", ")", and "$\rightsquigarrow$" represent deterministic linear transform, leaky rectified linear units (LeakyReLU) nonlinear activation, sigmoid nonlinear activation, and random sampling respectively.

	Nonlinear	Linear	Linear two layers
$q_\phi(h\|x)$	$784 \rightarrow 200] \rightarrow 200] \rightarrow 200) \rightsquigarrow 200$	$784 \rightarrow 200) \rightsquigarrow 200$	$784 \rightarrow 200) \rightsquigarrow 200 \rightarrow 200 \rightsquigarrow 200$
$p_\theta(x\|h)$	$784 \leftsquigarrow (784 \leftarrow [200 \leftarrow [200 \leftarrow 200$	$784 \leftsquigarrow (784 \leftarrow 200$	$784 \leftsquigarrow (784 \leftarrow 200 \leftsquigarrow (200 \leftarrow 200$

Table 5: Network architectures in generative modeling with categorical variables on MNIST, which are the same as in (Yin et al., 2019). The following symbols "$\rightarrow$", "]", ")", "$\rightsquigarrow$" and "$\hookrightarrow$" represent deterministic linear transform, LeakyReLU nonlinear activation, sigmoid nonlinear activation, random sampling over Bernoulli distributions and random sampling over categorical distributions respectively. For sampling over categorical distributions, there are 20 categorical variables each with 10 categories, thus the 200 output units are divided into 20 groups each with 10 units. Then we apply softmax to each group, and obtain the categorical distribution of each variable. After obtaining random sample of each variable, we use one-hot encoding and concatenate them to obtain a 200 dimensional array.

$q_\phi(h\|x)$	$p_\theta(x\|h)$
$784 \rightarrow 512] \rightarrow 256] \rightarrow 200 \hookrightarrow 200$	$784 \leftsquigarrow (784 \leftarrow [512 \leftarrow [256 \leftarrow 200$



(a) NLL vs training time for "Linear"

(b) NLL vs training time for "Noninear"

Figure 4: Plots of training and testing NLL curves for training models with Bernoulli variables on MNIST, against training wall-clock times. (a)(b) are for "Linear" and "Nonlinear" architectures respectively. The solid and dash lines correspond to the training and testing NLL respectively.