VARIATIONAL MESSAGE PASSING WITH STRUCTURED INFERENCE NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Powerful models require powerful algorithms to be effective on real-world problems. We propose such an algorithm for a class of models that combine deep models with probabilistic graphical models to develop interpretable deep-learning models. Our algorithm is a natural-gradient message-passing method whose messages automatically reduce to stochastic-gradients for the deep components of the model. Using a special-structure inference network, our algorithm exploits the structural properties of the model to gain computational efficiency while retaining the simplicity and generality of deep-learning algorithms. By combining the strength of two different types of inference procedures, our approach offers a framework that simultaneously enables structured, amortized, and natural-gradient inference for complex models.

1 INTRODUCTION

To analyze real-world data, machine learning relies on models that can extract useful patterns. For this purpose, deep models such as Deep Neural Networks (DNNs) are a popular choice and can learn flexible representations. Another popular choice are probabilistic graphical models (PGMs) which can find interpretable structures in the data. Recent work on combining these two types of models hopes to borrow their complimentary strengths and provide powerful models that are also easy to interpret (Johnson et al., 2016; Krishnan et al., 2015; Archer et al., 2015; Fraccaro et al., 2016).

To apply such hybrid models to real-world problems, we need efficient algorithms that can extract useful structure from the data. However, the two fields of deep learning and PGMs traditionally use different types of algorithms. For deep learning, stochastic-gradient (SG) methods are the most popular choice, e.g., those based on back-propagation. These algorithms are not only widely applicable, but also employ amortized inference to enable fast inference at test time (Rezende et al., 2014; Kingma & Welling, 2013). On the other hand, most popular algorithms for PGMs exploit the model's graphical structure to gain computational efficiency, e.g., variational message passing (VMP) (Winn & Bishop, 2005), expectation propagation (Minka, 2001), Kalman filtering, and more recently stochastic variational inference (SVI) (Hoffman et al., 2013). The two fields of deep learning and PGMs employ fundamentally different inferential strategies and a natural question is, whether we can design algorithms that combine their respective strengths.

In this paper, we propose such an algorithm. Our algorithm is a natural-gradient message-passing algorithm whose messages automatically reduce to stochastic-gradients for the deep components of the model. In addition to such a flexible message-passing scheme, our algorithm performs amortized inference by using a structured inference network that mimics the structure of the model. Overall, this enables Structured, Amortized, and Natural-gradient (SAN) updates and therefore we call our algorithm the SAN algorithm.

Unlike many existing methods that continue using SG methods even when some of the model components are PGMs, our SAN algorithm exploits the probabilistic structure to gain computational efficiency. Our algorithm is in a similar spirit to the work of Johnson et al. (2016) but it enjoys much simpler implementation. Experimental comparisons show that our method gives comparable performance to the method of Johnson et al. (2016) while simplifying and generalizing it.

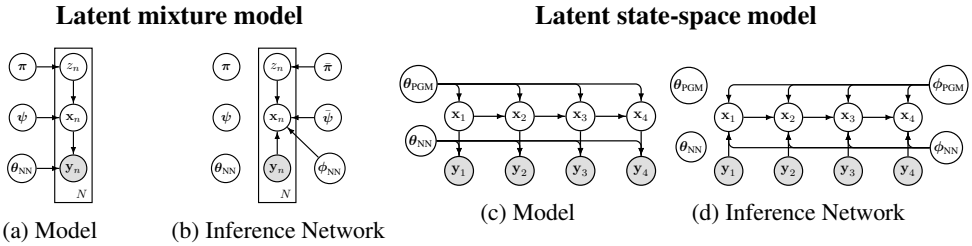


Figure 1: Two examples of model classes that combine probabilistic graphical models with deep neural networks. The two figures on the left show the model and corresponding inference network for the latent mixture model, while the two figures on the right show the same for the latent state-space model. For the mixture model, $\boldsymbol{\psi} := \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ contains the parmaeters of Gaussian mixture components, and $\bar{\boldsymbol{\psi}}$ contains the corresponding parameters for the inference network. Our inference network mimics the structure of the model. The only difference is that the arrows from $\mathbf{y}_n$ to $\mathbf{x}_n$ are reversed. Another important aspect is that the inference network uses $\boldsymbol{\theta}_{\text{PGM}}$ by a different set of parameters $\boldsymbol{\phi}_{\text{PGM}}$. These differences play an important role in the design of our algorithm.

2 The Model and Challenges with Its Inference

We consider the modeling of data vectors $\mathbf{y}_n$ by using local latent vectors $\mathbf{x}_n$. The correlations among N data vectors $\mathbf{y} := \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$ can then be captured by using a probabilistic graphical model (PGM) over all the latent vectors $\mathbf{x} := \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. This combination of probabilistic graphical model and neural network is referred to as structured auto-encoder (SVAE) by Johnson et al. (2016) and has also been used in other recent works (Krishnan et al., 2015; Archer et al., 2015).

Following previous approaches, we consider the following model in this paper:

$$p(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta}) = p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})p(\mathbf{x}, \boldsymbol{\theta}) = \underbrace{\left[p(\boldsymbol{\theta}_{\text{NN}}) \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_{\text{NN}})\right]}_{\text{DNN}} \underbrace{\left[p(\mathbf{x}|\boldsymbol{\theta}_{\text{PGM}})p(\boldsymbol{\theta}_{\text{PGM}})\right]}_{\text{PGM}}, \quad (1)$$

where $\boldsymbol{\theta}_{\text{NN}}$ are the parameters of DNN, $\boldsymbol{\theta}_{\text{PGM}}$ parameters of PGM, and $\boldsymbol{\theta} := \{\boldsymbol{\theta}_{\text{NN}}, \boldsymbol{\theta}_{\text{PGM}}\}$. Below, we give two examples of graphical models.

Mixture Models: To discover patterns in the data, we can model $\mathbf{x}_n$ by using a finite mixture model with K mixture components. To do so, we define additional discrete latent vector $\mathbf{z}_n$ whose k'th element is $z_{nk} \in \{0, 1\}$. We can use then as a Gaussian mixture model as the PGM prior,

$$p(\mathbf{y}, \mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \left[p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_{\text{NN}}) \prod_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}}\right], \quad (2)$$

where $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are mean and covariance of the k'th mixture and π_k are the mixture proportions summing to 1, and $\boldsymbol{\theta}_{\text{PGM}} := \{\boldsymbol{\psi}_k, \pi_k\}_{k=1}^{K}$ and $\boldsymbol{\psi}_k := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$. The parameter prior can be chosen as described in Bishop (2006) and the corresponding graphical model is shown in Figure 1a. This is an example of a model that combines PGM with deep networks.

State-Space Models: When $\mathbf{y}_n$ is a time series, we can model the latent $\mathbf{x}_n$ using a linear dynamical system, for example, we can use the following model:

$$p(\mathbf{y}, \mathbf{x}|\boldsymbol{\theta}) = \left[\prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}_{\text{NN}})\mathcal{N}(\mathbf{x}_n|\mathbf{A}\mathbf{x}_{n-1}, \mathbf{Q})\right] \mathcal{N}(\mathbf{x}_0|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \quad (3)$$

where $\mathbf{A}$ is the transition matrix, $\mathbf{Q}$ is the process noise, and μ_0 and $\boldsymbol{\Sigma}_0$ are the mean and covariance of the initial distribution. Here, $\boldsymbol{\theta}_{\text{PGM}} := \{\mathbf{A}, \mathbf{Q}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0\}$. Again, we can use standard conjugate-priors for $\boldsymbol{\theta}_{\text{PGM}}$ (Beal, 2003). The graphical model is shown in Figure 1c.

For these models, our goal is to approximate the posterior distribution $p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y})$ using an inference network that can be used to perform *amortized inference* (Ritchie et al., 2016). Such inference networks have previously been used in variational auto-encoders (VAE) (Kingma & Welling,

2013) where they are shown to perform extremely well, however, in our case, designing such high-performance networks is a challenging task.

The main challenge lies in designing networks that can flexibly capture the structure present in the PGM component, i.e., $p(\mathbf{x}|\boldsymbol{\theta})$. In VAEs, a DNN with parameters $\boldsymbol{\phi}$ is used as the inference network $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})$. This choice is reasonable since all of the structure in the posterior is attributed to the DNN likelihood $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$. The PGM part in this case is simply a standard normal distribution $\mathcal{N}(\mathbf{x}|0, \mathbf{I})$ which does not have any special structure. Therefore using a pure DNN based inference-network is sufficient.

Our model is explicitly designed to exploit the PGM structure so that we can have interpretable deep models. Therefore, it is essential to capture such structure in the inference network. However, a purely DNN based inference network puts all its power in the likelihood term and ignores the structure. This might give inaccurate predictions. For example, consider the case where $\mathbf{x}$ is a time series (as shown in Fig. 1c). Here, the prediction of a future observation, say, $\mathbf{y}_{10}$ at time $n = 10$ given past data, say, $\{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\}$ would be inaccurate because the inference network does not capture the correlation between $\mathbf{x}_{10}$ and $\mathbf{x}_3$. Rather, the network relies on a DNN that is conditioned on the unavailable observation $\mathbf{y}_{10}$. For such forecasting and imputation tasks, the inference network must capture the structure of the posterior accurately.

A solution to this problem is to exploit the PGM structure in the inference network by replacing all of its edges by neural networks (Krishnan et al., 2015; Fraccaro et al., 2016). When the PGM itself is complex, this solution is reasonable, but when it is a simple model, such as linear dynamical system, this approach might be too aggressive. Using DNNs in such cases would dramatically increase the number of parameters. This, in turn, will lead to a loss in speed and a possible deterioration in performance.

Johnson et al. (2016) propose an alternative double-loop procedure to make sure that the probabilistic structure in PGM can be captured in the posterior distribution. Their algorithm however requires running a double loop algorithm where the inner loop needs to be run until convergence otherwise the algorihtm might diverge. We give more details in the next section, but their approach is computationally intensive and might converge slowly.

To summarise, existing methods either simply ignore the graphical model structures or do not exploit them efficiently.

To solve this problem, we propose an inference network that preserves the structure of the posterior to facilitate fast and accurate amortized inference. We incorporate this network within a message-passing framework. For the deep components of the model, these messages automatically reduce to stochastic-gradients, while for the rest of the model, they are used to perform natural-gradient updates, just like in SVI (Hoffman et al., 2013). We give two examples of our approach and apply them to the experiments seen in Johnson et al. (2016). Our results show that our method performs comparably, while simplifying and generalizing their approach.

3 Conjugate-Structure Inference Networks

As explained in the previous section, designing an inference network that preserves the prior's probabilistic structure and enables flexible amortized inference is a challenging task. We solve this problem by proposing a structured inference network that takes the following form:

$$q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi}) := \frac{1}{\mathcal{Z}(\boldsymbol{\phi})} \underbrace{q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi}_{\text{NN}})}_{\text{DNN}} \underbrace{p(\mathbf{x}|\boldsymbol{\phi}_{\text{PGM}})}_{\text{PGM}} \tag{4}$$

where $p(\mathbf{x}|\boldsymbol{\phi}_{\text{PGM}})$ is the same distribution as $p(\mathbf{x}|\boldsymbol{\theta}_{\text{PGM}})$ but is parametrized by a new parameter $\boldsymbol{\phi}_{\text{PGM}}$, and $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi}_{\text{NN}})$ is an exponential-family distribution whose natural parameters are parametrized by a DNN in a similar fashion as in VAE. The role of the PGM term is to enforce the PGM structure, while the role of DNN term is to enable flexibility.

For amortized inference, we need to be able to generate samples from $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})$ which is an easy when we the normalizing constant $\mathcal{Z}(\boldsymbol{\phi})$ is available in closed form. To satisfy this requirement, we choose the network $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi}_{\text{NN}})$ to be conjugate to $p(\mathbf{x}|\boldsymbol{\phi}_{\text{PGM}})$, i.e., they both must take the same functional form with respect to $\mathbf{x}$. For example, if PGM is a Gaussian, we can choose $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi}_{\text{NN}})$

to be a Gaussian as well. This way $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})$ will also be Gaussian and $\mathcal{Z}(\boldsymbol{\phi})$ can be obtained in closed-form. In general, we choose the first term to be a member of conjugate exponential family. Sampling can then be performed using conjugate message-passing algorithm algorithm. We call our inference network a conjugate-structure inference networks.

An important feature of our inference network is that it is independent of $\boldsymbol{\theta}$. Due to this our method does not suffer from the "implicit gradient" problems encountered in the method of Johnson et al. (2016). Their method approximates $q(\mathbf{x}) \propto q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})p(\mathbf{x}, \boldsymbol{\theta})$ by optimizing a variational lower bound. As a result, the parameters of $q(\mathbf{x})$ depend on $\boldsymbol{\theta}$. Due to this dependence, if the inner loop is not run until convergence, the outer loop might not have correct gradients which might result in the divergence of their method. By simply using new parameters $\boldsymbol{\phi}_{\text{PGM}}$, we make sure that the normalizing constant $\mathcal{Z}$ as well as the generated samples are all independent of $\boldsymbol{\theta}_{\text{PGM}}$. This simplifies the inference because we do not have to worry about the inference network when taking gradients with respect to the variational parameters corresponding to the variational distribution of $\boldsymbol{\theta}$.

Another consequence of our proposed inference network is that when the original PGM is non-conjugate (e.g., a T-distribution), we can still use a conjugate term in the second term (e.g., a Gaussian) in (4). We only have to make sure that the inference network captures useful structure in the posterior. This modification enables us to design easy-to-sample inference networks for complex models. The conjugate structure in our inference network simplifies the inference by replacing difficult terms in the model by conjugate terms in the inference network.

The additional parameters $\boldsymbol{\phi}_{\text{PGM}}$ do not lead to a massive increase in parameters since the number of global parameters $\boldsymbol{\phi}_{\text{PGM}}$ is usually quite small.

Finally, using the above inference network, we can now specify the overall posterior approximation by assuming a mean-field approximation among $\mathbf{x}$ and $\boldsymbol{\theta}$:

$$p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y}) \approx q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})q(\boldsymbol{\theta}|\boldsymbol{\lambda}), \tag{5}$$

where $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ is a fixed-form minimal exponential-family distribution parametrized by λ.

Below, we give two examples of inference networks and show how to compute $Z(\boldsymbol{\phi})$.

Mixture Models: As described earlier, we can easily construct a conjugate-structure inference network where the normalizing constant is available in closed form. An example is shown below where the first component is an inference network as in VAE, while the second and third components correspond to a GMM:

$$q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi}) = \frac{1}{\mathcal{Z}(\boldsymbol{\phi})} \prod_{n=1}^{N} \left[\mathcal{N}\left(\mathbf{x}_n|\mathbf{m}_{y_n}, \mathbf{V}_{y_n}\right) \prod_{k=1}^{K} \bar{\pi}_k \mathcal{N}(\mathbf{x}_n|\bar{\boldsymbol{\mu}}_k, \bar{\boldsymbol{\Sigma}}_k)^{z_{nk}} \right], \tag{6}$$

where $\mathbf{m}_{y_n} := \mathbf{m}(\mathbf{y}_n, \boldsymbol{\phi}_{\text{NN}})$ and $\mathbf{V}_{y_n} := \mathbf{V}(\mathbf{y}_n, \boldsymbol{\phi}_{\text{NN}})$ are mean and covariance parameterized by a DNN with parameter $\boldsymbol{\phi}_{\text{NN}}$, $\boldsymbol{\phi}_{\text{PGM}} := \{\bar{\boldsymbol{\psi}}_k, \bar{\pi}_k\}_{k=1}^{K}$ and $\bar{\boldsymbol{\psi}}_k = \{\bar{\boldsymbol{\mu}}_k, \bar{\boldsymbol{\Sigma}}_k\}$ are the parameters on the PGM part in the inference network. The increase in number of parameter is only $O(K)$ which is usually much smaller than N. The graphical model of this inference network is shown in Figure 1b, where we clearly see that the structured inference network mimics the structure of the model.

An attractive property of the conjugate-structure inference network is that its normalizing constant is available in closed-form. In this case, we can obtain the following closed-form solution as described in Bishop (2006):

$$\mathcal{Z}(\boldsymbol{\phi}) := \sum_{n=1}^{N} \sum_{k=1}^{K} \bar{\pi}_k \, \mathcal{N}(\mathbf{m}_{y_n}|\bar{\boldsymbol{\mu}}_k, \mathbf{V}_{y_n} + \bar{\boldsymbol{\Sigma}}_k) \tag{7}$$

The above inference network is valid even when the mixture components are non-Gaussian, e.g., when they follow the Student's t distribution. For such cases, our inference network provides an efficient way to perform by replacing the non-Gaussian distribution by a Gaussian one.

State-Space Models: In this case, we can use the following conjugate-structure inference network:

$$q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi}) = \frac{1}{\mathcal{Z}(\boldsymbol{\phi})} \left[\prod_{n=1}^{N} \mathcal{N}\left(\mathbf{x}_n|\mathbf{m}_{y_n}, \mathbf{V}_{y_n}\right) \mathcal{N}(\mathbf{x}_n|\mathbf{A}\mathbf{x}_{n-1}, \mathbf{Q}) \right] \mathcal{N}(\mathbf{x}_0|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \tag{8}$$

where $\mathbf{m}_{y_n}$ and $\mathbf{V}_{y_n}$ are defined similarly to the mixture-model case, and the PGM parameters are $\boldsymbol{\phi}_{\text{PGM}} := \{\bar{\mathbf{A}}, \bar{\mathbf{Q}}, \bar{\boldsymbol{\mu}}_0, \bar{\boldsymbol{\Sigma}}_0\}$. The graphical model of this inference network is shown in Figure 1d. The normalizing constant for this inference network can be computed in closed-form. To see this, note that the first term can be written as a Gaussian measurement $\mathcal{N}(\mathbf{x}_n|\mathbf{m}_{y_n}, \mathbf{V}_{y_n}) := \mathcal{N}(\mathbf{m}_{y_n}|\mathbf{x}_n, \mathbf{V}_{y_n})$, i.e., $\mathbf{m}_{y_n}$ is a Gaussian measurement of $\mathbf{x}_n$. Therefore, (8) is a linear dynamical system where the marginal likelihood (which is equal to $\mathcal{Z}(\boldsymbol{\phi})$, can be computed using the forward-backward algorithm in $O(N)$ (a.k.a. Kalman filter) Bishop (2006).

Here again, when the dynamical system in (3) is non-linear, we can still use the inference network (8). Therefore, the structured inference network is simplifies the inference by replacing the nonlinear terms in the model by linear ones.

4 Variational Message Passing

In the previous section, we proposed a conjugate-structure inference network. Now, we describe a variational message-passing algorithm to perform efficient inference with it. An attractive property of our algorithm is that it enables natural-gradient updates for the PGM part, while its messages reduce to stochastic gradients when applied to deep neural networks. Our algorithm is an extension of the method proposed by and we show that our inference network plays an important role in this extension.

Our goal is to maximize the following lower bound:

$$\log p(\mathbf{y}) \geq \mathbb{E}_q \left[\log \frac{p(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta})}{q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi}) q(\boldsymbol{\theta}|\boldsymbol{\lambda})} \right] := L(\boldsymbol{\phi}, \boldsymbol{\lambda}) \tag{9}$$

This can further be expanded as shown below:

A straightforward solution is to use stochastic-gradient methods, e.g., RMSprop or Adam. These methods are suitable for estimating deterministic parameters $\boldsymbol{\phi}$, however they may be inefficient to estimate $\boldsymbol{\lambda}$. This is because $\boldsymbol{\lambda}$ is the parameter of a distribution and SG methods ignore this fact while optimizing. A preferred way to optimize with respect to $\boldsymbol{\lambda}$ is to use natural-gradient methods such as stochastic variational inference (Hoffman et al., 2013). Natural gradients exploit the geometry of the probabilistic model and show faster convergence. For mean-field variational inference, they are also easier to compute and updates can be performed locally at a node within the message-passing framework. SG methods cannot exploit nice properties of natural-gradients and might be slow to converge.

An alternative is to use natural-gradients for both $\boldsymbol{\phi}$ and $\boldsymbol{\lambda}$, but this is computationally intensive because a direct computation of the Fisher matrix for deterministic parameters is difficult. We could also simply apply natural-gradient updates to $\boldsymbol{\lambda}$ while using SG methods for $\boldsymbol{\phi}$, but this approach is not principled and step-size selection might be difficult due to the interaction between these different types of updates.

We propose a principled approach to solve this problem. The benefit of our approach is that we can automatically recover SG updates for $\boldsymbol{\phi}$ as a special case, while enabling natural-gradient updates for $\boldsymbol{\lambda}$. The key idea is to associate $\boldsymbol{\phi}$ with a Gaussian distribution $q(\boldsymbol{\phi}) := \mathcal{N}(\boldsymbol{\phi}|\boldsymbol{\mu}_\phi, \text{diag}(\boldsymbol{\sigma}_\phi^2))$ where $\boldsymbol{\mu}_\lambda$ and $\boldsymbol{\sigma}_\lambda$ are vectors of mean and standard deviation. Given this distribution, we maximize the following lower bound:

$$\bar{\mathcal{L}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\lambda}) := \mathbb{E}_{q(\phi)}[\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi})]. \tag{10}$$

This approach is known as variational optimization (Staines & Barber, 2012) and it works due to the following inequality: $\max \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\lambda}) \geq \mathbb{E}_{q(\phi)}[\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\lambda})]$, i.e., the maximum of a function is lower bounded by its expectation. Therefore, maximizing the expectation also maximizes the function. We will show that in our algorithm variances slowly shrink to zero as the algorithm converges and therefore the resulting algorithm converges to a local minima of $\mathcal{L}$ even though we optimize $\bar{\mathcal{L}}$. The good news is that now all of the optimization parameters, namely $\boldsymbol{\lambda}, \boldsymbol{\mu}$, and $\boldsymbol{\Sigma}$ are parameters of distributions. Therefore, we can use natural gradient updates.

A final step to obtain our message-passing algorithm is the choice of a natural-gradient method. The approach of SVI only works for conjugate models, but the DNN likelihood in our network is non-conjugate. We will therefore use a method proposed by Khan & Lin (2017) which is a generalization

of SVI. In this method, the messages for the conjugate part of the model reduces to SVI-like updates, but the messages for the rest of the model use stochastic-gradients. We now give detailed updates for the parameter $q(\boldsymbol{\theta}_{\text{PGM}}|\boldsymbol{\lambda}_{\text{PGM}})$, $q(\boldsymbol{\theta}_{\text{NN}}|\boldsymbol{\lambda}_{\text{NN}})$, and $q(\boldsymbol{\phi}|\boldsymbol{\mu}_\phi, \boldsymbol{\sigma}^2_\phi)$.

Update for $q(\boldsymbol{\theta}_{\text{PGM}}|\boldsymbol{\lambda}_{\text{PGM}})$: We assume that $p(\boldsymbol{\theta}_{\text{PGM}})$ and $p(\mathbf{x}|\boldsymbol{\theta}_{\text{PGM}})$ are conjugate to each other, although the algorithm can easily be modified when the relationship is non-conjugate. Conjugacy implies that the two distribution can be rewritten as follows:

$$p(\boldsymbol{\theta}_{\text{PGM}}) = h(\boldsymbol{\theta}_{\text{PGM}}) \exp\left[\langle \mathbf{T}(\boldsymbol{\theta}_{\text{PGM}}), \boldsymbol{\eta}\rangle - A(\boldsymbol{\eta})\right], \tag{11}$$

$$p(\mathbf{x}|\boldsymbol{\theta}_{\text{PGM}}) = \exp\left[\langle \mathbf{T}(\boldsymbol{\theta}_{\text{PGM}}), \boldsymbol{\eta}_{x\theta}(\mathbf{x})\rangle - f_x(\mathbf{x})\right], \tag{12}$$

where $\boldsymbol{\eta}$ is the natural parameter, $\mathbf{T}$ is the sufficient statistics, $\langle\cdot,\cdot\rangle$ is an inner product, h is the base measure, $A(\boldsymbol{\eta})$ is the log-partition function, and $\boldsymbol{\eta}_{x\theta}$ and f_x are functions that depend on $\mathbf{x}$ only. The neighbor of $\boldsymbol{\theta}_{\text{PGM}}$ is just $\mathbf{x}$, therefore using Algorithm 2 of Khan & Lin (2017), we can write the natural-gradient updates as follows:

$$\boldsymbol{\lambda}_{\text{PGM}} \leftarrow (1-\beta)\boldsymbol{\lambda}_{\text{PGM}} + \beta\, \mathbb{E}_{q(x|y,\phi)q(\phi)}\left[\boldsymbol{\eta} + \boldsymbol{\eta}_{x\theta}(x)\right] \tag{13}$$

where $\beta > 0$ is a step-size. This update is simple and does not require any gradient computations. To compute the expectation we simply generate samples $\mathbf{x}_*$ from the inference network. Given this sample, the computation of $\boldsymbol{\eta}_{x\theta}$ is equivalent to a variational inference in the model $p(\mathbf{x}_*, \boldsymbol{\theta}_{\text{PGM}})$ which can be performed using standard methods such as SVI and VMP. Therefore we can reuse existing software to perform this update.

Update for $q(\boldsymbol{\theta}_{\text{NN}}|\boldsymbol{\lambda}_{\text{NN}})$: To derive the updates, we will assume that the prior of all elements of $\boldsymbol{\theta}_{\text{NN}}$ is a Gaussian with mean μ_0 and variance σ_0^2. Since the number of parameters for DNNs is usually large, we will use a mean-field approximation. We will use a Gaussian approximation which makes sense since the prior is Gaussian too. For simplicity, we show updates only for the i'th element of $\boldsymbol{\theta}_{\text{NN}}$. We denote the mean and variance of this element by μ_i and σ_i^2 respectively.

The factors that involve $\boldsymbol{\theta}_{\text{NN}}$ in the joint distribution (1) are $p(\boldsymbol{\theta}_{\text{NN}})$ and $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_{\text{NN}})$. To derive the updates using Algorithm 2 of Khan & Lin (2017), we first note that the natural parameter of a Gaussian is equal to μ_i/σ_i^2. Using this and a few straightforward manipulations to express the gradients in terms of μ_i and σ_i^2, we get the following updates:

$$\sigma_i^{-2} \leftarrow (1-\beta)\sigma_i^{-2} - \beta\left[1/\sigma_0^2 + \nabla_{\sigma_i^2}\mathbb{E}_{q(\theta)q(\phi)q(x|y,\phi)}\left(\log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_{\text{NN}})\right)\right] \tag{14}$$

$$\mu_i \leftarrow \mu_i - \beta\sigma_i^2\left[\mu_0/\sigma_0^2 + \nabla_{\mu_i}\mathbb{E}_{q(\theta)q(\phi)q(x|y,\phi)}\left(\log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_{\text{NN}})\right)\right] \tag{15}$$

These gradients can be approximated by using samples from the distribution and by using the reparameterization trick. These updates are very similar to adaptive-gradient methods because the update for the mean is scaled by the variance which acts as the scaling vector in the adaptive methods. There are a two small differences. First, a regularization term μ_0/σ_0^2 and $1/\sigma_0^2$ is added. Second, gradient with respect to variance is computed. Third gradient are computed at a randomly sampled location from q. In our experiments, this does not make a big difference and our method converges at a very similar rate as the adaptive gradient methods.

Update for $q(\boldsymbol{\phi}|\boldsymbol{\mu}_\phi, \boldsymbol{\sigma}^2_\phi)$: To derive these updates, we note that the neighbors of $\boldsymbol{\phi}$ are the factors that use $\mathbf{x}$, i.e., $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_{\text{NN}})$, $p(\mathbf{x}|\boldsymbol{\theta}_{\text{PGM}})$ and $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})$ itself. It is easier to derive the update by simply taking the derivative of the lower bound as a function of $\boldsymbol{\phi}$ shown below:

$$\bar{\mathcal{L}}(\boldsymbol{\mu}_\phi, \boldsymbol{\sigma}^2_\phi) = \mathbb{E}_{q(\theta)q(\phi)q(x|y,\phi)}\left[\log \frac{p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_{\text{NN}})p(\mathbf{x}|\boldsymbol{\theta}_{\text{PGM}})}{q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})}\right] \tag{16}$$

Since the distribution is Gaussian we can use updates very similar to (14)-(15). To compute the gradient we can use reparameterization trick since a sample from $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})$ is a function of $\boldsymbol{\mu}_\phi$ and $\boldsymbol{\sigma}^2_\phi$. The updates again resemble adaptive-gradient methods and can be easily obtained.

Summary of the method: The method of Khan & Lin (2017) is guaranteed to converge under stochastic SVI like updates where a local variable update is followed by the update of global variable. Since we are using an inference network for the local variables, this is equivalent to generating samples from $q(\mathbf{x}|\mathbf{y}, \boldsymbol{\phi})$. Using the generated samples we can either perform the above three updates in parallel of use a sequential mode. It is also allowed to randomly pick a global variable and update them.

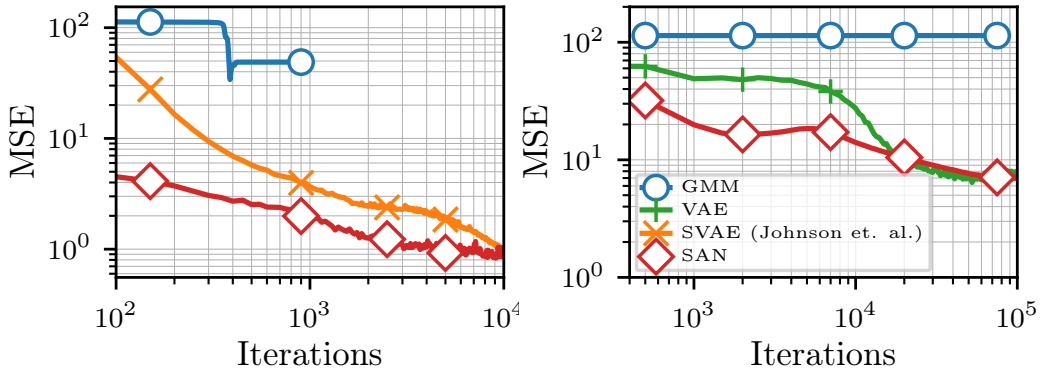


Figure 2: Reconstruction mean-squared error (MSE) computed on the Pinwheel dataset (left) and the Auto dataset (right). The performance has been measured every 500 iterations but the markers are shown only at iteration 500, 2000, 7000 and 75000. In the left figure, we compare to SVAE and GMM, where we see that SAN converges faster than SVAE, although both methods achieve similar error but perform better than GMM. In the right figure, we compare to VAE, where we see very similar trends. The performance of the GMM is represented as a constant because training converges after a few iterations already.

5 Experiments and Results

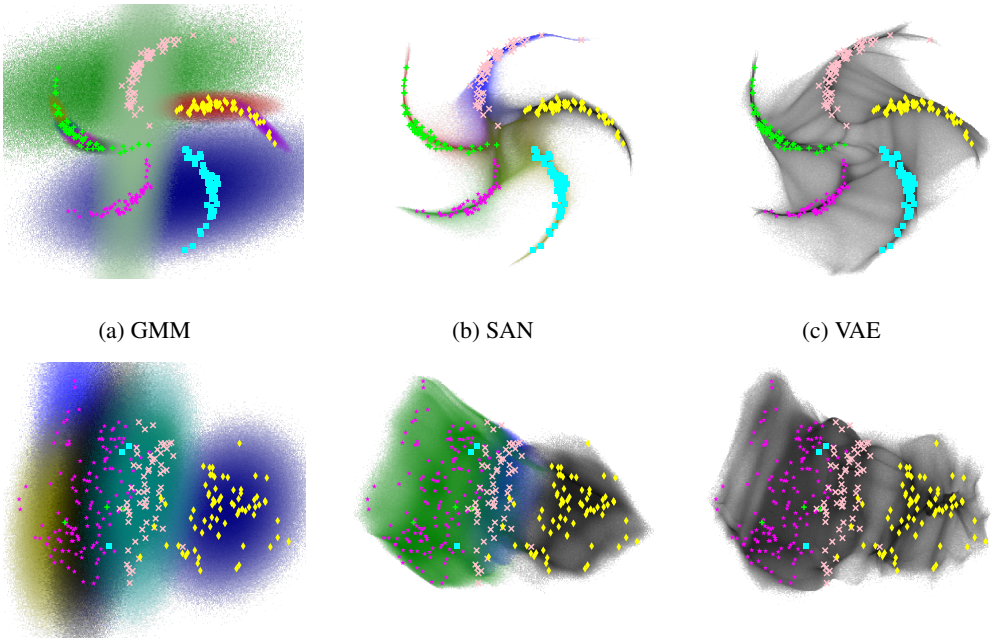
The main goal of our experiment is to show that our SAN algorithm gives similar results to the method of Johnson et al. (2016). The advantage of our algorithm over the method of Johnson et al. (2016) is that our algorithm is simpler and more general than their method, and does not involve a double-loop procedure. For this reason, we apply our algorithm to the two examples considered in Johnson et al. (2016), namely the latent Gaussian mixture model (GMM) and latent linear dynamical system (LDS). In this section, we show results on GMM (see details of the model in in section 3). An additional result for LDS is included in Appendix A. Our results show that the SAN algorithm can learn complex representations with interpretable structure, similar to the method of Johnson et al. (2016).

For the GMM case, we compare SAN to three algorithms: first, the standard variational EM algorithm applied to GMM (referred to as 'GMM'), second, the VAE approach of Kingma & Welling (2013), and third, the SVAE approach of Johnson et al. (2016).

We use two datasets. The first dataset is the synthetic two-dimensional Pinwheel dataset ($N = 5000$ and $D = 2$), and the second dataset is the Auto dataset ($N = 392$ and $D = 6$, available in the UCI repository). The latter contains information about cars. The dataset also contains a five-class label which indicates the number of cylinders in a car. We use these labels to validate our results. For both datasets we use 70% data for training.

For all methods, we tune their stepsizes, the number of mixture components, and the latent dimensionality on a validation set. We train the GMM baseline on the full dataset, and, for VAE and SVAE, we use stochastic minibatches of size 64. Our neural-network likelihood consists of two layers with 50 hidden units and an output layer of dimensionality 6 for the Auto dataset and 2 for the Pinwheel dataset.

In Figure 2, we compare the performance during training. In the left figure, we compare to SVAE and GMM, where we see that SAN converges faster than SVAE. Both methods achieve similar performance upon convergence and perform better than GMM. In the right figure, we compare to VAE and observe similar trends. In this figure, the performance of GMM is represented as a constant because training converges after a few iterations already. We found that the implementation provided by Johnson et al. (2016) does not perform well on the Auto dataset which is why we have not included it in the comparison. We also compared test log-likelihood and imputation error which show very similar trends.



(a) GMM (b) SAN (c) VAE

(d) GMM (e) SAN (f) VAE

Figure 3: These plots show the learned representations $p(\mathbf{y})$ in the background of the Pinwheel dataset (3a, 3b, 3c) and the Auto dataset (3d, 3e, 3f). In the foreground, we show 300 data samples which are colored according to the true labels (not used during the training). Different colors signify a mixture component and color intensities indicate the density. We use $K = 10$ mixture components to train all models. For the Auto dataset, we show only the first two principle components.

In Figure 3, we show samples generated from the learned distribution $p(\mathbf{y})$ for all methods. On the top of this plot, we show the data with true labels. These labels were not used during training. The plots (a)-(c) show results on Pinwheel dataset, while plots (d)-(e) shows results on the Auto dataset. For the Auto dataset, each label corresponds to the number of cylinders present in a car. We observe that SAN can learn meaningful partition of the data space, while VAE does not posses this capability. We also observe that SAN and VAE can learn flexible patterns while GMM does not do so.

These results show that with SAN we are able to combine complementary characteristics of VAE and GMM. For both datasets, our method learns flexible representations containing meaningful structural information.

6 Discussion and Conclusion

We propose a message-passing algorithm for the models that use both deep networks and probabilistic graphical models. We discussed the challenges involved when designing an algorithm to exploit the structural properties. Our algorithm can automatically adjust its messages and adapt it to the type of the node. For the DNN part, the messages reduce to computation of stochastic gradients, while on the PGM part, the messages are equal to natural gradients. Experimental comparisons confirm that our method gives similar results to existing methods, while using simpler updates.

Our experimental results in this paper are limited to small scale data. We found that it is non-trivial to implement a message-passing framework that goes well with the deep learning framework. Our current implementation is based on AutoGrad and TensorFlow, both of which are slow due to additional communication overhead due to messages passing. We are going to pursue this direction in the future and investigate good platforms to integrate the capabilities of these two different flavors of algorithms.

REFERENCES

Evan Archer, Il Memming Park, Lars Buesing, John Cunningham, and Liam Paninski. Black box variational inference for state space models. *arXiv preprint arXiv:1511.07367*, 2015.

Matthew James Beal. *Variational algorithms for approximate Bayesian inference*. University of London London, 2003.

Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, pp. 2199–2207, 2016.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems*, pp. 2946–2954, 2016.

Mohammad Emtiyaz Khan and Wu Lin. Conjugate-computation variational inference: Converting variational inference in non-conjugate models to inferences in conjugate models. *arXiv preprint arXiv:1703.04265*, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Rahul G Krishnan, Uri Shalit, and David Sontag. Deep Kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pp. 306–351. IEEE Press, 2001.

T. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2001.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah D. Goodman. Neurally-guided procedural models: Learning to guide procedural models with deep neural networks. *CoRR*, abs/1603.06143, 2016. URL `http://arxiv.org/abs/1603.06143`.

Joe Staines and David Barber. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

John Winn and Christopher M Bishop. Variational message passing. *Journal of Machine Learning Research*, 6(Apr):661–694, 2005.

A RESULTS FOR LATENT LINEAR DYNAMICAL SYSTEM

In this experiment, we apply our SAN algorithm to the latent LDS discussed in Section 3. For comparison, we compare our method, Structured Variational Auto-Encoder (SVAE) (Johnson et al., 2016), and LDS on the Dot dataset used in Johnson et al. (2016). Our results show that our method achieves comparable performance to SVAE. For LDS, we perform batch learning for all model parameters using the EM algorithm. For SVAE and SAN, we perform mini-batch updates for all model parameters. We use the same neutral network architecture as in Johnson et al. (2016), which contains two hidden layers with tanh activation function. We repeat our experiments 10 times and measure model performance in terms of the following mean absolute error for τ-steps ahead prediction. The error measures the absolute difference between the ground truth and the generative outputs by averaging across generated results.

$$\sum_{n=1}^{N}\sum_{t=1}^{T-\tau}\frac{1}{N(T-\tau)d}\left\{||\mathbf{y}^{*}_{t+\tau,n}-\mathbb{E}_{p(y_{t+\tau,n}|y_{1:t,n})}\left[\mathbf{y}_{t+\tau,n}\right]||_1\right\} \tag{17}$$

where N is the number of testing time series with T time steps, d is the dimensionality of observation $\mathbf{y}$, and observation $\mathbf{y}^{*}_{t+\tau,n}$ denotes the ground-truth at time step $t+\tau$.

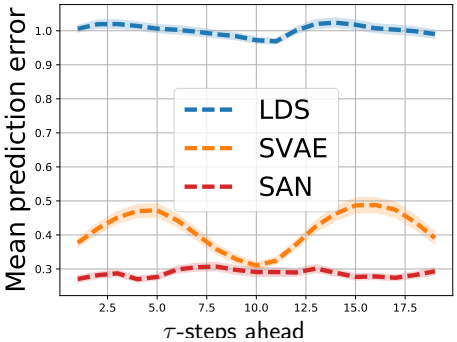


Figure 4: Prediction error, where shadows denote the standard errors across 10 runs

From Figure 4, we can observe that our method performs as good as SVAE and outperforms LDS. Our method is slightly robust than SVAE. In Figure 5, there are generated images obtained from all methods. From Figure 5, we also see that our method performs as good as SAVE and is able to recover the ground-truth observation.

(a) SAN (b) SVAE (c) LDS (d) Ground Truth

Figure 5: Generated images, where each column of pixels represents an observation, each row of pixels represents one time step, and each vertical white line denotes the first time step to generate images