Transforming Variational Autoencoders

Nathan Kong
University of Toronto
nathan.kong@mail.utoronto.ca

Jingyao (Jason) Li
University of Toronto
jingyao.li@mail.utoronto.ca

Abstract

In variational inference, the approximate posterior distribution that is chosen is very important. It needs to be computationally tractable, yet flexible enough to approximate the true posterior. In this paper, we discuss an application of variational inference in dimensionality reduction. We experiment with the variational autoencoder (VAE), which was developed by Kingma and Welling (2013), by comparing two different variational inference methods. The first method is the vanilla VAE and the second method improves variational inference by introducing normalizing flows, developed by Rezende and Mohamed (2015). Normalizing flows increase the complexity of an initial simple distribution, so that more complex true posteriors can potentially be approximated.

1 Introduction

Nowadays, with increasingly large amounts of data, making posterior inferences is intractable since the evidence in Bayes' Rule consists of a computationally intractable integral. Stochastic variational inference was developed that makes this inference more tractable, by turning the inference problem into an optimization problem. In variational inference, an intractable posterior distribution is approximated by a simpler probability distribution, whose parameters are optimized.

Unfortunately, extremely complex posterior distributions may not be successfully approximated using such simple distributions, so novel methods must be developed to improve the approximations. In this paper, we discuss various methods that improve posterior distribution approximations and also compare the performance of two different methods of variational inference.

2 Formal Description

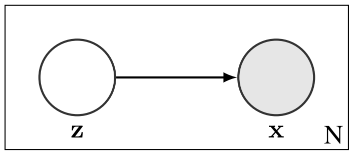


Figure 1: Probabilistic graphical model with latent variables, $\mathbf{z}$, and observed variables, $\mathbf{x}$.

From a probabilistic graphical model perspective, we have a latent space, governed by $\mathbf{z}$, and an observed space, which are our data, $\mathbf{x}$. The observed variables depend on the latent variables. Figure 1 illustrates this model. Using this framework, the joint density is: $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})$. $p(\mathbf{z})$ is the prior over the latent variables and $p(\mathbf{x} \mid \mathbf{z})$ is the likelihood of the data given the latent variables.

2.1 Variational Lower Bound

In order to obtain the marginal likelihood $p(\mathbf{x})$, we must integrate over $\mathbf{z}$, which is intractable. So, a posterior distribution, $q(\mathbf{z} \,|\, \mathbf{x})$, is introduced allowing us to obtain a lower bound on the marginal likelihood:

$$\begin{aligned}
\log p(\mathbf{x}) &= \log \int_{\mathbf{z}} p(\mathbf{x} \,|\, \mathbf{z})\, p(\mathbf{z})\, \mathrm{d}\mathbf{z} && (1)\\
&= \log \int_{\mathbf{z}} \frac{q(\mathbf{z} \,|\, \mathbf{x})}{q(\mathbf{z} \,|\, \mathbf{x})} p(\mathbf{x} \,|\, \mathbf{z})\, p(\mathbf{z})\, \mathrm{d}\mathbf{z} && (2)\\
&= \log \left(\mathbb{E}_q \left[\frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z} \,|\, \mathbf{x})} \right] \right) && (3)\\
&\geq \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z}) \right] - \mathbb{E}_q \left[\log q(\mathbf{z} \,|\, \mathbf{x}) \right] && (4)\\
&= \log p(\mathbf{x}) - \mathbb{D}_{\mathrm{KL}} \left[q(\mathbf{z} \,|\, \mathbf{x}) \,||\, p(\mathbf{z} \,|\, \mathbf{x}) \right] && (5)\\
&= -\mathbb{D}_{\mathrm{KL}} \left[q(\mathbf{z} \,|\, \mathbf{x}) \,||\, p(\mathbf{z}) \right] + \mathbb{E}_{q(\mathbf{z} \,|\, \mathbf{x})} \left[\log p(\mathbf{x} \,|\, \mathbf{z}) \right] && (6)
\end{aligned}$$

Equation 4 follows from Equation 3 by applying Jensen's inequality. Equation 6 is known as the variational lower bound, which we want to maximize. Note that from Equation 5, maximizing the lower bound minimizes the Kullback-Leibler (KL) divergence between the approximate posterior and the true posterior and maximizes the marginal likelihood since the KL divergence is always positive. We want $q(\mathbf{z} \,|\, \mathbf{x})$ to be computationally tractable, but flexible enough to be able to match $p(\mathbf{z} \,|\, \mathbf{x})$ such that the KL divergence is close to 0.

2.2 Vanilla VAE

In an autoencoder, there are two main stages: the encoder stage and the decoder stage, which are both neural networks. The input to the encoder, or recognition, stage is the high dimension feature vector that we want to reduce into a space with lower dimensionality. In a VAE, the output of the encoder stage are parameters to the approximate posterior, $q(\mathbf{z} \,|\, \mathbf{x})$. If $q(\mathbf{z} \,|\, \mathbf{x})$ is a Gaussian, the parameters that are output would be the mean, $\boldsymbol{\mu}$, and the variance, $\boldsymbol{\sigma}^2$. In this case, the covariance matrix is a diagonal matrix. So, both $\boldsymbol{\mu} \in \mathbb{R}^D$ and $\boldsymbol{\sigma}^2 \in \mathbb{R}^D$, where D is the dimension of the Gaussian.

The inputs to the decoder, or generator, are sampled values, $\mathbf{z}$, from the distribution, $q(\mathbf{z} \,|\, \mathbf{x})$. For $q(\mathbf{z} \,|\, \mathbf{x}) \sim \mathcal{N}(\mathbf{z} \,|\, \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, the sampled values are: $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$. The output of the decoder stage is a feature vector which has the same dimension as that of the input vector. They are parameters to $p(\mathbf{x} \,|\, \mathbf{z})$.

In order to train the VAE, we want to minimize the negative variational lower bound (negative of Equation 6). It is: $\mathbb{D}_{\mathrm{KL}} \left[q(\mathbf{z} \,|\, \mathbf{x}) \,||\, p(\mathbf{z}) \right] - \mathbb{E}_{q(\mathbf{z} \,|\, \mathbf{x})} \left[\log p(\mathbf{x} \,|\, \mathbf{z}) \right]$. The first term measures how different the approximate posterior is from $p(\mathbf{z})$, so that a smaller value indicates a better approximation. The second term is the reconstruction error, which describes how faithful the reconstructed input is to the actual input. The loss function for the VAE is:

$$\mathcal{F}(\mathbf{x}) = -\frac{1}{2} \sum_{d=1}^{D} \left(1 + \log \boldsymbol{\sigma}_d^2 - \boldsymbol{\mu}_d^2 - \boldsymbol{\sigma}_d^2 \right) - \left(\sum_{k=1}^{K} \mathbf{x}_k \log \hat{\mathbf{x}}_k + (1 - \mathbf{x}_k) \log (1 - \hat{\mathbf{x}}_k) \right) \qquad (7)$$

where K is the dimension of the input vector and $\hat{\mathbf{x}}$ being the reconstructed input. This is computed for every sampled value of $\mathbf{z}_i \sim q(\mathbf{z} \,|\, \mathbf{x})$ and then averaged.

2.3 VAE with Normalizing Flow

In Rezende and Mohamed (2015), variational inference is improved by introducing normalizing flows. Recall that in variational inference, we want $q(\mathbf{z} \,|\, \mathbf{x})$ to be flexible enough to approximate the true posterior, $p(\mathbf{z} \,|\, \mathbf{x})$, since the true posterior can be extremely complicated. Figure 2 illustrates how inferences are made using normalizing flow and how output is generated.

Normalizing flow describes a series of transformations on the initial probability distribution. These transformations are invertible mappings and at the end of the series, a new probability distribution is obtained. The transformations, f, must be chosen such that they are smooth and invertible (i.e.

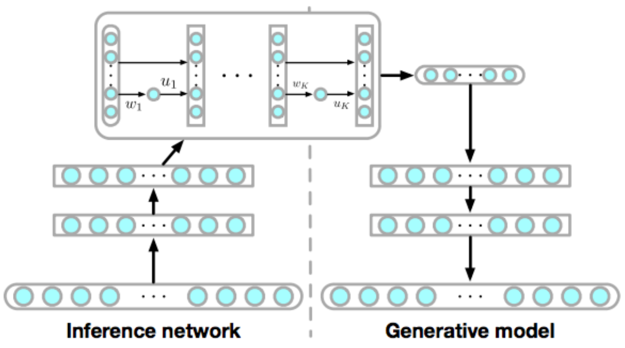


Figure 2: VAE with normalizing flow — flow diagram, from Rezende and Mohamed (2015).

$f^{-1} = g$, where $g(f(\mathbf{z})) = \mathbf{z}$). Also, if $\mathbf{z} \sim q(\mathbf{z})$, then the transformed random variable, $\mathbf{z}_1 = f(\mathbf{z})$, has the following distribution, $q(\mathbf{z}_1)$:

$$\mathbf{z}_1 \sim q(\mathbf{z}_1) = q(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1} \tag{8}$$

We can perform K transformations to the initial random variable, $\mathbf{z}_0$, to obtain the final random variable, $\mathbf{z}_K$, where:

$$\mathbf{z}_K = f_K(f_{K-1}(\cdots(f_2(f_1(\mathbf{z}_0))))) \tag{9}$$

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right| \tag{10}$$

The transformations, f, that are used are of the form:

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u} \cdot h(\mathbf{w}^\top \mathbf{z} + b) \tag{11}$$

where $\mathbf{u}, \mathbf{w} \in \mathbb{R}^D$, $b \in \mathbb{R}$, and h is a smooth non-linearity that is at least once differentiable. This is called a planar flow. We use $h(\cdot) = \tanh(\cdot)$ and $h'(\cdot) = 1 - \tanh^2(\cdot)$ in our experiments. Plugging this into Equation 10, we get:

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| 1 + \mathbf{u}_k^\top \psi_k(\mathbf{z}_{k-1}) \right| \tag{12}$$

where $\psi_k(\mathbf{z}) = h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{w}$. With these equations, the function we want to minimize becomes:

$$\mathcal{F}(\mathbf{x}) = \mathbb{E}_{q_0}\left[\log q_0(\mathbf{z}_0)\right] - \mathbb{E}_{q_0}\left[\log p(\mathbf{x}, \mathbf{z}_K)\right] - \mathbb{E}_{q_0}\left[\sum_{k=1}^{K} \log \left| 1 + \mathbf{u}_k^\top \psi_k(\mathbf{z}_{k-1}) \right|\right] \tag{13}$$

3 Related Work

Rezende and Mohamed (2015) also introduce another type of flow called radial flow. Radial flow differs from planar flow in the transformation function, f. The transformation is $f(\mathbf{z}) = \mathbf{z} + \beta \cdot h(\alpha, r)(\mathbf{z} - \mathbf{z}_0)$, where $\alpha \in \mathbb{R}^+$ and $\beta \in \mathbb{R}$. The changes to the initial distribution, q_0, differ between the two transformation functions. Planar flows apply contractions and expansions perpendicular to the hyperplane, $\mathbf{w}^\top \mathbf{z} + b$, whereas radial flows apply contractions and expansions radially around $\mathbf{z}_0$.

Inverse autoregressive flow, which is a part of the family of normalizing flows, was introduced by Kingma et al. (2016). In this flow, the series of transformations is initialized by sampling $\mathbf{z}_0 \sim q(\mathbf{z} \,|\, \mathbf{x})$, as before. Then, each successive $\mathbf{z}$ is computed using the transformation: $\mathbf{z}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1}$, where t indexes the transformations. After T transformations, the log distribution becomes: $\log q(\mathbf{z}_T \mid \mathbf{x}) = -\sum_{i=1}^{D}\left(\frac{1}{2}\epsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T}\log\sigma_{t,i}\right)$ The resulting log marginal probabilities proved to be larger than those computed using previous methods of variational inference.

Dinh et al. (2016) introduce another type of transformation called the real-valued non-volume preserving (real NVP) transformation. Similar to normalizing flow, Dinh et al. (2016) propose an invertible transformation from a simple latent distribution to a complex distribution of the form:

$$y_{1:d} = x_{1:d} \tag{14}$$

$$y_{d+1:D} = x_{d+1:D} \odot \exp\left(s(x_{1:d})\right) + t\left(x_{1:d}\right) \tag{15}$$

Using this mapping as f in Equation 10, we can compute the determinant and obtain $\log q_K(\mathbf{z}_K)$ accordingly. The difference between real NVP and normalizing flows is that real NVP only transforms a portion of the input, from dimension $d+1$ onwards, through the exponential of an affine linear transformation and element-wise multiplied with the input. They claim that the generated samples using this transformation performs better than autoregressive models such as PixelRNN.

4 Results and Comparison

4.1 Overview

We compared the performance of the two models on the MNIST dataset to see the effects of the normalizing flow layers. We used the same model and hyperparameters to achieve a fair comparison. The only difference between the models is the training loss, $\mathcal{F}$, as presented in Section 2, and the addition of normalizing flow layers from the latent sample to the generator network. We created 3 normalizing flow models, each with a different number of normalizing flow layers. The code for these experiments can be found online[1].

For each model, we trained them for 100 epochs of the dataset with the Adam optimizer. In order to approximate the expectations present in Equations 7 and 13, we used Monte Carlo estimation with a single sample of the latent variables per data point. For the latent space, we chose to parameterize it as a 20-dimensional Gaussian with mean and diagonal covariance as determined by the recognition network. Table 1 shows the various hyperparameters that were used to train the models.

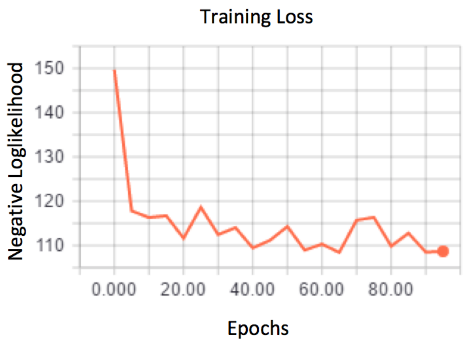
Table 1: Table of hyperparameters used to train the VAEs.

Models	Baseline	Normalizing Flow	Normalizing Flow	Normalizing Flow
Training Hyperparameters				
Optimizer	Adam with TensorFlow Defaults			
Learning Rate	0.001	0.001	0.001	0.001
Batch Size	128	128	128	128
Training Epochs	100	100	100	100
Latent Gaussian Dimensions	20	20	20	20
Recognition Network				
Hidden Layers	2	2	2	2
Hidden Units	500	500	500	500
Normalizing Flow Layers	0	3	5	8
Hidden Units Activation	ReLU	ReLU	ReLU	ReLU
Final Activation	tanh	tanh	tanh	tanh
Generator Network				
Hidden Layers	2	2	2	2
Hidden Units	500	500	500	500
Hidden Units Activation	ReLU	ReLU	ReLU	ReLU
Final Activation	sigmoid	sigmoid	sigmoid	sigmoid

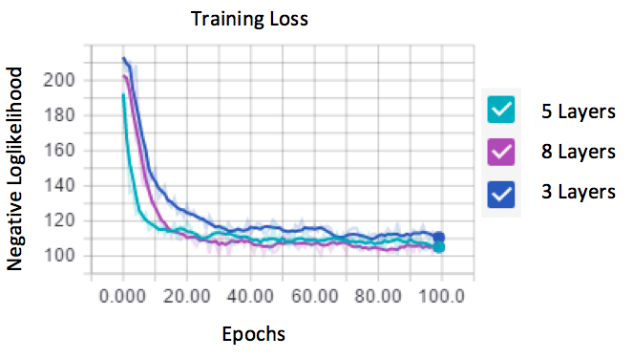
[1] `https://github.com/blisc/CSC412-Project`

4.2 Baseline (Vanilla) VAE

After training the baseline VAE to the MNIST dataset, the negative variational bound of the model was computed by averaging the loss across 10 samples of the latent space given each data point from the data set. The negative variational lower bound was approximately 109.23. The training loss is shown in Figure 3a.



(a) Loss curve for the vanilla VAE.



(b) Loss curve for the VAE with normalizing flows.

Figure 3: Loss curves for different VAE models.

4.3 VAE with Normalizing Flow

In the same manner as the baseline VAE, we approximated the log-likelihood using the lower bound as stated in Equation 13. In order to evaluate the performance of the normalizing flow layers, we trained 3 models with 3, 5 and 8 normalizing flow layers to see how much of an impact these layers have. We obtained negative variational bound values of 111.96, 105.09 and 107.67 respectively. The training loss for all 3 models is shown in Figure 3b.

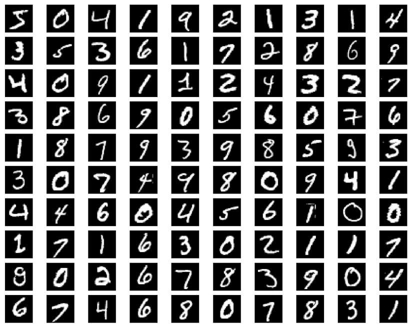
4.4 Model Comparison

Comparing the variational bound values of the 4 models, we see that the model with 8 normalizing flow layers performs the best, followed by the model with 5 layers. An interesting observation is that the model with 3 normalizing flow layers performs worse than the baseline model. We expected that the normalizing flows would have been able to better approximate the posterior than a model without those layers.

Figures 4, 5 and 6 show the reconstructions produced by each model. We took the first 100 images from the MNIST training set and plot their reconstruction. Qualitatively, it seems that the baseline model produces the sharpest images whereas the normalizing flow models produce blurrier images. This is the opposite of what we expected given the variational bounds discussed previously. However, because the loss functions for the two models are slightly different, the lower bound might not be the best measurement of image quality.

5 Limitations and Future Work

Our original goal was to explore the difference between vanilla VAEs and normalizing flow VAEs. As we explained in Section 2, normalizing flows should allow for more complex posteriors to be approximated and give our model more expressive power. However, as can seen from the qualitative samples, the addition of these normalizing flow layers does not always make our generator perform better. Perhaps one limitation of our review was the size of the model that we used. In Rezende and Mohamed (2015), they trained a model with 40 latent dimensions and up to 80 normalizing flow layers. In the interest of time, our models were much smaller. This decision might have limited the complexity of our posterior. Another issue might be due to the added complexity of the normalizing flow layers. It might have required more training iterations for the normalizing flow models to fully converge in comparison to the baseline model. Perhaps, training the normalizing flow models for longer might yield better results.


(a) Input images from MNIST dataset.

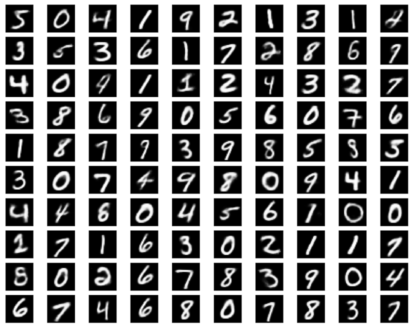
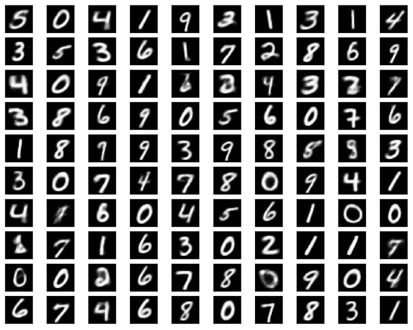

(b) Reconstructed images using the vanilla VAE.

Figure 4: Training images and reconstructed images using the vanilla VAE.


(a) Reconstructed images using the VAE with 3 normalizing flow layers.

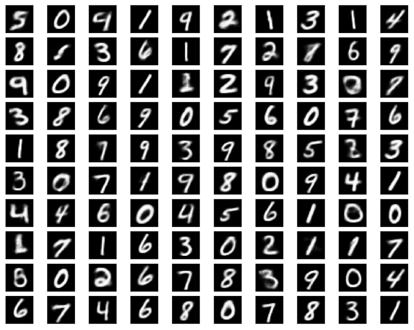

(b) Reconstructed images using the VAE with 5 normalizing flow layers.

Figure 5: Reconstructed images using the VAE with 3 and 5 normalizing flow layers.

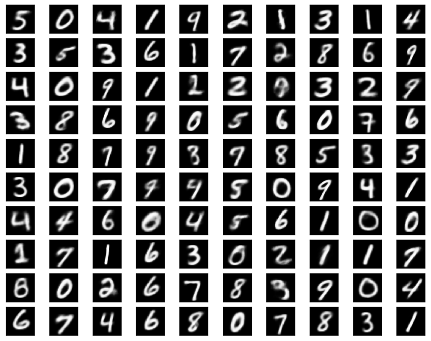


Figure 6: Reconstructed images using the VAE with 8 normalizing flow layers.

Another limitation of the normalizing flow models is that while they increase the complexity of the posterior, the authors only provide an empirical result to show that normalizing flow models can learn arbitrary complex distributions. While these transformations help construct more complex posterior distributions, there is no proof that these models can learn any type of posterior distribution. Thus, we are still stuck with an approximation of the true posterior.

In Section 3, we presented three other transformations that can be applied. A good next step to expand the review would be to try these transformations as well to see if they result in better generated images. If we had more time and computing resources, it would be a good idea to test some of the presented limitations by increasing model complexity and training the model for more epochs.

6 Conclusions

In our work, we described two models of VAEs: vanilla VAEs and VAEs with differing numbers of normalizing flow layers. We compared the training loss among the different VAE models and found that the model with the greatest number of normalizing flow layers performed the best quantitatively. This is what we expected since normalizing flow layers allow for greater expressivity in the posterior approximation. Therefore, by introducing various transformations to a simple initial posterior distribution, more complex posterior distributions can be approximated. However, it is worth noting that the generated samples produced by the normalizing flow models are more blurry than those from the vanilla VAE indicating possible limitations in our review.

References

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

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

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems 29*, pages 4743–4751. 2016.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, pages 1530–1538, 2015.