Supplementary

Code for the experiments in this paper can be found at: https://github.com/chriscremer/Inference-Suboptimality and https://github.com/lxuechen/inference-suboptimality.

7. Model Architectures and Training Hyperparameters

7.0.1. 2D VISUALIZATION

The VAE model of Fig. 2 uses a decoder $p(x|z)$ with architecture: $2 - 100 - 784$, and an encoder $q(z|x)$ with architecture: $784 - 100 - 4$. We use tanh activations and a batch size of 50. The model is trained for 3000 epochs with a learning rate of 10^{-4} using the ADAM optimizer (Kingma & Ba, 2014).

7.0.2. MNIST & FASHION-MNIST

Both MNIST and Fashion-MNIST consist of a training and test set with 60000 and 10000 datapoints respectively, where each datapoint is a 28x28 grey-scale image. We rescale the original images so that pixel values are within the range $[0, 1]$. For MNIST, We use the statically binarized version described by (Larochelle & Bengio, 2008). We also binarize Fashion-MINST *statically*. For both datasets, we adopt the Bernoulli likelihood for the generator.

The VAE models for MNIST and Fashion-MNIST experiments have the same architecture. The encoder has two hidden layers with 200 units each. The activation function is chosen to be the exponential linear unit (ELU, Clevert et al. (2015)), as we observe improved performance compared to tanh. The latent space has 50 dimensions. The generator is the reverse of the encoder. We follow the same learning rate schedule and train for the same amount of epochs as described by (Burda et al., 2016). All models are trained with the a batch-size of 100 with ADAM.

In the large encoder setting, we change the number of hidden units for the inference network to be 500, instead of 200. The warm-up models are trained with a linear schedule over the first 400 epochs according to Section 5.6.

The auxiliary variable of requires a couple distributions: $q(v_0|z_0)$ and $r(v_T|z_T)$. These distributions are both factorized Gaussians which are parameterized by MLP's with two hidden layers, 100 units each, with ELU activations.

The flow transformation $q(z_{t+1}, v_{t+1}|z_t, v_t)$ involves functions σ_1, σ_2, μ_1, and μ_2 from Eqn. 9 and 10. These also have two hidden layers with 100 units each and ELU units.

7.0.3. 3-BIT CIFAR

CIFAR-10 consists of a training and test dataset with 50000 and 10000 datapoints respectively, where each datapoint is a 32×32 RGB image. We rescale individual pixel values to be in the range $[0, 1]$. We then statically binarize the scaled pixel values by setting individual pixel values of channels to 1 if the rescaled value is greater than 0.5 and 0 otherwise. In this manner, we can model the observation with a factorized Bernoulli likelihood. We call this binarized CIFAR-10 dataset as *3-BIT CIFAR*, since 3 bits are required to encode each pixel, where 1 bit is needed for each of the channels. We acknowledge that such binarization scheme may reduce the complexity of the original problem, since originally 24 bits were required to encode a single pixel. Nevertheless, the 3-bit CIFAR dataset is still much more challenging compared MNIST and Fashion. This is because 784 bits are required to encode one MNIST/Fashion image, whereas for one 3-bit CIFAR image, 3072 bits are required. Most notably, we were able to validate our AIS estimates using BDMC with the simplified dataset. This, however, was not achievable in any reasonable amount of time with the original CIFAR-10 dataset.

For the latent variable, we use a 50-dimensional factorized Gaussian for $q(z|x)$. For all neural networks, ELU is chosen to be the activation function. The inference network consists of three 4 by 4 convolution layers with stride 2, batch-norm, and 64, 128, 256 channels respectively. Then a fully-connected layer outputs the 50-dimensional mean and log-variance of the latent variable. Similarly, the generator consists of a fully-connected layer outputting 256 by 2 by 2 tensors. Then three deconvolutional layers each with 4 by 4 filters, stride 2, batch-norm, and 128, 64, and 3 channels respectively. For the model with expressive inference, we use three normalizing flow steps, where the parametric functions in the flow and auxiliary variable distribution also take in a hidden layer of the encoder.

We use a learning rate of 10^{-3}. Warm-up is applied with a linear schedule over the first 50 epochs. All models are trained with a batch-size of 100 with ADAM. Early-stopping is applied based on the performance computed with the IWAE bound (k=1000) on the held-out set of 5000 examples from the original training set.

7.1. Inference Generalization

These models are trained with batch size 50 and latent dimension size of 20. The rest of the hyperparameters are equivalent to Section 7.0.2.

Architecture of q_{Flow}: The flow transformation involves functions σ_1, σ_2, μ_1, and μ_2 from Eqn. 9 and 10. Each function is an MLP with a 50 unit hidden layer and ELU activations. We apply this flow transformation twice.

Fig. 4 are the plots for the q_{AF} model. The transformations are the same as q_{Flow}, but rather than partitioning the latent variable, we introduce an auxiliary variable. The auxiliary variable also requires a reverse model $r(v|z)$ which is a factorized Gaussian parameterized by an MLP with a 50 unit hidden layer and ELU activations.

Comparing AF in Fig. 4 to Flow in Fig. 3, we see that the AF has a larger approximation gap. This increase is likely due to the KL $(q(v|z,x)\|r(v|x,z))$ term of the auxiliary variable lower bound from 2.2.2. This motivates also using expressive approximations for the reverse model $r(v|z)$.

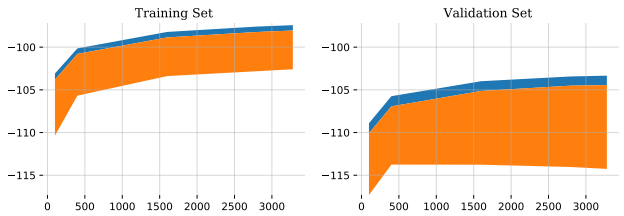

Figure 4. Gaps over epochs of the AF (auxiliary flow) model.

7.2. Influence of Flows On Amortization Gap Experiment

The aim of this experiment is to show that the parameters used for increasing the expressiveness of the approximation also contribute to reducing the amortization error. To show this, we train a VAE on MNIST, discard the encoder, then retrain two encoders on the fixed decoder: one with a factorized Gaussian distribution and the other with a parameterized 'flow' distribution. We use fixed decoder so that the true posterior is constant for both encoders. See 5.3 for the results and below for the architecture details.

The architecture of the decoder is: $D_Z - 200 - 200 - D_X$. The architecture of the encoder used to train the decoder is $D_X - 200 - 200 - 2D_Z$. The approximate distribution $q(z|x)$ is a factorized Gaussian.

Next, we describe the encoders which were trained on the fixed trained decoder. In order to highlight a large amortization gap, we employed a very small encoder architecture: $D_X - 2D_Z$. This encoder has no hidden layers, which greatly impoverishes its ability and results in a large amortization gap.

We compare two approximate distributions $q(z|x)$. Firstly, we experiment with the typical fully factorized Gaussian (FFG). The second is what we call a flow distribution. Specifically, we use the transformations of (Dinh et al., 2017). We also include an auxiliary variable so we don't need to select how to divide the latent space for the transformations. The approximate distribution over the latent z and auxiliary variable v factorizes as: $q(z,v|x) = q(z|x)q(v)$. The $q(v)$ distribution is simply a N(0,1) distribution. Since we're using a auxiliary variable, we also require the $r(v|z)$ distribution which we parameterize as $r(v|z)$: $[D_Z] - 50 - 50 - 2D_Z$. The flow transformation is the same as in Section 3.2, which we apply twice.

7.3. Computation of the Determinant for Flow

The overall mapping f that performs $(z,v) \mapsto (z',v')$ is the composition of two sheer mappings f_1 and f_2 that respectively perform $(z,v) \mapsto (z,v')$ and $(z,v') \mapsto (z',v')$. Since the Jacobian of either one of the sheer mappings is diagonal, the determinant of the composed transformation's Jacobian Df can be easily computed:

$$\begin{aligned}\det(Df) &= \det(Df_1)\det(Df_2) \\ &= \Big(\prod_{i=1}^{n}\sigma_1(z)_i\Big)\Big(\prod_{j=1}^{n}\sigma_2(v')_j\Big).\end{aligned}$$

7.4. Annealed Importance Sampling

Annealed importance sampling (AIS, Neal (2001); Jarzynski (1997)) is a means of computing a lower bound to the marginal log-likelihood. Similarly to the importance weighted bound, AIS must sample a proposal distribution $f_1(z)$ and compute the density of these samples, however, AIS then transforms the samples through a sequence of reversible transitions $\mathcal{T}_t(z'|z)$. The transitions anneal the proposal distribution to the desired distribution $f_T(z)$.

Specifically, AIS samples an initial state $z_1 \sim f_1(z)$ and sets an initial weight $w_1 = 1$. For the following annealing steps, z_t is sampled from $\mathcal{T}_t(z'|z)$ and the weight is updated according to:

$$w_t = w_{t-1}\frac{f_t(z_{t-1})}{f_{t-1}(z_{t-1})}.$$

This procedure produces weight w_T such that $\mathbb{E}[w_T] = \mathcal{Z}_T/\mathcal{Z}_1$, where Z_T and Z_1 are the normalizing constants of $f_T(z)$ and $f_1(z)$ respectively. This pertains to estimating the marginal likelihood when the target distribution is $p(x,z)$ when we integrate with respect to z.

Typically, the intermediate distributions are simply defined to be geometric averages: $f_t(z) = f_1(z)^{1-\beta_t} f_T(z)^{\beta_t}$,

where β_t is monotonically increasing with $\beta_1 = 0$ and $\beta_T = 1$. When $f_1(z) = p(z)$ and $f_T(z) = p(x, z)$, the intermediate distributions are: $f_i(x) = p(z)p(x|z)^{\beta_i}$.

Model evaluation with AIS appears early on in the setting of deep belief networks (Salakhutdinov & Murray, 2008). AIS for decoder-based models was also used by Wu et al. (2017).

7.5. Extra MNIST Inference Gaps

To demonstrate that a very small inference gap can be achieved, even with a limited approximation such as a factorized Gaussian, we train the model on a small dataset. In this experiment, our training set consists of 1000 datapoints randomly chosen from the original MNIST training set. The training curves on this small datatset are shown in Fig. 5. Even with a factorized Gaussian distribution, the inference gap is very small: the AIS and IWAE bounds are overlapping and the VAE is just slightly below. Yet, the model is overfitting as seen by the decreasing test set bounds.

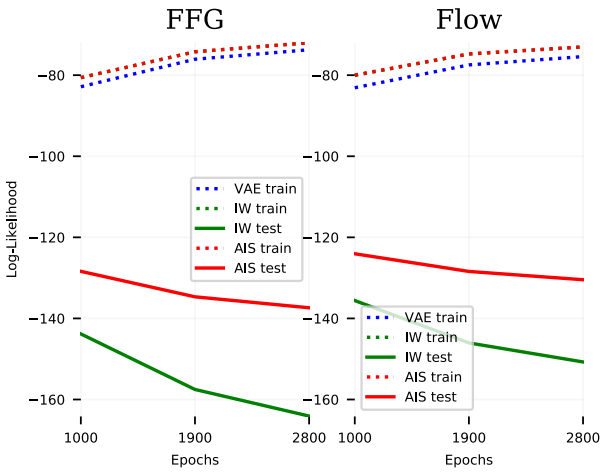

Figure 5. Training curves for a FFG and a Flow inference model on MNIST. AIS provides the tightest lower bound and is independent of encoder overfitting. There is little difference between FFG and Flow models trained on the 1000 datapoints since inference is nearly equivalent.