Supplementary Materials for Semi-Amortized Variational Autoencoders

A. Training Semi-Amortized Variational Autoencoders with Gradient Clipping

For stable training we found it crucial to modify Algorithm 1 to clip the gradients at various stages. This is shown in Algorithm 2, where we have a clipping parameter η. The $\text{clip}(\cdot)$ function is given by

$$\text{clip}(\mathbf{u}, \eta) = \begin{cases} \frac{\eta}{\|\mathbf{u}\|_2}\mathbf{u}\ , & \text{if } \|\mathbf{u}\|_2 > \eta \\ \mathbf{u}\ , & \text{otherwise} \end{cases}$$

We use $\eta = 5$ in all experiments. The finite difference estimation itself also uses gradient clipping. See https://github.com/harvardnlp/sa-vae/blob/master/optim_n2n.py for the exact implementation.

Algorithm 2 Semi-Amortized Variational Autoencoders with Gradient Clipping

```
Input: inference network $\phi$, generative model $\theta$,
       inference steps $K$, learning rate $\alpha$, momentum $\gamma$,
       loss function $f(\lambda, \theta, \mathbf{x}) = -\text{ELBO}(\lambda, \theta, \mathbf{x})$,
       gradient clipping parameter $\eta$
Sample $\mathbf{x} \sim p_{\mathcal{D}}(\mathbf{x})$
$\lambda_0 \leftarrow \text{enc}(\mathbf{x}; \phi)$
$v_0 \leftarrow 0$
for $k = 0$ to $K - 1$ do
    $v_{k+1} \leftarrow \gamma v_k - \text{clip}(\nabla_\lambda f(\lambda_k, \theta, \mathbf{x}), \eta)$
    $\lambda_{k+1} \leftarrow \lambda_k + \alpha v_{k+1}$
end for
$\mathcal{L} \leftarrow f(\lambda_K, \theta, \mathbf{x})$
$\overline{\lambda}_K \leftarrow \nabla_\lambda f(\lambda_K, \theta, \mathbf{x})$
$\overline{\theta} \leftarrow \nabla_\theta f(\lambda_K, \theta, \mathbf{x})$
$\overline{v}_K \leftarrow 0$
for $k = K - 1$ to $0$ do
    $\overline{v}_{k+1} \leftarrow \overline{v}_{k+1} + \alpha\overline{\lambda}_{k+1}$
    $\overline{\lambda}_k \leftarrow \overline{\lambda}_{k+1} - \text{H}_{\lambda,\lambda} f(\lambda_k, \theta, \mathbf{x})\overline{v}_{k+1}$
    $\overline{\lambda}_k \leftarrow \text{clip}(\overline{\lambda}_k, \eta)$
    $\overline{\theta} \leftarrow \overline{\theta} - \text{clip}(\text{H}_{\theta,\lambda} f(\lambda_k, \theta, \mathbf{x})\overline{v}_{k+1}, \eta)$
    $\overline{v}_k \leftarrow \gamma\overline{v}_{k+1}$
end for
$\frac{\text{d}\mathcal{L}}{\text{d}\theta} \leftarrow \overline{\theta}$
$\frac{\text{d}\mathcal{L}}{\text{d}\phi} \leftarrow \frac{\text{d}\lambda_0}{\text{d}\phi}\overline{\lambda}_0$
Update $\theta, \phi$ based on $\frac{\text{d}\mathcal{L}}{\text{d}\theta}, \frac{\text{d}\mathcal{L}}{\text{d}\phi}$
```

B. Experimental Details

For all the variational models we use a spherical Gaussian prior. The variational family is the diagonal Gaussian parameterized by the vector of means and log variances. For models trained with SVI the initial variational parameters are randomly initialized from a Gaussian with standard deviation equal to 0.1.

B.1. Synthetic Data

We generate synthetic data points according to the following generative process:

$$z_1, z_2 \sim \mathcal{N}(0, 1) \qquad \mathbf{h}_t = \text{LSTM}(\mathbf{h}_{t-1}, \mathbf{x}_t)$$
$$\mathbf{x}_{t+1} \sim \text{softmax}(\text{MLP}([\mathbf{h}_t, z_1, z_2]))$$

Here LSTM is a one-layer LSTM with 100 hidden units where the input embedding is also 100-dimensional. The initial hidden/cell states are set to zero, and we generate for 5 time steps for each example (so $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_5]$). The MLP consists of a single affine transformation to project out to the vocabulary space, which has 1000 tokens. LSTM/MLP parameters are randomly initialized with $\mathcal{U}(-1, 1)$, except for the part of the MLP that directly connects to the latent variables, which is initialized with $\mathcal{U}(-5, 5)$. This is done to make sure that the latent variables have more influence in predicting $\mathbf{x}$. We generate 5000 training/validation/test examples.

When we learn the generative model the LSTM is initialized over $\mathcal{U}(-0.1, 0.1)$. The inference network is also a one-layer LSTM with 100-dimensional hidden units/input embeddings, where the variational parameters are predicted via an affine transformation on the final hidden state of the encoder. All models are trained with stochastic gradient descent with batch size 50, learning rate 1.0, and gradient clipping at 5. The learning rate starts decaying by a factor of 2 each epoch after the first epoch at which validation performance does not improve. This learning rate decay is not triggered for the first 5 epochs. We train for 20 epochs, which was enough for convergence of all models. For SVI/SA-VAE we perform 20 steps of iterative inference with stochastic gradient descent and learning rate 1.0 with gradient clipping at 5.

B.2. Text

We use the same model architecture as was used in Yang et al. (2017). The inference network and the generative model are both one-layer LSTMs with 1024-dimensional

Inference Network	3-layer ResNet		2-layer MLP	
Model	VAE	SA-VAE	VAE	SA-VAE
Data Size: 25%	92.21 (0.81)	91.89 (2.31)	92.33 (0.27)	92.03 (1.32)
Data Size: 50%	91.38 (0.77)	91.01 (2.54)	91.40 (0.51)	91.10 (1.48)
Data Size: 75%	90.82 (1.06)	90.51 (2.07)	90.90 (0.45)	90.67 (1.34)
Data Size: 100%	90.43 (0.98)	90.05 (2.78)	90.56 (0.61)	90.25 (1.77)
1-layer PixelCNN	96.53 (10.36)	96.01 (10.93)	98.30 (8.87)	96.43 (10.14)
3-layer PixelCNN	93.75 (7.10)	93.16 (8.73)	94.45 (5.46)	93.55 (7.20)
6-layer PixelCNN	91.24 (3.25)	90.79 (4.44)	91.40 (2.06)	91.01 (3.27)
9-layer PixelCNN	90.54 (1.78)	90.28 (3.02)	90.72 (1.14)	90.34 (2.26)
12-layer PixelCNN	90.43 (0.98)	90.05 (2.78)	90.56 (0.61)	90.25 (1.77)

Table 4. Upper bounds on negative log-likelihood (i.e. negative ELBO) of VAE/SA-VAE trained on OMNIGLOT, where we vary the capacity of the inference network (3-layer ResNet vs 2-layer MLP). KL portion of the loss is shown in parentheses. (Top) Here we vary the training set size from 25% to 100%, and use a 12-layer PixelCNN as the generative model. (Bottom) Here we fix the training set size to be 100%, and vary the capacity of the generative model.

hidden states where the input word embedding is 512-dimensional. We use the final hidden state of the encoder to predict (via an affine transformation) the vector of variational means and log variances. The latent space is 32-dimensional. The sample from the variational posterior is used to initialize the initial hidden state of the generative LSTM (but not the cell state) via an affine transformation, and additionally fed as input (i.e. concatenated with the word embedding) at each time step. There are dropout layers with probability 0.5 between the input-to-hidden layer and the hidden-to-output layer on the generative LSTM only.

The data contains 100000/10000/10000 train/validation/test examples with 20000 words in the vocabulary. All models are trained with stochastic gradient descent with batch size 32 and learning rate 1.0, where the learning rate starts decaying by a factor of 2 each epoch after the first epoch at which validation performance does not improve. This learning rate decay is not triggered for the first 15 epochs to ensure adequate training. We train for 30 epochs or until the learning rate has decayed 5 times, which was enough for convergence for all models. Model parameters are initialized over $\mathcal{U}(-0.1, 0.1)$ and gradients are clipped at 5. We employ a KL-cost annealing schedule whereby the multiplier on the KL-cost term is increased linearly from 0.1 to 1.0 each batch over 10 epochs. For models trained with iterative inference we perform SVI via stochastic gradient descent with momentum 0.5 and learning rate 1.0. Gradients are clipped after each step of SVI (also at 5).

B.3. Images

The preprocessed OMNIGLOT dataset does not have a standard validation split so we randomly pick 2000 images from training as validation. As with previous works the pixel value is scaled to be between 0 and 1 and interpreted as probabilities, and the images are dynamically binarized during training.

Our inference network consists of 3 residual blocks where each block is made up of a standard residual layer (i.e. two convolutional layers with 3×3 filters, ReLU nonlinearities, batch normalization, and residual connections) followed by a downsampling convolutional layer with filter size and stride equal to 2. These layers have 64 feature maps. The output of residual network is flattened and then used to obtain the variational means/log variances via an affine transformation.

The sample from the variational distribution (which is 32-dimensional) is first projected out to the image spatial resolution with 4 feature maps (i.e. $4 \times 28 \times 28$) via a linear transformation, then concatenated with the original image, and finally fed as input to a 12-layer Gated PixelCNN (van den Oord et al., 2016). The PixelCNN has three 9×9 layers, followed by three 7×7 layers, then three 5×5 layers, and finally three 3×3 layers. All the layers have 32 feature maps, and there is a final 1×1 convolutional layer followed by a sigmoid nonlinearity to produce a distribution over binary output. The layers are appropriately masked to ensure that the distribution over each pixel is conditioned only on the pixels left/top of it. We train with Adam with learning rate 0.001, β_1 = 0.9, β_2 = 0.999 for 100 epochs with batch size of 50. Gradients are clipped at 5.

For models trained with iterative inference we perform SVI via stochastic gradient descent with momentum 0.5 and learning rate 1.0, with gradient clipping (also at 5).

C. Data Size/Model Capacity

In Table 4 we investigate the performance of VAE/SA-VAE as we vary the capacity of the inference network, size of the training set, and the capacity of the generative model. The MLP inference network has two ReLU layers with 128 hidden units. For varying the PixelCNN generative model,

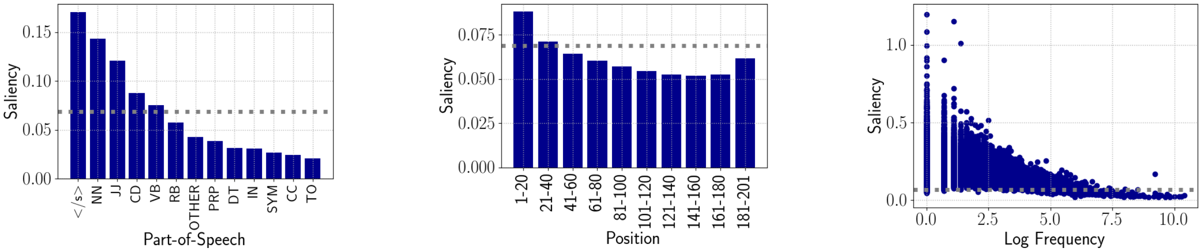

Figure 5. Input saliency by part-of-speech tag (left), position (center), and log frequency (right). The dotted gray line in each plot shows the average saliency across all words.

we sequentially remove layers from our baseline 12-layer model starting from the bottom (so the 9-layer PixelCNN has three 7×7 layers, three 5×5 layers, three 3×3 layers, all with 32 feature maps).

Intuitively, we expect iterative inference to help more when the inference network and the generative model are less powerful, and we indeed see this in Table 4. Further, one might expect SA-VAE to be more helpful in small-data regimes as it is harder for the inference network amortize inference and generalize well to unseen data. However we find that SA-VAE outperforms VAE by a similar margin across all training set sizes.

Finally, we observe that across all scenarios the KL portion of the loss is much higher for models trained with SA-VAE, indicating that these models are learning generative models that make more use of the latent representations.

D. Input Saliency Analysis

In Figure 5 we show the input saliency by part-of-speech tag (left), position (center), and frequency (right). Input saliency of a token $\mathbf{x}_t$ is defined as:

$$\left\| \mathbb{E}_{q(\mathbf{z};\lambda)}\left[\frac{\mathrm{d}\|\mathbf{z}\|_2}{\mathrm{d}\mathbf{w}_t}\right]\right\|_2$$

Here $\mathbf{w}_t$ is the encoder word embedding for $\mathbf{x}_t$. Part-of-speech tagging is done using NLTK.