EECS 498/598 Deep Learning - Homework 3

March 2nd, 2019

Instructions

- This homework is Due March 26th at 11.59pm. Late submission policies apply.
- You will submit a write-up and your code for this homework.

1 [25 points] Text Classification using CNNs

In this problem, you will implement a CNN text classifier similar to the network of [1] for sentiment classification. The network has the following architecture.

`Embedding layer` → `1-d Convolution` → `Pooling` → `ReLU` → `Linear` → `Sigmoid`

Assume that we perform global average-pooling in the pooling layer.

Assume the input to the convolution layer is given by $X \in \mathbb{R}^{N \times C \times H}$. Further assume that the temporal dimension (or sequence dimension) is the third dimension, of size H. Consider a convolutional kernel $W^{\text{conv}} \in \mathbb{R}^{F \times C \times H'}$ and a bias vector $b \in \mathbb{R}^F$. The output of 1-d convolution is given by $Y \in \mathbb{R}^{N \times F \times H''}$.

SOLUTION:

1. Based on the definition of a convolutional layer, we can express Y as $Y_{n,f} = X_n *_{\text{filt}} W_f^{\text{conv}} + b_f$ Alternatively, it could also be expressed as $Y_{n,f} = \sum_c X_{n,c} *_{\text{filt}} W_{f,c}^{\text{conv}} + b_f$ for an appropriate definition of $*_{\text{filt}}$ for 1-dimensional signals.
2. $H - H' + 1$ (Additional singleton dimension is also fine)
3. $N \times F$: After pooling along the temporal dimension, we have a single feature per filter

2 [10 points] Siamese Networks for Learning Embeddings.

In this problem, you will implement a Siamese network for face verification in `siamese_face.py`. The dataset is `att_faces.tar` and `lfw_faces.tar`.You can download the data from `https://drive.google.com/drive/folders/1Vpf8XctTtc-Swug7JE5YJU2URkvvxByV?usp=sharing`

1. Implement the contrasive loss class `ContrasiveLoss`

$$L(x^{(1)}, x^{(2)}, y) = (1-y)\|f(x^{(1)}) - f(x^{(2)})\|_2^2 + y(max\{0, m - \|f(x^{(1)}) - f(x^{(2)})\|_2\})^2$$

where m is the margin value, y is the label denoting whether $x^{(1)}$ and $x^{(2)}$ are from the same person. For more details, you may want to read `http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf`

2. Design architecture and build the embedding network in class `SiameseNetwork`, and train the siamese network for face images in att dataset. Report the learning curve of losses on training dataset and the qualitative result on training/testing dataset, i.e. the figures showing a pair of images and the dissimilarity measurement between these two pictures. You may follow the skeleton of architecture given in code comment to get reasonable performance. It's also great if you could adjust the optimizer, learning rate, number of epochs, network architecture, image pre-processing, batch size, margin value, etc. to get even better performance. We expect the reported dissimilarity number is consistent with visual similarity.

3. **Extra credit:** Repeat the process in the above question to train siamese network for face images in lfw dataset. The given skeleton of architecture and hyperparameters might not work for this large dataset, so you may need to adjust the optimizer, learning rate, number of epochs, network architecture, image pre-processing, batch size, margin value, etc. to get even better performance.

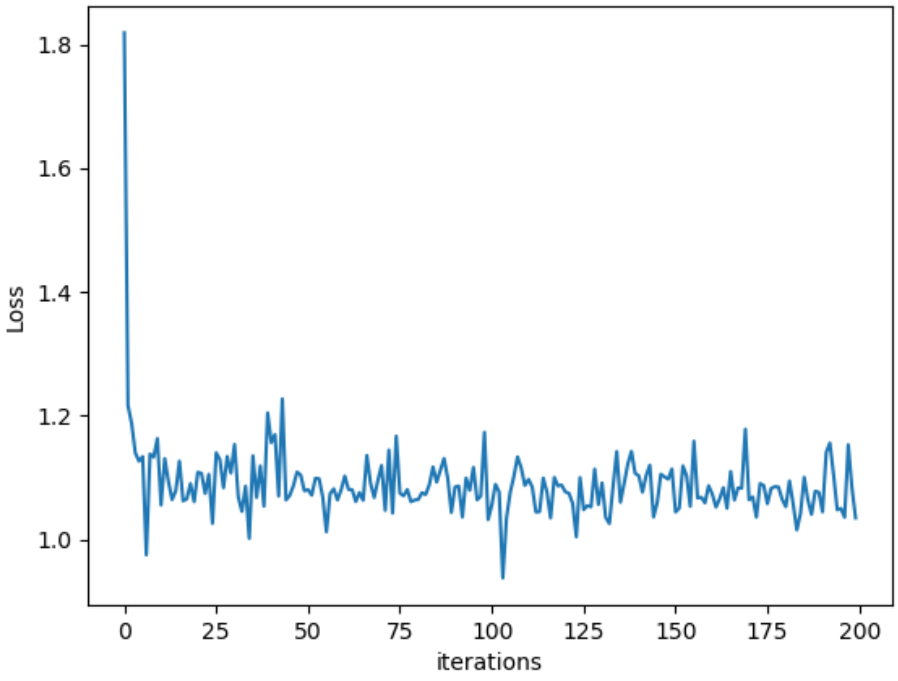

Figure 1: **SOLUTION:** Estimated loss curve. It could be better depending on your implementation.

Figure 2: **SOLUTION:** Training data dissimilarity.

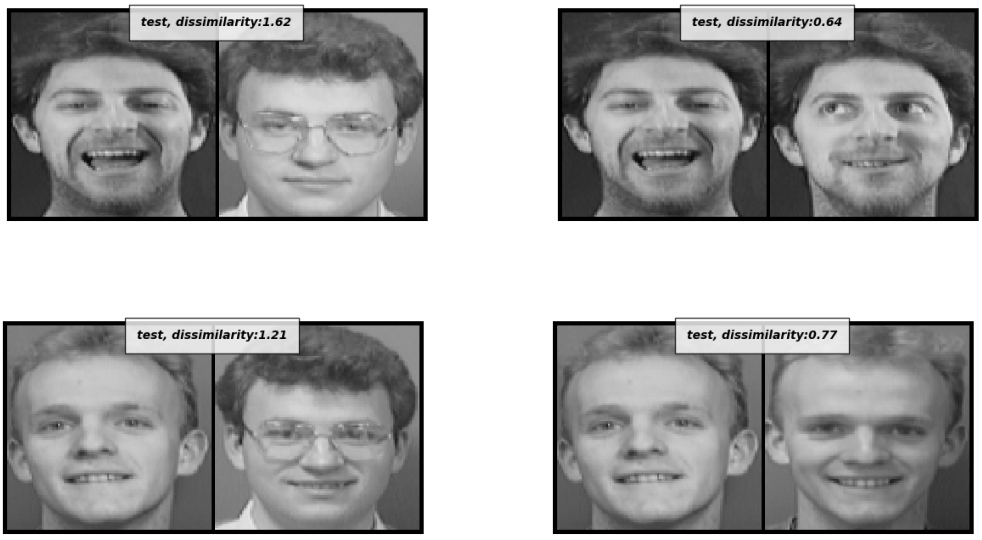

Figure 3: **SOLUTION:** Testing data dissimilarity.

3 [35 points] Conditional Variational Autoencoders.

In this problem, you will implement a conditional variational autoencoder (CVAE) from [2] and train it on the MNIST dataset.

1. Derive the variational lowerbound of a conditional variational autoencoder. Show that:

$$\begin{aligned}\log p_\theta(\mathbf{x}|\mathbf{y}) &\geq \mathcal{L}(\theta,\phi;\mathbf{x},\mathbf{y}) \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})\right] - D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\,\|p_\theta(\mathbf{z}|\mathbf{y})\right), \end{aligned} \tag{1}$$

where $\mathbf{x}$ is a binary vector of dimension d, $\mathbf{y}$ is a one-hot vector of dimension c defining a class, $\mathbf{z}$ is a vector of dimension m sampled from the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$. The posterior distribution is modeled by a neural network of parameters ϕ. The generative distribution $p_\theta(\mathbf{x}|\mathbf{y})$ is modeled by another neural network of parameters θ.

SOLUTION: From the second line on, we denote the $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x},\mathbf{y})}$ as $\mathbb{E}_{\mathbf{z}}$ for simplicity. Therefore,

$$\begin{aligned}\log p_\theta(\mathbf{x}|\mathbf{y}) &= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x},\mathbf{y})}\Big[\log p_\theta(\mathbf{x}|\mathbf{y})\Big] \\ &= \mathbb{E}_{\mathbf{z}}\left[\log \frac{p(\mathbf{x}|\mathbf{y},\mathbf{z})p(\mathbf{z}|\mathbf{y})}{p(\mathbf{z}|\mathbf{x},\mathbf{y})}\right] \\ &= \mathbb{E}_{\mathbf{z}}\left[\log \frac{p(\mathbf{x}|\mathbf{y},\mathbf{z})p(\mathbf{z}|\mathbf{y})}{p(\mathbf{z}|\mathbf{x},\mathbf{y})}\frac{q(\mathbf{z}|\mathbf{x},\mathbf{y})}{q(\mathbf{z}|\mathbf{x},\mathbf{y})}\right] \\ &= \mathbb{E}_{\mathbf{z}}\Big[\log p(\mathbf{x}|\mathbf{y},\mathbf{z})\Big] - \mathbb{E}_{\mathbf{z}}\left[\log \frac{q(\mathbf{z}|\mathbf{x},\mathbf{y})}{p(\mathbf{z}|\mathbf{y})}\right] + \mathbb{E}_{\mathbf{z}}\left[\log \frac{q(\mathbf{z}|\mathbf{x},\mathbf{y})}{p(\mathbf{z}|\mathbf{x},\mathbf{y})}\right] \\ &= \mathbb{E}_{\mathbf{z}}\Big[\log p(\mathbf{x}|\mathbf{y},\mathbf{z})\Big] - D_{KL}\left(q(\mathbf{z}|\mathbf{x},\mathbf{y})||p(\mathbf{z}|\mathbf{y})\right) + \underbrace{D_{KL}\left(q(\mathbf{z}|\mathbf{x},\mathbf{y})||p(\mathbf{z}|\mathbf{x},\mathbf{y})\right)}_{\geq 0} \\ &\geq \mathbb{E}_{\mathbf{z}}\Big[\log p(\mathbf{x}|\mathbf{y},\mathbf{z})\Big] - D_{KL}\left(q(\mathbf{z}|\mathbf{x},\mathbf{y})||p(\mathbf{z}|\mathbf{y})\right) \\ &= \mathcal{L}(\theta,\phi;\mathbf{x},\mathbf{y})\end{aligned}$$

2. Derive the analytical solution to the KL-divergence between two Gaussian distributions $D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\,\|p_\theta(\mathbf{z}|\mathbf{y})\right)$. Let us assume that $p_\theta(\mathbf{z}|\mathbf{y}) \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ and show that:

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\,\|p_\theta(\mathbf{z}|\mathbf{y})\right) = -\frac{1}{2}\sum_{j=1}^{J}\left(1+\log\left(\sigma_j^2\right)-\mu_j^2-\sigma_j^2\right), \tag{2}$$

where μ_j and σ_j are the outputs of the neural network that estimates the parameters of the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$.

SOLUTION: First, remember that all z_j are assumed to be independent (i.e. $p_\theta(\mathbf{z}|\mathbf{y}) = \prod_j p_\theta(z_j|\mathbf{y})$. Therefore, we have

$$\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})) &= \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}{p_\theta(\mathbf{z}|\mathbf{y})} d\mathbf{z} \\
&= \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \log q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\, d\mathbf{z} - \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \log p_\theta(\mathbf{z}|\mathbf{y})\, d\mathbf{z} \\
&= \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \log \prod_j q_\phi(z_j|\mathbf{x},\mathbf{y})\, d\mathbf{z} - \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \log \prod_j p_\theta(z_j|\mathbf{y})\, d\mathbf{z} \\
&= \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \sum_j \log q_\phi(z_j|\mathbf{x},\mathbf{y})\, d\mathbf{z} - \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \sum_j \log p_\theta(z_j|\mathbf{y})\, d\mathbf{z} \\
&= \sum_j \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \log q_\phi(z_j|\mathbf{x},\mathbf{y})\, d\mathbf{z} - \sum_j \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \log p_\theta(z_j|\mathbf{y})\, d\mathbf{z}
\end{aligned}$$

Now, note that for each j in the sum, we have

$$\begin{aligned}
\int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \log q_\phi(z_j|\mathbf{x},\mathbf{y})\, d\mathbf{z} &= \int_{\mathbf{z}} \prod_{j'} q_\phi(z_{j'}|\mathbf{x},\mathbf{y}) \log q_\phi(z_j|\mathbf{x},\mathbf{y})\, d\mathbf{z} \\
&= \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \log q_\phi(z_j|\mathbf{x},\mathbf{y})\, dz_j \underbrace{\prod_{j' \neq j} \int_{z_{j'}} q_\phi(z_{j'}|\mathbf{x},\mathbf{y})\, dz_{j'}}_{=1} \\
&= \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \log q_\phi(z_j|\mathbf{x},\mathbf{y})\, dz_j
\end{aligned}$$

Therefore, we have

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})) = \sum_j \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \log q_\phi(z_j|\mathbf{x},\mathbf{y})\, dz_j - \sum_j \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \log p_\theta(z_j|\mathbf{y})\, dz_j$$

Now, let us plug in Gaussian distributions. Let's start with the left side:

$$\begin{aligned}
\sum_j \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \log q_\phi(z_j|\mathbf{x},\mathbf{y})\, dz_j &= \sum_j \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \log \left(\frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(z_j-\mu_j)^2}{2\sigma_j^2}} \right) dz_j \\
&= \sum_j \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \left(\log \frac{1}{\sqrt{2\pi\sigma_j^2}} - \frac{(z_j-\mu_j)^2}{2\sigma_j^2} \right) dz_j \\
&= \sum_j -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma_j^2 - \frac{1}{2\sigma_j^2} \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \left[(z_j-\mu_j)^2 \right] dz_j \\
&= \sum_j -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma_j^2 - \frac{1}{2\sigma_j^2}\sigma_j^2 \\
&= \sum_j -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma_j^2 - \frac{1}{2}
\end{aligned}$$

Now, let us solve the right side. Remember that we assume that $p_\theta(\mathbf{z}|\mathbf{y}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Therefore:

$$\begin{aligned}
\sum_j \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \log p_\theta(z_j|\mathbf{y})\, dz_j &= \sum_j \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \log\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{z_j^2}{2}}\right) dz_j \\
&= \sum_j \int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y}) \left(\log\frac{1}{\sqrt{2\pi}} - \frac{z_j^2}{2}\right) dz_j \\
&= \sum_j -\frac{1}{2}\log 2\pi - \frac{1}{2}\int_{z_j} q_\phi(z_j|\mathbf{x},\mathbf{y})\, z_j^2 dz_j \\
&= \sum_j -\frac{1}{2}\log 2\pi - \frac{1}{2}\Big(\sigma_j^2 + \mu_j^2\Big)
\end{aligned}$$

Therefore, combining the solutions of the left and right side we have:

$$\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})) &= \Big(\sum_j -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma_j^2 - \frac{1}{2}\Big) - \Big(\sum_j -\frac{1}{2}\log 2\pi - \frac{1}{2}\Big(\sigma_j^2 + \mu_j^2\Big)\Big) \\
&= \sum_j -\frac{1}{2}\log\sigma_j^2 - \frac{1}{2} + \frac{1}{2}\Big(\sigma_j^2 + \mu_j^2\Big) \\
&= -\frac{1}{2}\sum_j \Big(1 + \log\sigma_j^2 - \sigma_j^2 - \mu_j^2\Big)
\end{aligned}$$

3. Fill in code for CVAE network as a `nn.Module` class called `CVAE` in the file `cvae.py`
   - Implement the `recognition_model` function $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$.
   - Implement the `generative_model` function $p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})$.
   - Implement the `forward` function by inferring the Gaussian parameters using the recognition model, sampling a latent variable using the reparametrization trick and generating the data using the generative model.
   - Implement the variational lowerbound `loss_function` $\mathcal{L}(\theta,\phi;\mathbf{x},\mathbf{y})$.
   - Train the CVAE and visualize.

If trained successfully, you should be able to sample images $\mathbf{x}$ that reflect the given label $\mathbf{y}$ given the noise vector $\mathbf{z}$.

Figure 4: **SOLUTION:** Digits should be identifiable. Start from 0 and end at 9.

4 [30 points] Generative Adversarial Networks.

In this problem, you will implement generative adversarial networks and train it on the MNIST dataset. Specifically, you will implement the Deep Convolutional Generative Adversarial Networks (DCGAN) from [3]. In the generative adversarial networks formulation, we have a generator network G that takes in random vector $\mathbf{z}$ and a discriminator network D that takes in an input image $\mathbf{x}$. The parameters of G and D are optimized via the adversarial objective:

$$\min_G \max_D \quad \mathbb{E}_{\mathbf{x}\sim p_{data}}\Big[\log D(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log(1 - D(G(\mathbf{z})))\Big], \tag{3}$$

In practice, we alternate between training D and G where we train G to maximize:

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log D(G(\mathbf{z}))\Big], \tag{4}$$

and we follow by training D to maximize:

$$\mathbb{E}_{\mathbf{x}\sim p_{data}}\Big[\log D(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log(1 - D(G(\mathbf{z})))\Big], \tag{5}$$

Therefore, the two separate optimizations make up one full training step. Given this information, you will do the following:

1. Fill in code for the DCGAN network in the `gan.py`. Descriptions of what should be filled in is written as comments in the code itself.
    - Implement the `sample_noise` function.
    - Implement the `build_discriminator` function.
    - Implement the `build_generator` function.
    - Implement the `get_optimizer` function.
    - Implement the `bce_loss` function.
    - Use the previously implemented `bce_loss` to implement the `discriminator_loss` function.
    - Use the previously implemented `bce_loss` implement the `generator_loss` function.
    - Train your DCGAN!

If trained successfully, you should see the progression of sample quality getting better as training epochs increase.

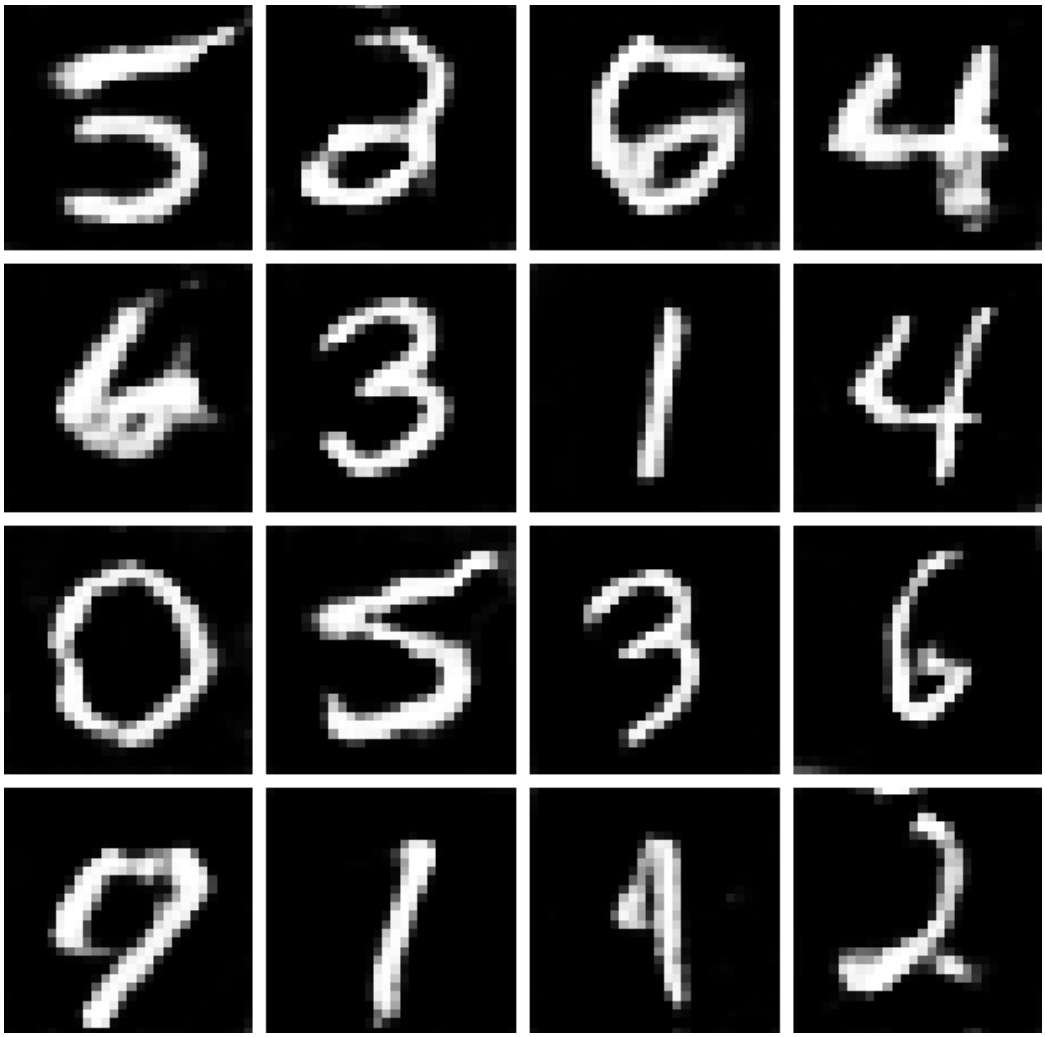

Figure 5: **SOLUTION:** Please note that you may get different digits during the sampling process, however, the digit quality should be similar.

References

[1] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[2] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *NeurIPS*. 2015.

[3] Alec Radford, Luke Me, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *ICLR*. 2016.