HOMEWORK 6

Saikumar Yadugiri
9083079468, saikumar@cs.wisc.edu

Instructions: Use this latex file as a template to develop your homework. Submit your homework on time as a single pdf file. Please wrap your code and upload to a public GitHub repo, then attach the link below the instructions so that we can access it. Answers to the questions that are not within the pdf are not accepted. This includes external links or answers attached to the code implementation. Late submissions may not be accepted. You can choose any programming language (i.e. python, R, or MATLAB). Please check Piazza for updates about the homework. It is ok to share the results of the experiments and compare them with each other.

GitHub Link: `https://github.com/saikumarysk/cs760_hw6`

1 Implementation: GAN (50 pts)

In this part, you are expected to implement GAN with MNIST dataset. We have provided a base jupyter notebook (gan-base.ipynb) for you to start with, which provides a model setup and training configurations to train GAN with MNIST dataset.

(a) Implement training loop and report learning curves and generated images in epoch 1, 50, 100. Note that drawing learning curves and visualization of images are already implemented in provided jupyter notebook. (20 pts)

Procedure 1 Training GAN, modified from Goodfellow et al. (2014)

Input: m: real data batch size, n_z: fake data batch size
Output: Discriminator D, Generator G
for number of training iterations **do**
# Training discriminator
Sample minibatch of n_z noise samples $\{z^{(1)}, z^{(2)}, \cdots, z^{(n_z)}\}$ from noise prior $p_g(z)$
Sample minibatch of $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$
Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d}\Big(\frac{1}{m}\sum_{i=1}^{m}\log D(x^{(i)}) + \frac{1}{n_z}\sum_{i=1}^{n_z}\log(1 - D(G(z^{(i)})))\Big)$$

Training generator
Sample minibatch of n_z noise samples $\{z^{(1)}, z^{(2)}, \cdots, z^{(n_z)}\}$ from noise prior $p_g(z)$
Update the generator by ascending its stochastic gradient:

$$\nabla_{\theta_g}\frac{1}{n_z}\sum_{i=1}^{n_z}\log D(G(z^{(i)}))\Big)$$

end for
# The gradient-based updates can use any standard gradient-based learning rule. In the base code, we are using Adam optimizer (Kingma and Ba, 2014)

Expected results are as follows.

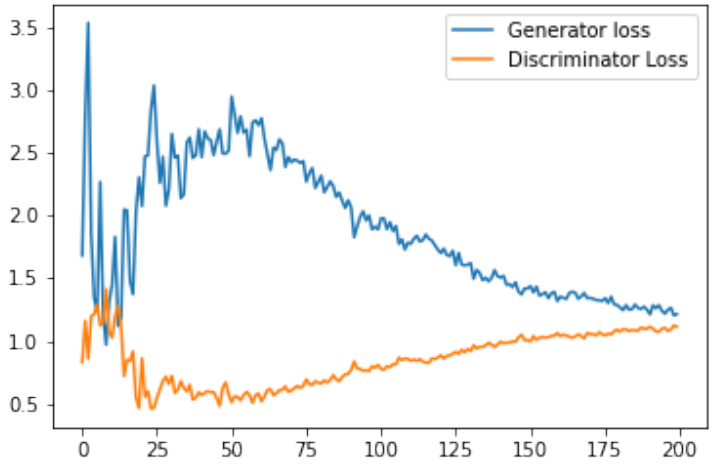
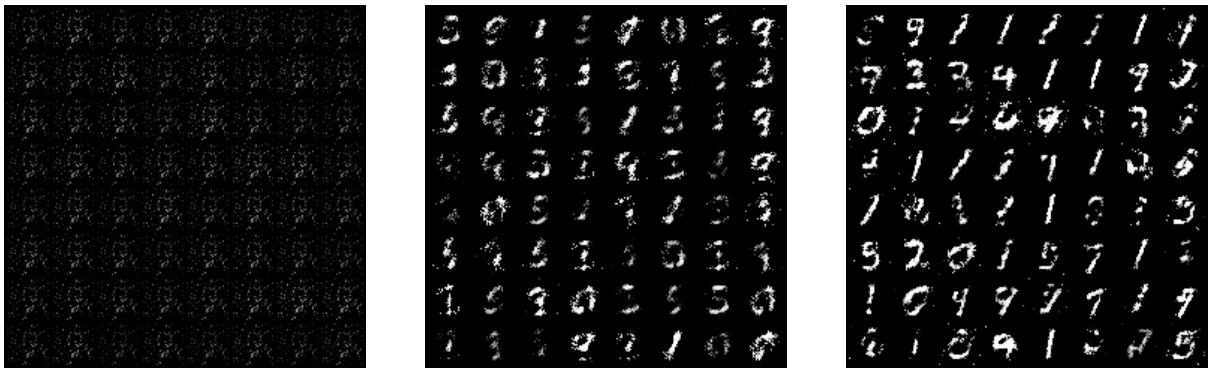


Figure 1: Learning curve



(a) epoch 1 (b) epoch 50 (c) epoch 100

Figure 2: Generated images by G

My implementation results are shown in figures 3 and 4. Specifically, the loss function is shown in 3 and the epochs are shown in 4.

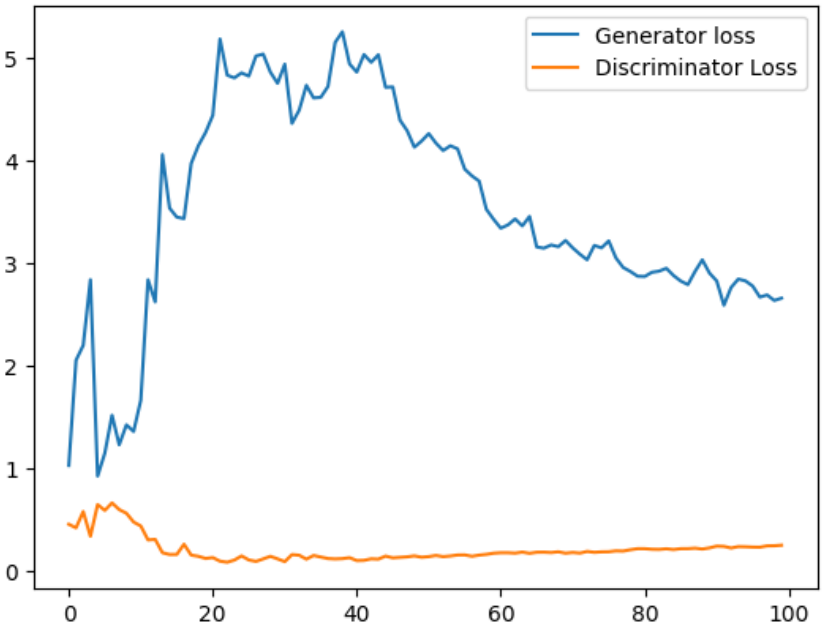
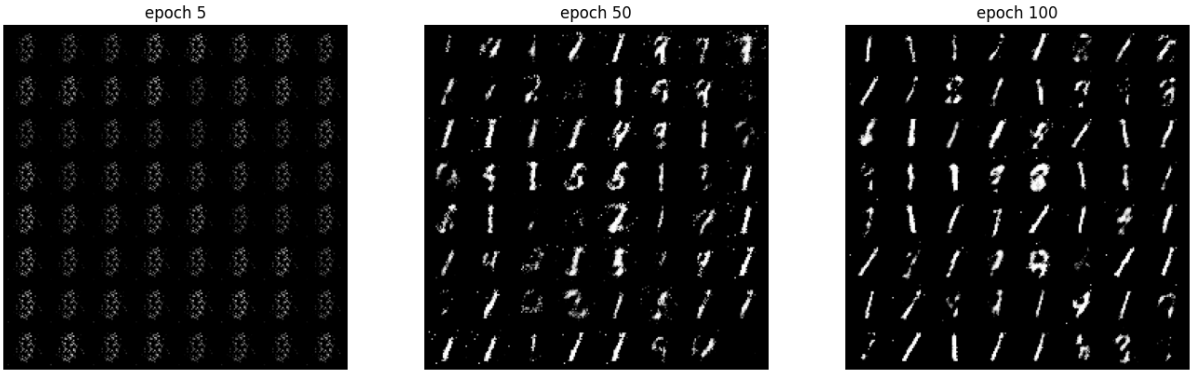


Figure 3: Learning curve



(a) epoch 1 (b) epoch 50 (c) epoch 100

Figure 4: Generated images by G

(b) Replace the generator update rule as the original one in the slide,
"Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{n_z} \sum_{i=1}^{n_z} \log(1 - D(G(z^{(i)})))\Big)$$

" , and report learning curves and generated images in epoch 1, 50, 100. Compare the result with (a). Note that it may not work. If training does not work, explain why it doesn't work.
You may find this helpful: https://jonathan-hui.medium.com/gan-what-is-wrong-with-the-gan-cost-function-6f594162ce01 (10 pts)

My implementation results are shown in figures 5 and 6. Specifically, the loss function is shown in 5 and the epochs are shown in 6.

The impression I have about why this doesn't work is that this rule has a "vanshing gradient" issue. As $D(G(z))$ is near 0, SGD cannot step too much limiting the capability of learning of the Generator.

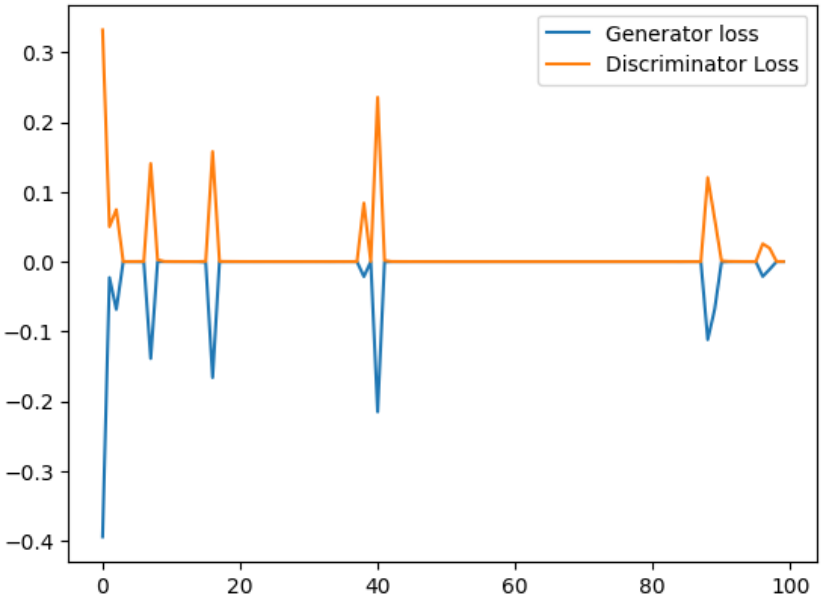
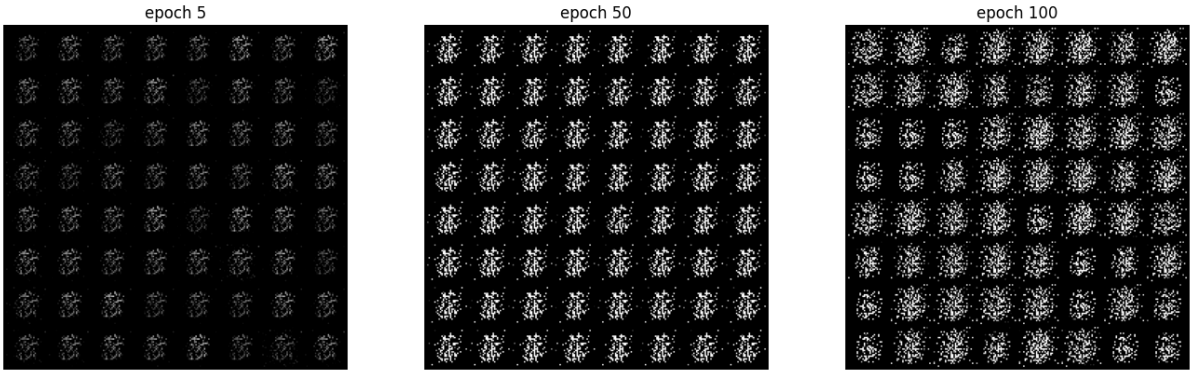


Figure 5: Learning curve



(a) epoch 1 (b) epoch 50 (c) epoch 100

Figure 6: Generated images by G

(c) Except the method that we used in (a), how can we improve training for GAN? Implement that and report your setup, learning curves, and generated images in epoch 1, 50, 100. This question is an open-ended question and you can choose whichever method you want. (20 pts)

I changed the batch size to 128 to make the curve a lot smoother. I also used 0.9 for the discriminator's real label instead of 1 so that the network will not suffer from overconfidence. I tried a different torch seed as well. The implementation results are shown in figures 7 and 8. Specifically, the loss function is shown in 7 and the epochs are shown in 8. Visual improvement can be seen in the process.

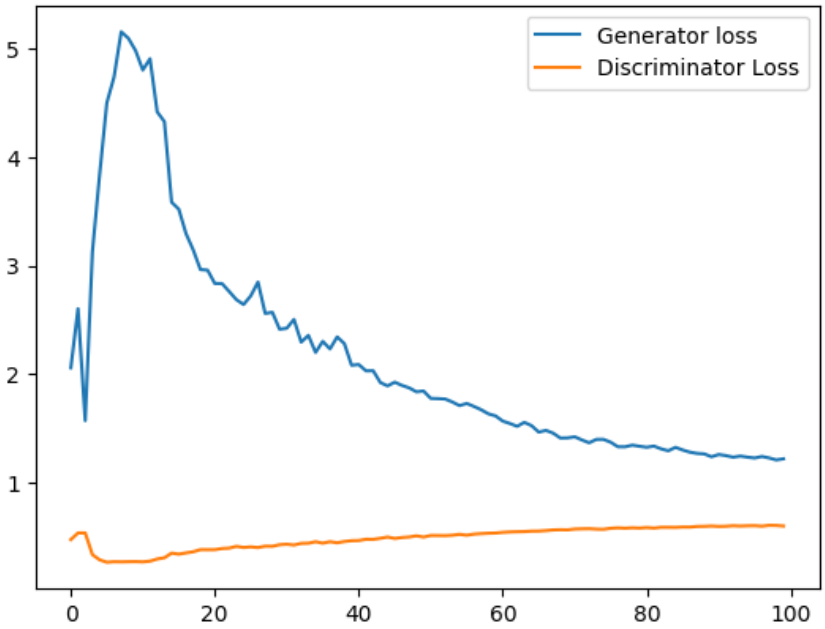


Figure 7: Learning curve

(a) epoch 1 (b) epoch 50 (c) epoch 100

Figure 8: Generated images by G

2 Directed Graphical Model [25 points]

Consider the directed graphical model (aka Bayesian network) in Figure 9.

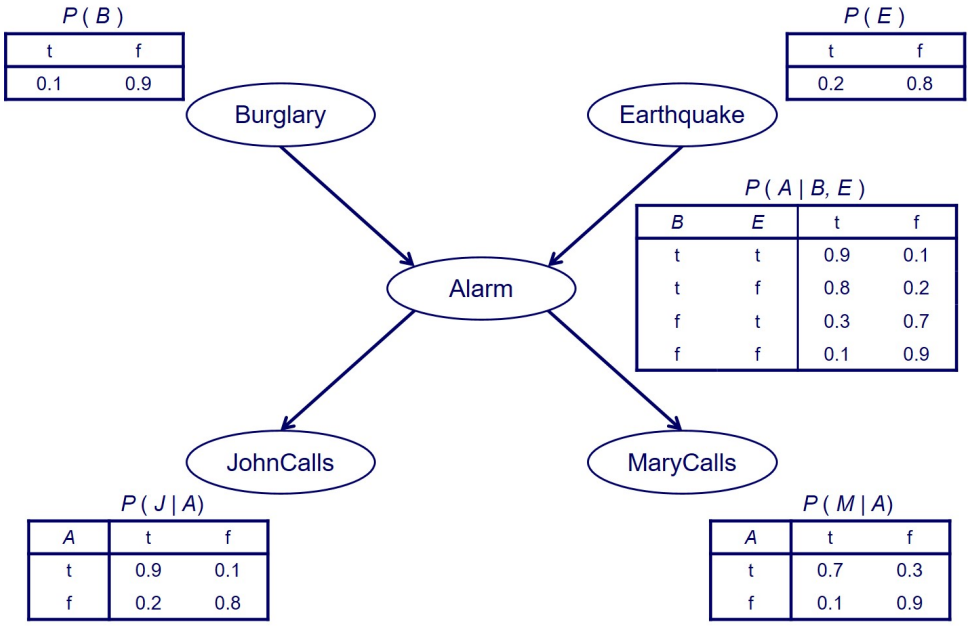


Figure 9: A Bayesian Network example.

Compute $P(B = t \mid E = f, J = t, M = t)$ and $P(B = t \mid E = t, J = t, M = t)$. (10 points for each) These are the conditional probabilities of a burglar in your house (yikes!) when both of your neighbors John and Mary call you and say they hear an alarm in your house, but without or with an earthquake also going on in that area (what a busy day), respectively.

Using Bayes' theorem,

$$\Pr[B = t|E = f, J = t, M = t] = \frac{\Pr[E = f, J = t, M = t|B = t] \Pr[B = t]}{\Pr[E = f, J = t, M = t]}$$

From the law of total probability,

$\Pr[E = f, J = t, M = t] = \Pr[E = f, J = t, M = t|B = t] \Pr[B = t] + \Pr[E = f, J = t, M = t|B = f] \Pr[B = f]$.

$\Pr[E = f, J = t, M = t|B = t] = \Pr[J = t, M = t|E = f, B = t] \Pr[E = f]$.

$= \sum_{a \in \{t,f\}} \Pr[J = t, M = t|A = a] \Pr[A = a|E = f, B = t] \Pr[E = f] = ((0.9 \times 0.7) \times 0.8 + (0.2 \times 0.1) \times 0.2) \times 0.8 = 0.4064$.

Similarly, $\Pr[E = f, J = t, M = t|B = t] = ((0.9 \times 0.7) \times 0.1 + (0.2 \times 0.1) \times 0.9) \times 0.8 = 0.0648$.

Hence, $\Pr[E = f, J = t, M = t] = 0.4064 \times 0.1 + 0.0648 \times 0.9 = 0.09896$.

$\therefore \Pr[B = t|E = f, J = t, M = t] = \frac{0.4064}{0.09896} = \mathbf{0.4106}$.

Following the same procedure above,

$$\Pr[B = t|E = t, J = t, M = t] = \frac{\Pr[E = t, J = t, M = t|B = t] \Pr[B = t]}{\Pr[E = t, J = t, M = t]}$$

And, $\Pr[E = t, J = t, M = t] = \sum_{a \in \{t,f\}} \sum_{b \in \{t,f\}} \Pr[J = t, M = t|A = a] \Pr[A = a|B = b, E = t] \Pr[E = t] \Pr[B = b] = 0.01138 + 0.03654 = 0.0479$.

$\therefore \Pr[B = t|E = t, J = t, M = t] = \frac{0.01138}{0.0479} = \mathbf{0.237}$.

3 Chow-Liu Algorithm [25 pts]

Suppose we wish to construct a directed graphical model for 3 features X, Y, and Z using the Chow-Liu algorithm. We are given data from 100 independent experiments where each feature is binary and takes value T or F. Below is a table summarizing the observations of the experiment:

X	Y	Z	Count
T	T	T	36
T	T	F	4
T	F	T	2
T	F	F	8
F	T	T	9
F	T	F	1
F	F	T	8
F	F	F	32

1. Compute the mutual information $I(X, Y)$ based on the frequencies observed in the data. (5 pts)
   $I(X,Y) = H_D(X) - H_D(X|Y)$.
   $H_D(X) = -\Pr[X = t]\log_2(\Pr[X = t]) - \Pr[X = f]\log_2(\Pr[X = f]) = -\frac{50}{100}\log_2\left(\frac{50}{100}\right) - \frac{50}{100}\log_2\left(\frac{50}{100}\right) = 1.$
   $H_D(X|Y = t) = -\frac{40}{50}\log_2\left(\frac{40}{50}\right) - \frac{10}{50}\log_2\left(\frac{10}{50}\right) = 0.722$
   $H_D(X|Y = f) = -\frac{10}{50}\log_2\left(\frac{10}{50}\right) - \frac{40}{50}\log_2\left(\frac{40}{50}\right) = 0.722$

   $H_D(X|Y) = 0.5(0.722) + 0.5(0.722) = 0.722$. Hence, $I(X, Y) = 1 - 0.722 = \mathbf{0.278}$

2. Compute the mutual information $I(X, Z)$ based on the frequencies observed in the data. (5 pts)
   Similar to above, we have that $I(X, Z) = \mathbf{0.125}$.

3. Compute the mutual information $I(Z, Y)$ based on the frequencies observed in the data. (5 pts)
   Similar to part 1, we can calculate that $I(Z, Y) = \mathbf{0.3916}$.

4. Which undirected edges will be selected by the Chow-Liu algorithm as the maximum spanning tree? (5 pts)
   Using Kruskal's algorithm for maximum spanning tree, we select the two nodes with maximum edge weight. Here, the edge weight used is the mutual information between nodes. As the mutual information between the Y, Z and X, Y is the biggest two, we will select these edges.

5. Root your tree at node X, assign directions to the selected edges. (5 pts)
   The tree is a simple path tree as shown in figure 10.

Figure 10: Directed Tree from Chow-Liu's Algorithm

References

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.