HOMEWORK 6

Nick Boddy
nboddy
GitHub repo: github.com/Nick-Boddy/CS760-HW6-GAN-BayesNet

2023-11-17

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.

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.

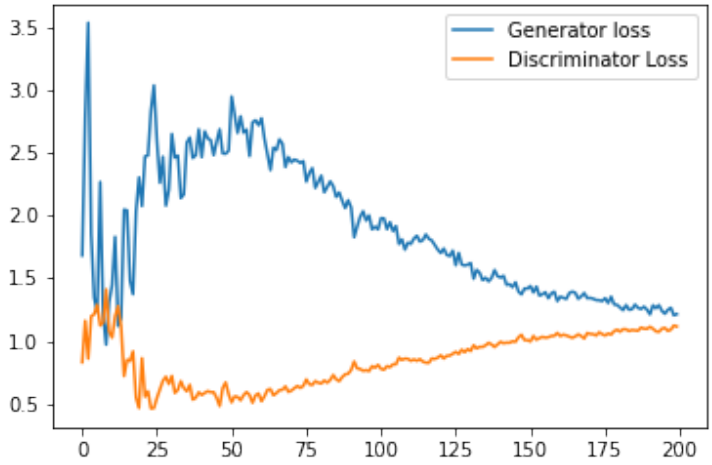
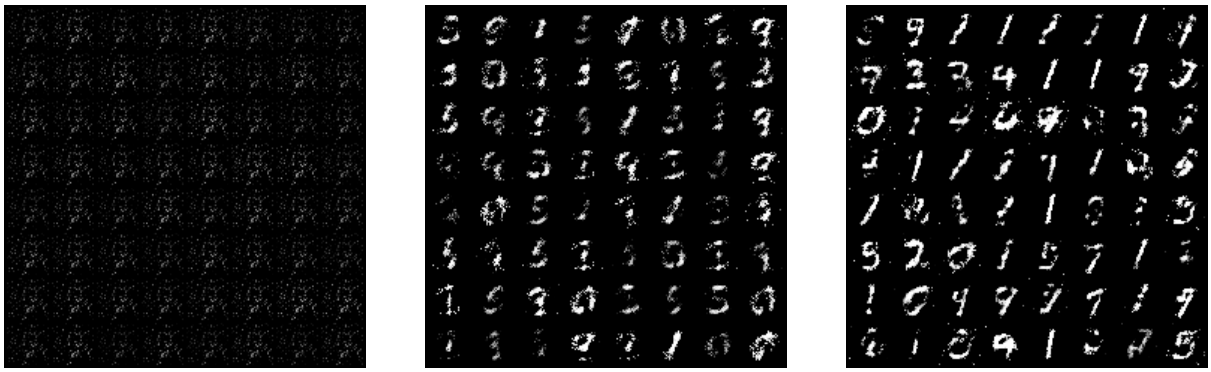


Figure 1: Learning curve



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

Figure 2: Generated images by G

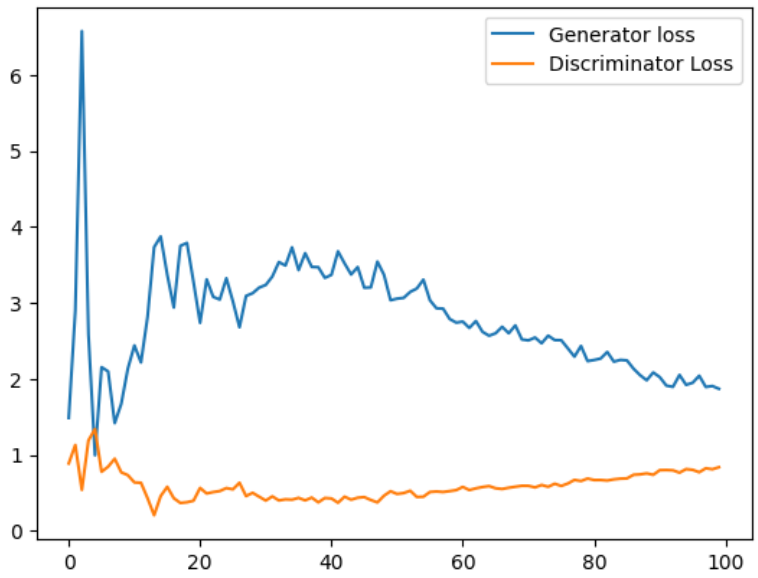


Figure 3: My learning curve

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

Figure 4: My generated images from 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)

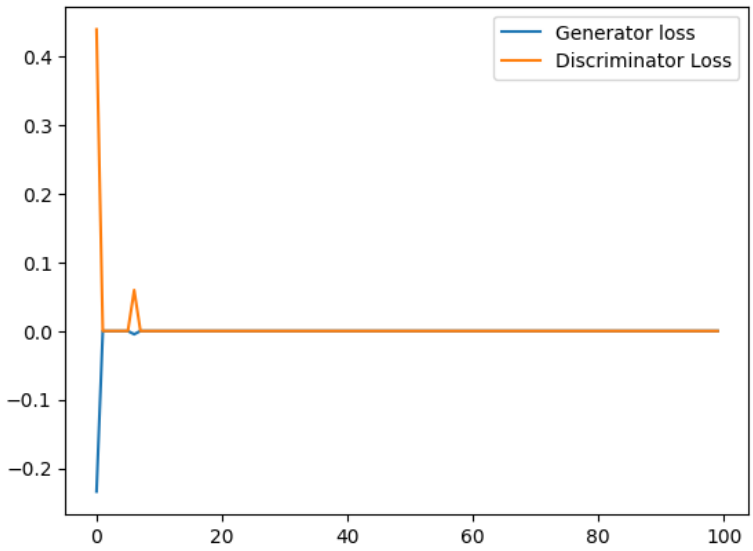
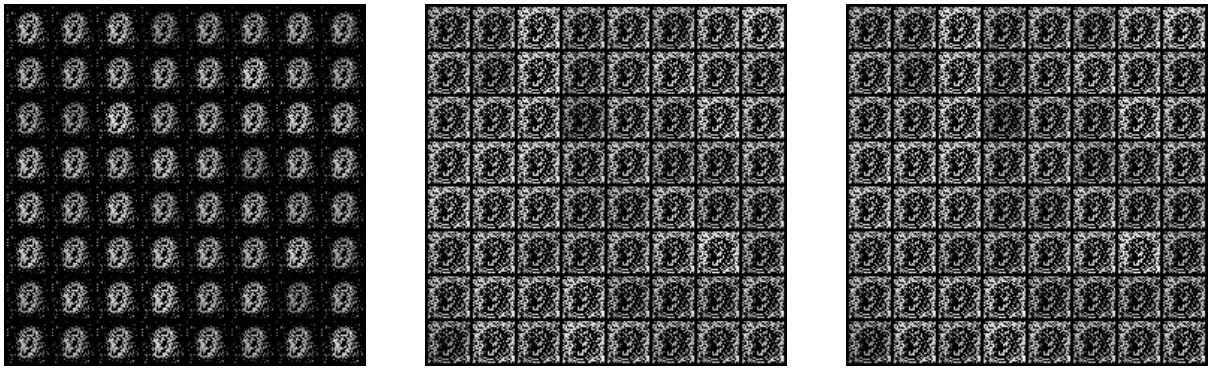


Figure 5: My learning curve



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

Figure 6: My generated images from G using an altered update rule

The results of using this altered rule for updating the generator clearly does not result in realistic images. This makes sense, since the new update rule of descending $\nabla_{\theta_g} \frac{1}{n_z} \sum_{i=1}^{n_z} \log(1 - D(G(z^{(i)})))\big)$ diminishes when the discriminator improves. Even though conceptually the θ_g that minimizes $\frac{1}{n_z} \sum_{i=1}^{n_z} \log(1 - D(G(z^{(i)})))\big)$ is the same θ_g that maximizes $\frac{1}{n_z} \sum_{i=1}^{n_z} \log D(G(z^{(i)}))\big)$, when considering the backpropagating gradients, the former values diminish to the point where the gradient vanishes, whereas the latter (which is used in part a), does not vanish (but could possibly be unstable).

(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've tried a couple modifications, some more successful than others. I tried altering the labels to be more smooth in training, with a 0.9 for real images instead of 1, and 0.1 for fake images instead of 0.
Another modification I tried was adjusting the optimizers to have a decaying learning rate. However, after applying with many different decay rates and other hyperparameters, I couldn't see any improvement. In fact, most of the time I found that the discriminator reached a (relatively) optimal state really early, which prevented the generator from improving at all.
A third modification that actually saw improvements was adding some noise to the real images when training the discriminator. Conceptually, I would suppose that this helps to stabilize the gradient. So the added noise

remains in the final training method.
Lastly, I changed some of the parameters to the models and batching, and trained up to 200 epochs. This did improve performance, even at 100 epochs. Figures and final parameters are given below.

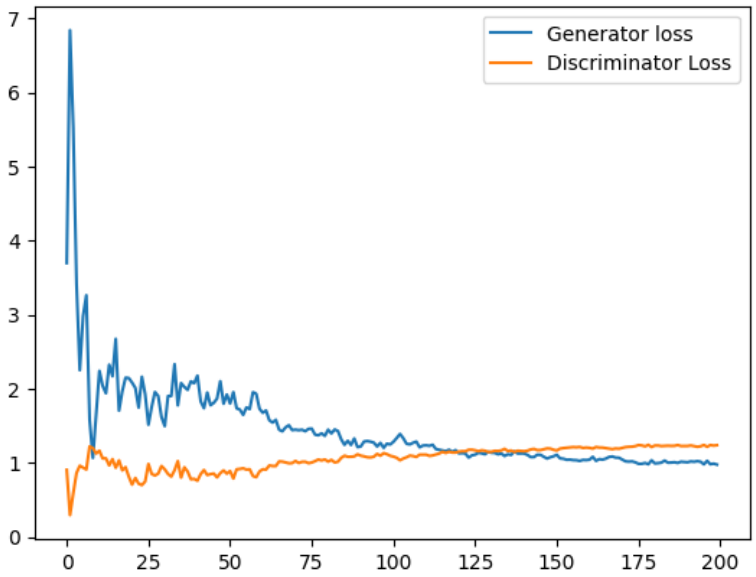
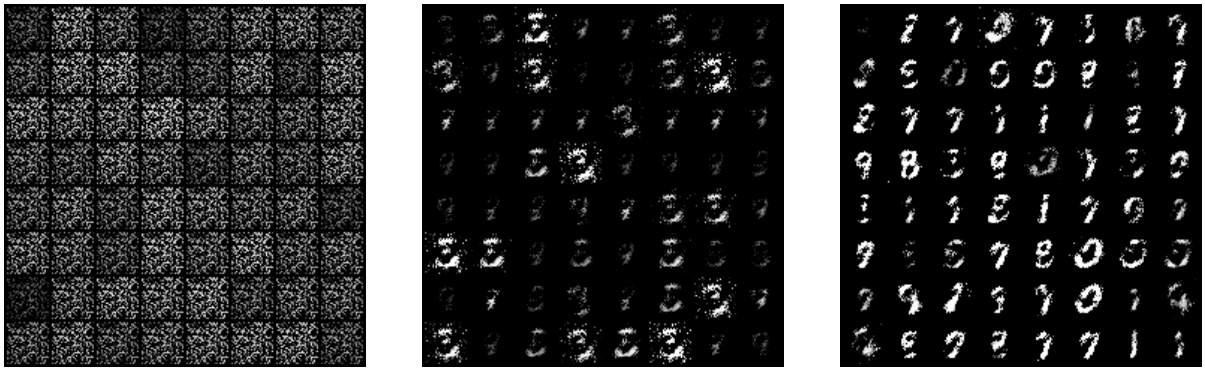


Figure 7: My learning curve



(a) epoch 1 (b) epoch 25 (c) epoch 50

(a) epoch 100 (b) epoch 150 (c) epoch 200

Figure 9: Generated G images after improvements

PARAMETERS

```
# learning parameters
batch_size = 256
epochs = 200
sample_size = 64 # fixed sample size for generator
nz = 128 # latent vector size
k = 1 # number of steps to apply to the discriminator


optimizer type: Adam
optimizer learning rate: 0.0002
loss function: Binary Cross-Entropy
added (clamped) noise to real images that follows a standard Gaussian * 0.1


##### GENERATOR #####
Generator(
(main): Sequential(
(0): Linear(in_features=128, out_features=256, bias=True)
(1): LeakyReLU(negative_slope=0.2)
(2): Linear(in_features=256, out_features=512, bias=True)
(3): LeakyReLU(negative_slope=0.2)
(4): Linear(in_features=512, out_features=1024, bias=True)
(5): LeakyReLU(negative_slope=0.2)
(6): Linear(in_features=1024, out_features=784, bias=True)
(7): Tanh()
)
)
######################

##### DISCRIMINATOR #####
Discriminator(
(main): Sequential(
(0): Linear(in_features=784, out_features=1024, bias=True)
(1): LeakyReLU(negative_slope=0.3)
(2): Dropout(p=0.4, inplace=False)
(3): Linear(in_features=1024, out_features=512, bias=True)
(4): LeakyReLU(negative_slope=0.3)
(5): Dropout(p=0.4, inplace=False)
(6): Linear(in_features=512, out_features=256, bias=True)
(7): LeakyReLU(negative_slope=0.3)
(8): Dropout(p=0.4, inplace=False)
(9): Linear(in_features=256, out_features=1, bias=True)
(10): Sigmoid()
)
)
######################
```

2 Directed Graphical Model [25 points]

Consider the directed graphical model (aka Bayesian network) in Figure 10.

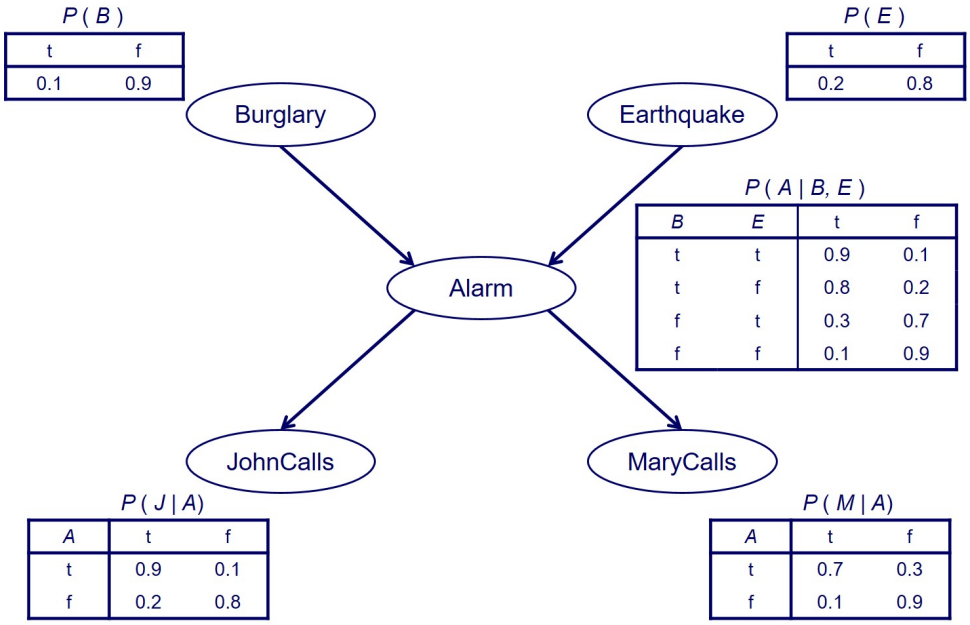


Figure 10: A Bayesian Network example.

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.

$$P(B = t \mid E = f, J = t, M = t)$$

$$= P(B \mid \neg E, J, M) = \frac{P(B, \neg E, J, M)}{P(\neg E, J, M)}$$

$$= \frac{P(B, \neg E, J, M)}{P(B, \neg E, J, M) + P(\neg B, \neg E, J, M)}$$

$$P(B, \neg E, J, M) = \sum_{a \in \{A, \neg A\}} P(B)P(\neg E)P(a|B, \neg E)P(J|a)P(M|a)$$

$$= (0.1)(0.8)(0.8)(0.9)(0.7) + (0.1)(0.8)(0.2)(0.2)(0.1) = 0.04064$$

$$P(\neg B, \neg E, J, M) = \sum_{a \in \{A, \neg A\}} P(\neg B)P(\neg E)P(a|\neg B \neg E)P(J|a)P(M|a)$$

$$= (0.9)(0.8)(0.1)(0.9)(0.7) + (0.9)(0.8)(0.9)(0.2)(0.1) = 0.05832$$

$$P(B \mid \neg E, J, M) = \frac{0.04064}{0.04064 + 0.05832} = 0.410670978$$

$$P(B = t \mid E = t, J = t, M = t)$$

$$= P(B \mid E, J, M) = \frac{P(B, E, J, M)}{P(E, J, M)}$$

$$= \frac{P(B, E, J, M)}{P(B, E, J, M) + P(\neg B, E, J, M)}$$

$$P(B, E, J, M) = \sum_{a \in \{A, \neg A\}} P(B)P(E)P(a|B, E)P(J|a)P(M|a)$$

$$= (0.1)(0.2)(0.9)(0.9)(0.7) + (0.1)(0.2)(0.1)(0.2)(0.1) = 0.01138$$

$$P(\neg B, E, J, M) = \sum_{a \in \{A, \neg A\}} P(\neg B)P(E)P(a|\neg B, E)P(J|a)P(M|a)$$
$$= (0.9)(0.2)(0.3)(0.9)(0.7) + (0.9)(0.2)(0.7)(0.2)(0.1) = 0.03654$$
$$P(B \mid E, J, M) = \frac{0.01138}{0.01138 + 0.03654} = 0.237479132$$

$$P(B \mid \neg E, J, M) \approx 0.411$$
$$P(B \mid E, J, M) \approx 0.237$$

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)

$$I(X, Y) = \sum_{x \in \{X, \neg X\}} \sum_{y \in \{Y, \neg Y\}} P(x, y) \log_2 \frac{P(x, y)}{P(x)P(y)}$$
$$= 0.4 \log_2 \frac{0.4}{0.5 \times 0.5} + 0.1 \log_2 \frac{0.1}{0.5 \times 0.5} + 0.1 \log_2 \frac{0.1}{0.5 \times 0.5} + 0.4 \log_2 \frac{0.4}{0.5 \times 0.5}$$
$$\approx 0.2781$$

2. Compute the mutual information $I(X, Z)$ based on the frequencies observed in the data. (5 pts)

$$I(X, Z) = \sum_{x \in \{X, \neg X\}} \sum_{z \in \{Z, \neg Z\}} P(x, z) \log_2 \frac{P(x, z)}{P(x)P(z)}$$
$$= 0.38 \log_2 \frac{0.38}{0.5 \times 0.55} + 0.12 \log_2 \frac{0.12}{0.5 \times 0.45} + 0.17 \log_2 \frac{0.17}{0.5 \times 0.55} + 0.33 \log_2 \frac{0.33}{0.5 \times 0.45}$$
$$\approx 0.1328$$

3. Compute the mutual information $I(Z, Y)$ based on the frequencies observed in the data. (5 pts)

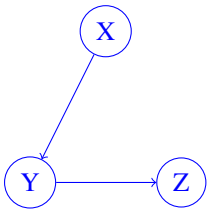
$$I(Z, Y) = \sum_{z \in \{Z, \neg Z\}} \sum_{y \in \{Y, \neg Y\}} P(z, y) \log_2 \frac{P(z, y)}{P(z)P(y)}$$
$$= 0.45 \log_2 \frac{0.45}{0.55 \times 0.5} + 0.1 \log_2 \frac{0.1}{0.55 \times 0.5} + 0.05 \log_2 \frac{0.05}{0.45 \times 0.5} + 0.4 \log_2 \frac{0.4}{0.45 \times 0.5}$$
$$\approx 0.3973$$

4. Which undirected edges will be selected by the Chow-Liu algorithm as the maximum spanning tree? (5 pts)

   The candidate edges are (X, Y), (X, Z) and (Y, Z). The Chow-Liu algorithm first selects the edge with the largest weight (in this case, our edge weights are the mutual information), which will be (Y, Z) with a mutual information of 0.3973.
   The next largest edge will be (X, Y) with a weight of 0.2781.
   The next largest edge is (X, Z) with a weight of 0.1328, but since adding (X, Z) creates a cycle in the graph, we do not add it.

   So the maximum spanning tree generated is $G = \Big(V := \{X, Y, Z\}, E := \{(Y, Z), (X, Y)\}\Big)$

5. Root your tree at node X, assign directions to the selected edges. (5 pts)



$P(X) =$

T	F
0.5	0.5

$P(Y \mid X) =$

X	T	F
T	0.8	0.2
F	0.2	0.8

$P(Z \mid Y) =$

Y	T	F
T	0.9	0.1
F	0.2	0.8

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*.