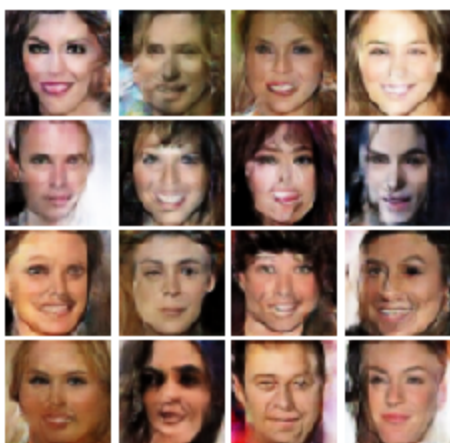


Iter: 14700, D: 0.3034, G:3.74



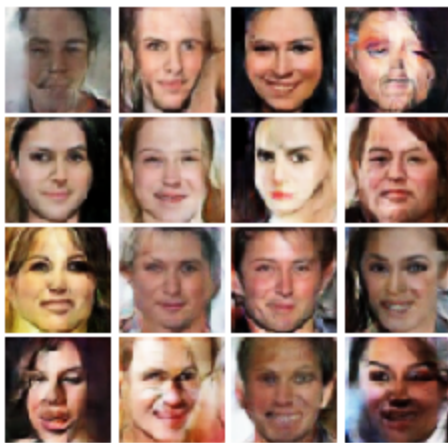
Below pictures are three final results from LSGAN of epoch 15.
 Iter: 14200, D: 0.1239, G:0.5692



Iter: 14400, D: 0.04065, G:0.2812



Iter: 14600, D: 0.03472, G:0.341



Discuss any differences you observed in quality of output or behavior during training of the two GAN models.

Above pictures from GAN and LSGAN proves that LSGAN lead to have much less loss on discriminator and generator compared to GAN. The significant difference between them at the start was that the images of GAN were dominated by the color of pink when those LSGAN was dominated by the color of green. At the end, having much less loss, we can conclude that LSGAN has the better quality of output compared to GAN.

Do you notice any instances of mode collapse in your GAN training (especially early in training)? Show some instances of mode collapse (if any) from your training output.

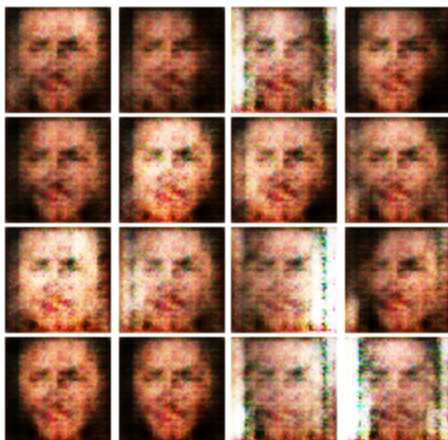
Mode collapse occurs when a generative model begins to only generate a single mode or a few modes such as mode collapse occur when there are duplicate faces generated. As below images are beginning of GAN train, we can tell that there are mode collapse since we can see few of images are similar are same

to each other. For Iter of 300, only a significant difference is the background, and faces look similar which tells that duplicate faces can be generated. However, this is solved as there are more trains since there were no duplicate faces at the end.

Iter: 150, D: 0.8672, G:1.586

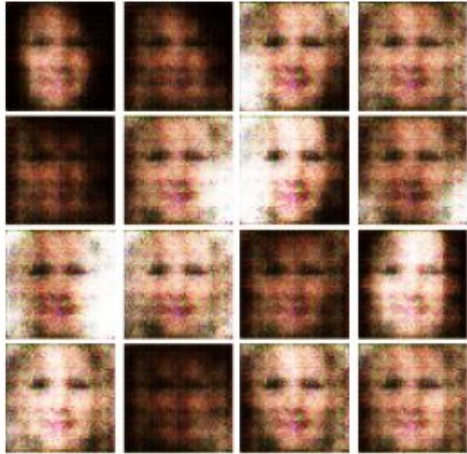







Iter: 300, D: 1.979, G:3.622



Discuss briefly how/whether spectral normalization helps generate higher quality images in your implementation. Ideally, you should show samples from models with and without normalization.

We can definitely state that spectral normalization helps generate higher quality images in the implementations. This can be proved by looking at first images that have decent face pictures for each epoch to see how qualities of images are different. Below table tells that pictures with spectral normalization always have significant face look while pictures with no spectral normalization sometimes have white blurs in the center or strong color of pink in the face, which should not happen in the center. This tells that quality is much better with using spectral normalization.

Epoch	With Spectral Normalization	Without Spectral Normalization
<p>1</p> <p>Iter: 150</p>		
<p>2</p> <p>Iter: 1050</p>		
<p>3</p> <p>Iter: 2100</p>		

Extra credit: If you completed the extra credit for this portion, explain what you did (describing all model changes and hyperparameter settings) and provide output images.

Part 2 (generation):

Give the hyperparameters for your best network on classification task below. Note any other changes you made to the base network in addition to the hyperparameters listed in the table below.

Hyperparameter	Value
RNN type:	GRU
Number of layers:	1
Hidden layer size:	100
Learning rate:	0.0001

Between the basic model of RNN and GRU with few changes, it came out to be that few changes with GRU made better results than RNN. Both of the models were tested with a change of learning rate 0.0001. GRU has an extra layer in the linear. Final outputs of loss for both of them were very similar. With three changes: RNN type, learning rate and number of layer, I compared the results between them, and RNN outputted train loss 1.8202 and test loss 1.9241. For GRU, I got 1.6378 for train loss and 1.8250 for test loss.

Give an example 1000 character output from your network:

The come,

And on theneswall of this swailow?

BENEL IPINI:

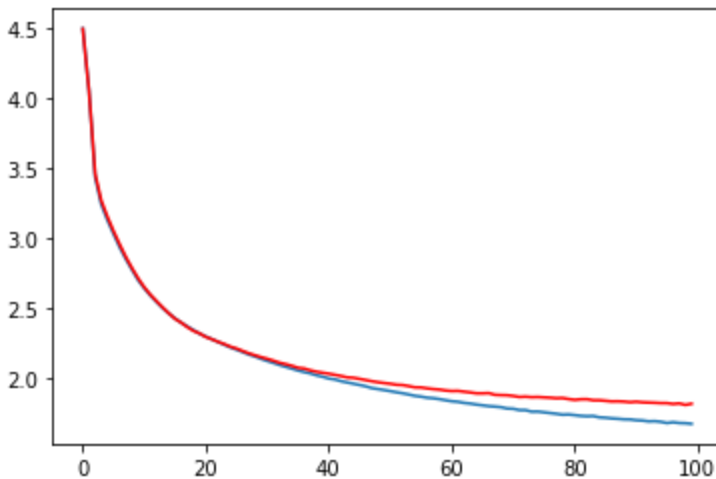
Hanced we with not the coust old with offern thy come senciount dear
kingly I the grather do lown it;

And be so be he to bedly to here quanters of rowness for thought of
fess, lived thee, thin reep his laits and his, is doing dit will have
breenthath manigne?

Tith thing mondere sand give
And on up a dlesp's are you this hamford:
I this depomences to saile other dight strifher be wow pronged am the
blow
Mortion thee piny pese afto; your hore on ross a way from this now as
an king; this moran, wedber,
The terray shall for the other he suking
Well Perpote
First say to the cammente.

Serith dives is of my comnthat you couttre
thou, I ment this here the it the goot bited you our love so in thus
not witce a prears hand trim, do do then tr fe told shands in be to
the sirst ak good of it your, and now weads upon that enery lo he
crastens,
Who minging reard, is the tood of reto like their hold am himp gust
probes, at this the soy reso now

Insert the training & test loss plot from your RNN generation notebook below:



Extra credit: If you completed the extra credit for this portion, describe where your dataset came from, give an example from your training dataset (1000 characters), give an example output from your model trained on the dataset (1000 characters), and detail the hyperparameters you used to train a model on the dataset.

Part 2 (classification):

Give the hyperparameters for your best network on classification task below. Note any other changes you made to the base network in addition to the hyperparameters listed in the table below.

Hyperparameter	Value
RNN type:	LSTM
Number of layers:	2
Hidden layer size:	300
Learning rate:	0.001

The test began with the LSTM model of RNN since LSTM seems to be better than basic RNN and little better than GRU. I first tried with a learning rate of 0.0001 with an increase of hidden size up to 2000. Since I thought training more can lead to high accuracy, I also set the number of iterations to 20000. This gave me high accuracy on the train up to almost 98, but gave low accuracy. This can be due to the overfit on the training, but I did not detect any overfit on the graph of loss, so I increased the chunk length more up to 200. Even though I still got high accuracy on the train, I did not get any near the desired test accuracy. The reason why it is low is that over-training can actually lead to not learning. Believing that overtraining is the problem, I reset the hyperparameter to the original and test individually. Decreasing the chunk length showed the improve on test accuracy so I found that 25 gives highest accuracy. After this, I changed my hidden size to 300, which was a little higher than the original of 100, but not as great as 2000 to avoid over-training. Also the number of layers of 2 showed improvement. Since the layer is greater than 1, I was able to use dropout in the LSTM model, so I used 0.1 and added one more linear layer. Adding dropout linear did not show any improvement, but adding 0.2 of dropout after embedding showed the improvement on test accuracy. I thought that 0.0001 would be a good learning rate; however, it was too low, so I increased the learning rate until accuracy got higher and stopped when it got lower, which the answer was 0.001. The test was based on a number of iterations 1000. With 1000, accuracy was about 0.87, but losses on the train set showed that it can still get decreased, so I set number 10000, which will give enough to learn. By this I was able to get a test accuracy of 0.896, and 0.90320 on the kaggle test. Since these two numbers are very close, I can see that I have passed the benchmark.

You should reach the Kaggle accuracy benchmark with your Kaggle submission. Your notebook evaluation results should be similar to your performance on Kaggle. Insert the confusion matrix image outputted from your best model, and report the corresponding accuracy:

Passed the kaggle rnn benchmark of 0.89840 by 0.90320

Test accuracy: 0.896

