Murat Sezgin Baloğlu HW-3 2469518

Q1)

An Autoencoder is a network trained to attempt to copy its input to its output. It is a non linear generalization of PCA (Compressed representation is a low dimensional representation of input.

Autoencoding " can be considered a data compression algorithm where the compression and decompression functions are :

- Data specific
- Lossy
- Learned automatically from examples rather than engineered by a human.

There are, basically, 7 types of autoencoders [1]:

- Denoising autoencoder : Denoising autoencoders create a corrupted copy of the input by introducing some noise. This helps to avoid the autoencoders to copy the input to the output without learning features about the data.
- Sparse Autoencoder : Sparse autoencoders have hidden nodes greater than input nodes. They can still discover important features from the data.
- Deep Autoencoder : Deep Autoencoders consist of two identical deep belief networks, one network for encoding and another for decoding. Typically deep autoencoders have 4 to 5 layers for encoding and the next 4 to 5 layers for decoding.
- Contractive Autoencoder : The objective of a contractive autoencoder is to have a robust learned representation which is less sensitive to small variation in the data. Robustness of the representation for the data is done by applying a penalty term to the loss function.
- Undercomplete Autoencoder : The objective of undercomplete autoencoder is to capture the most important features present in the data. Undercomplete autoencoders have a smaller dimension for hidden layer compared to the input layer.
- Convolutional Autoencoder : Autoencoders in their traditional formulation does not take into account the fact that a signal can be seen as a sum of other signals. Convolutional Autoencoders use the convolution operator to exploit this observation.
- Variational Autoencoder : Variational autoencoder models make strong assumptions concerning the distribution of latent variables. They use a variational approach for latent representation learning, which results in an additional loss component and a specific estimator for the training algorithm called the Stochastic Gradient Variational Bayes estimator.

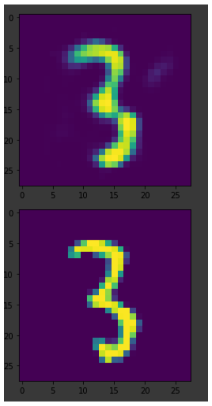
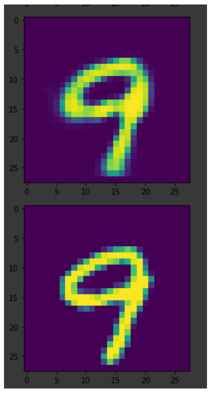
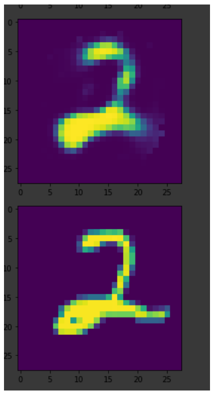
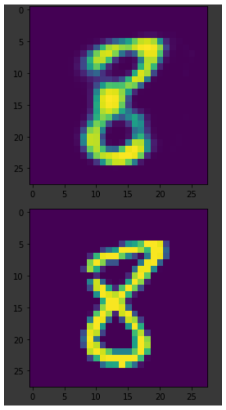
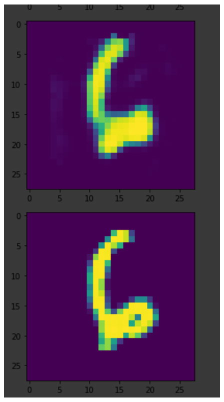
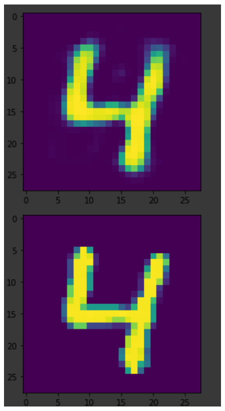
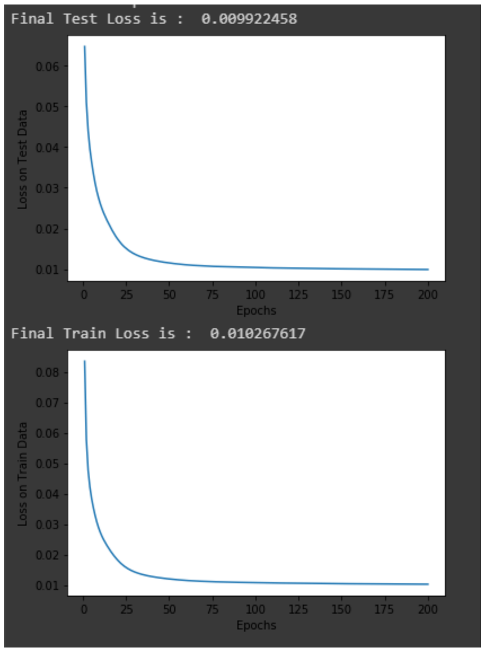
1) ReLu is used as activation function.
2) ReLu is replaced with sigmoid and background became darker. Probably due to ReLu is a greater valued function with same input.
3) Learning rate increased to 0.3 from 0.1. Loss is improved.
4) Tanh is used instead of sigmoid, loss improved. Probably backward propagation is increased.
5) Epoch size increased to 300 from 100. Images are slightly better.
6) Scheduled lr is experimented, loss is still around 0.017.
7) All the previous act. Funcs. were applied on only to hidden layer and output layer doesn't have act. Func. At this step, ReLu is used in hidden layer, sigmoid is used in output layer. Satisfactory result achived.

Murat Sezgin Baloğlu HW-3 2469518

Hyperparameters:

Network is already defined, Adagrad optimizer with scheduled lr is used. lr = 0.5 upto 70th epoch, lr = 0.3 for the remaining 130 epoch that adds up to a total number of 200 epochs with batch size = 100.

Results :

```
Final Test Loss is :  0.009922458
```

```
Final Train Loss is :  0.010267617
```

Q2)

GANs :

- An adversarial process for estimating generative models
- Consist s of 2 simultaneously trained models : a generative model G and adiscriminator model D.
- The generative model G takes random noise as input and generates datacandidates
- Discriminator model D tries to distinguish which is real data.

I have used tf.nn.sigmoid_cross_entropy_with_logits as loss function. This function takes logits of generator and discriminator and computed the loss as if they have an label which actually they have. Real images have labels as 1, and generated images have labeled as 0 for discriminator.

Steps :

Due to limited time left to write this report I copy + paste my development logbook in its raw form. Couldn't rephrase them. Any points welcome :

1) gen loss :0, disc_loss: inf

2) mb 100 den 10'a düştü, gen loss 0 -> nan, disc_loss inf -> nan

3) generator_input dist. uniform -> normal bişi değişmedi.

4) relu -> tanh bişey değişmedi

5) hiddenlar relu, output sigmoid -> tanh, genloss : nan, disc_loss: -0.6931 stuck

6) variable scope olayı incelendi, disc ve gen için variablelar sürekli güncellenmiyor.

7) adagrad to adam generator gene nan, discriminator 0.6931'de stuck ancak daha geç stuck oluyor.

8) generator, discriminatorun 5 katı çalışıyor. sonuç : FAIL AGAIN

9) generator, discriminatorun 5te biri kadar çalışıyor sonuç : FAIL AGAIN

10) I couldnt fix the weight sharing issue while i was using tf.layers.dense, hence after some time i gave up and used the

low level functions tf.nn.matmul etc. with variables initialized by hand. After that I was able to train the network.

However, I still wonder whether there is a way to train the network while using tf.layers.dense module

11) after 70th epoch, gen loss went inf. then added an epsilon to disc. output. helped but not a lot.

12) disc. output is nan after 40th epoch. lr 0.0003 to 0.0001. rsult : better but still some NaNs occur, generator

only generates 9s xd.

13) loss function değişti, sonuç : NaN olmuyor ancak network 8-9 üretmeye meyilli

14)mb_size 128 -> 32 sonuç : ilerleme olmadı

15) generator,works 3 times of disc.: loss is smaller but outputs are not sufficent

16) back to old training func. still gen works 3 times of disc. works. PEK DEĞİŞMEDİ

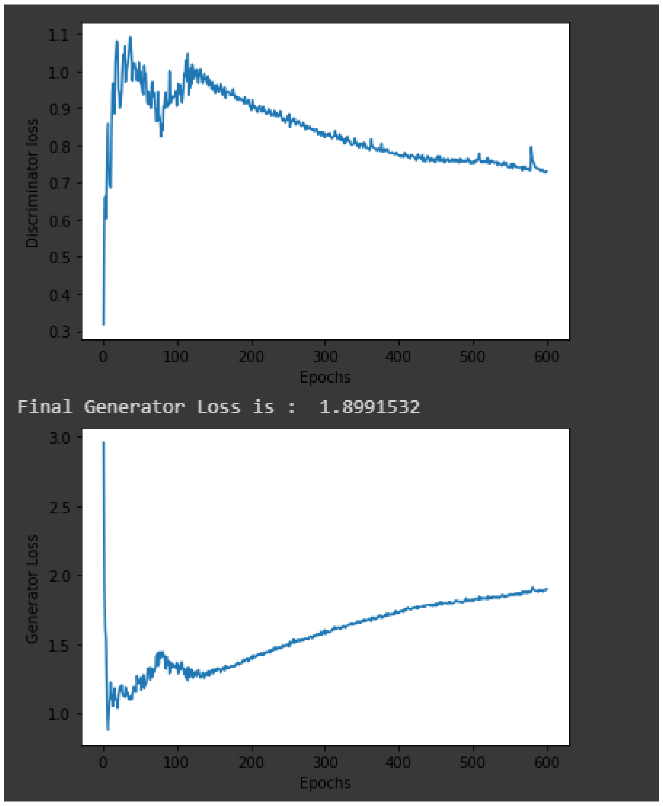
18) 600 epoch mbsize 256 : meh

19) gen worked 10 times disc. worked. converged to 0s

Hyperparameters:

Adamoptimizer with lr = 0.0001 used. Batch size is 512 and epoch is 600

Time and Loss Graphs:

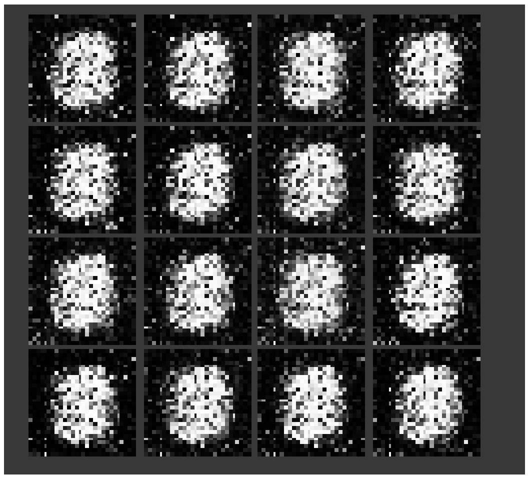
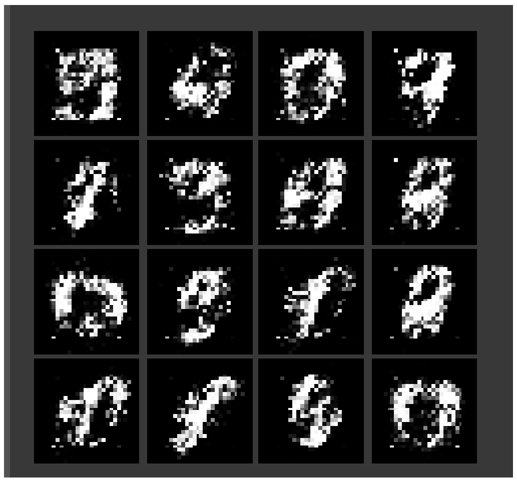
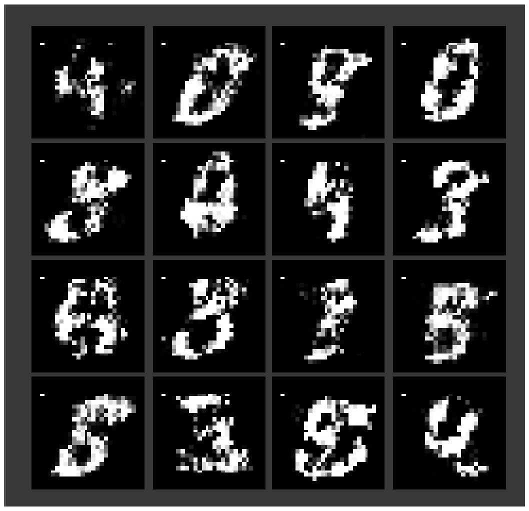
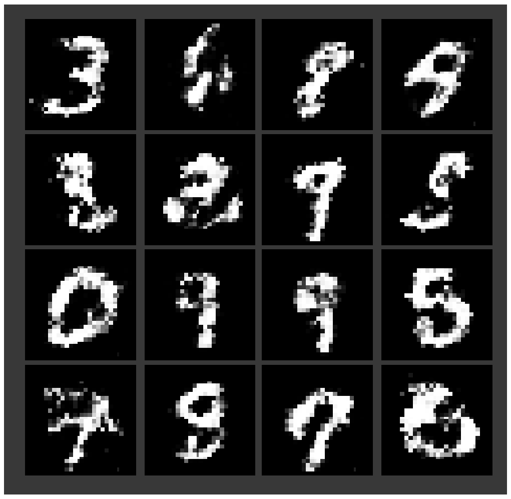
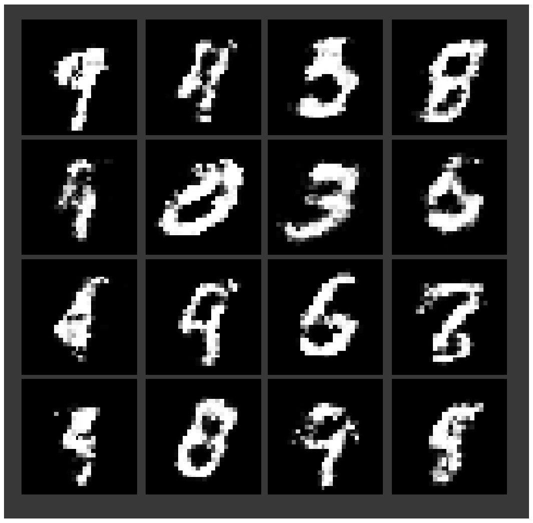
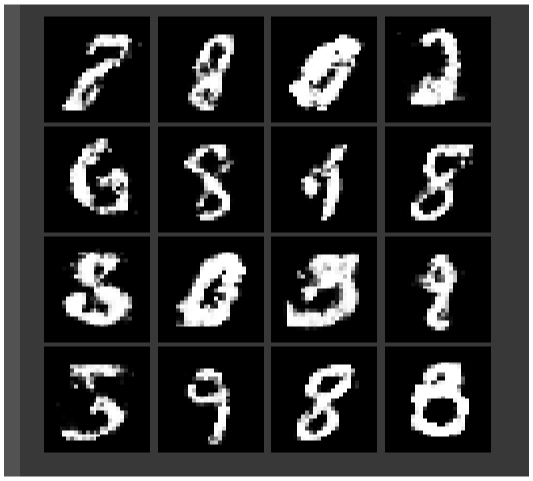


```
Final Generator Loss is :  1.8991532
```

Graph on the top belogns to discriminator and its final loss is 0.72. From the graph we can see that, loss of the generator and discriminator are inversely correlated, a decrease in generator’s loss is resulting in a increase in discriminator’s loss and vice versa. Resembling the total entropy of the universe.

Murat Sezgin Baloğlu HW-3 2469518

Results:

Q3)

RNNs:

- Recurrent networks share parameters :
  - Each member of the output is a function of the previous members of the output
  - Each member of the output is produced using the same update rule applied to the previous outputs
  - This recurrent formulation results in the sharing of parameters through a very deep computational graph
- Simple RNN consists of :
  - Input layer
  - Hidden layer with recurrent connections
  - Output layer

LSTMs:

- Have explicit memory cells to store short-term activations the presence of additionalgates partly solves the vanishing gradient problem.
- The key to LSTMs is the cell state.
- The LSTM have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionnaly let information go through.

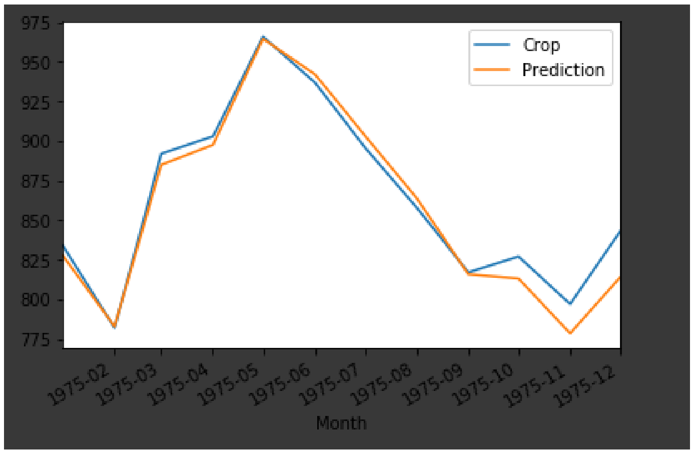
GRUs:

- GRU is modification of LSTM.
- Forget and input gates are combined into a single "update gate".
- Its model is simpler that standart LSTM models.

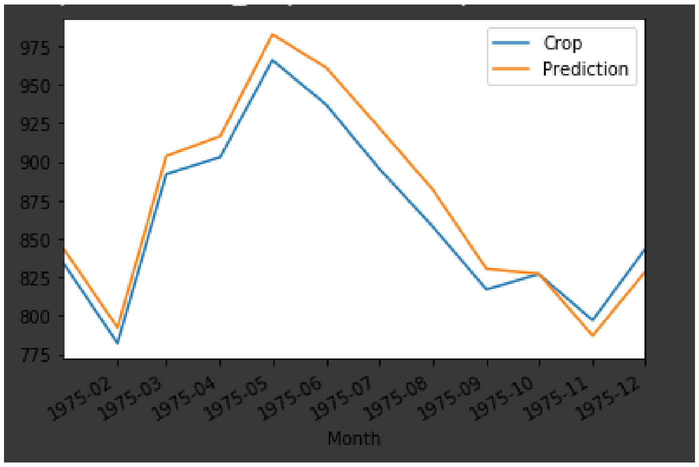
GRUs and LSTMS are modificated versions of RNSs that allow gradients to flow much better and mitigates the vanishing gradient problem.

Steps:

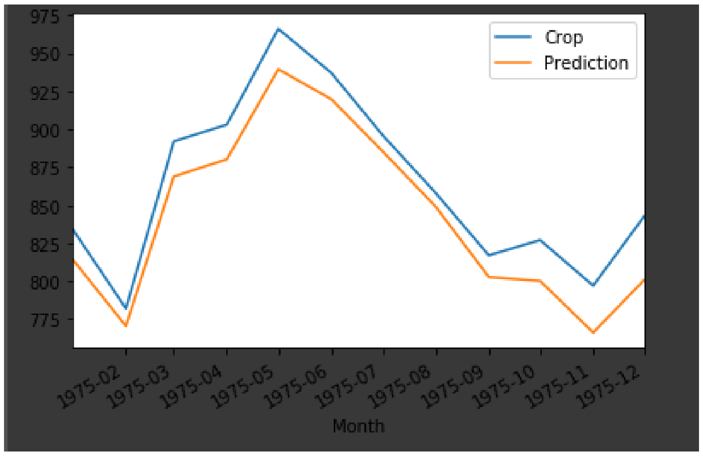
Optimizer, network, parameters were already defined beforehand. I have only iterated through test part. Implementing sliding window approach on last 12 training samples and newly generated predictions were both hard to figure out and debug since thier dimensions doesn't easily match.



LSTM Cell with OutputProjectorWrapper final MSE : 0.021 Training dur. 44 sec.



RNN Cell with OutputProjectorWrapper final MSE : 0.030 Training dur: 33 sec.



GRU Cell with OutputProjectionWrapper final MSE : 0.077 Training dur: 47 sec.

LSTM Cell has the best performance among all. From performance point of view this result is not surprising. Moreover, RNN is the fastest one to traing as expected since LSTM and GRU are more complex in terms of computation. However, GRU’s training duration expected to be shorter than LSTM’s but that is not the case in the experiment results.

We can say that, when we move from RNN to LSTM, we are introducing more & more controlling knobs, which control the flow and mixing of Inputs as per trained weights and thus, bringing in more flexibility in controlling the outputs.

LSTM gives us the most Control-ability and thus, Better Results. But also comes with more Complexity and Operating Cost. [2]

GRU couples forget as well as input gates. GRU use less training parameters and therefore use less memory, execute faster and train faster than LSTM's whereas LSTM is more accurate on dataset using longer sequence. In short, if sequence is large or accuracy is very critical, please go for LSTM whereas for less memory consumption and faster operation go for GRU. If you donot have much floating point operations per second (FLOP's) to spare switch to GRU. LSTM has three values at output (output, hidden and cell) whereas GRU has two values at output (output and hidden). [3]