HW7 – Report

Achyuthan Unni Krishnan, Kishor Sabarish

1. VAE

The encoder of the VAE was defined as shown in the following image:

```
class Encoder (nn.Module):
    def __init__ (self):
        super(Encoder, self).__init__()
        # TODO initialize layers

        self.L1 = nn.Linear(784, 512)
        self.L2 = nn.Linear(512,256)
        self.L3 = nn.Linear(256,32)
        self.drop = nn.Dropout(p=0.5)

    def forward (self, X):
        # TODO execute layers and return result
        X = F.tanh(self.L1(X))
        X = self.drop(X)
        X = F.tanh(self.L2(X))
        mu = (self.L3(X))
        sigma = (self.L3(X))

        return mu, sigma
```

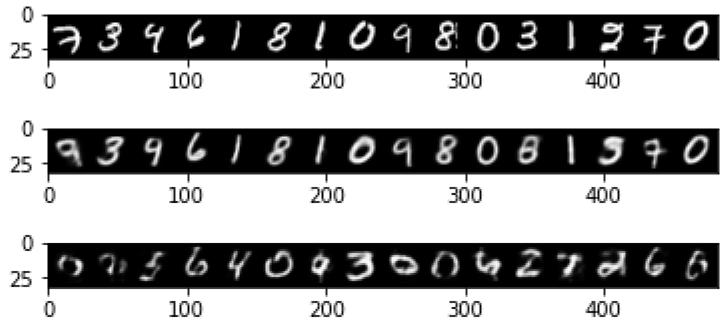
The decoder of the VAE was defined as shown in the following image:

```
class Decoder (nn.Module):
    def __init__ (self):
        super(Decoder, self).__init__()
        # TODO initialize layers
        self.d1 = nn.Linear(32, 256)
        self.d2 = nn.Linear(256,512)
        self.d3 = nn.Linear(512, 784)

    def forward (self, Z):
        # TODO execute layers and return result
        Z = F.tanh(self.d1(Z))
        Z = F.tanh(self.d2(Z))
        Z = (self.d3(Z))
        Z = F.sigmoid(Z)

        return Z
```

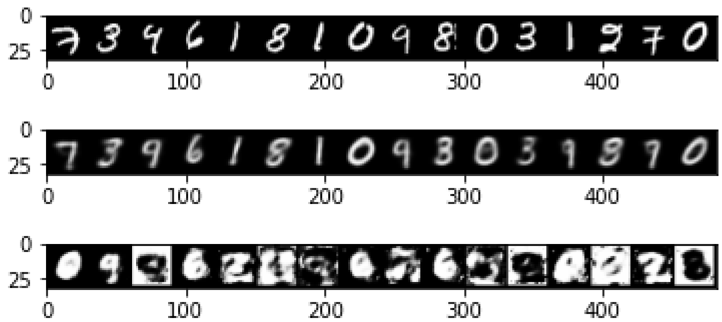
Below is a snippet of the training for the VAE. All trainings were run for 50 epochs. The loss printed is the sum of reconstruction and KL divergence losses.

```
Epoch 30
84.08832024147728
Epoch 31
83.95664691051137
Epoch 32
83.98458158735795
Epoch 33
83.7554681196733
Epoch 34
83.6363697709517
Epoch 35
83.54702003728693
Epoch 36
83.54544573863636
Epoch 37
83.38251393821022
Epoch 38
83.3385424272017
Epoch 39
83.23137761008523
Epoch 40
83.11197724609374
Epoch 41
83.11410267223012
Epoch 42
83.04217843572444
Epoch 43
82.97887238991477
Epoch 44
82.92600430575284
Epoch 45
82.83873409090909
Epoch 46
82.87027643821023
Epoch 47
82.80291663707386
Epoch 48
82.89972421875
Epoch 49
82.9321150834517
Epoch 50
82.75378301669033
```

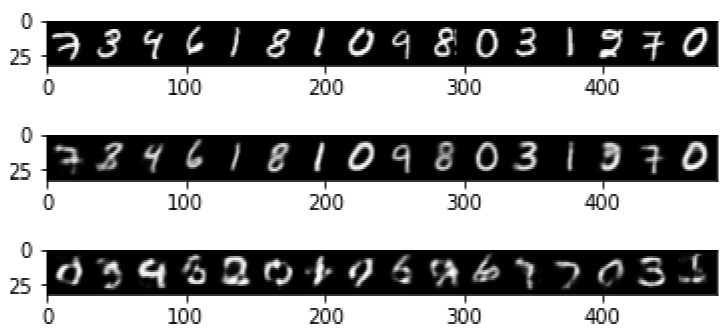
The VAE was implemented for multiple 1/L values. Below is the Original, Reconstructed and Decoder outputs of the VAE for 1/L = 0.45. Here, the reduction argument for the BCE loss function was given ‘sum’.

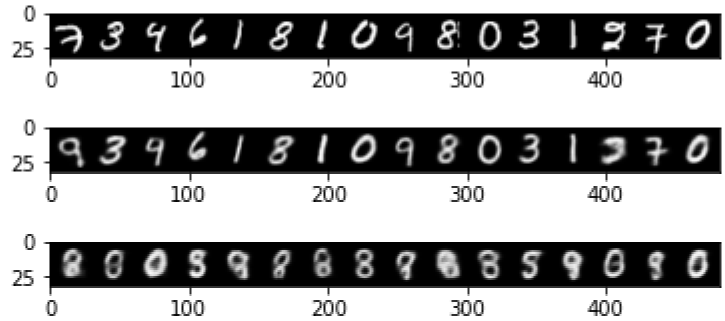


The same VAE network with default argument for reduction parameter for BCE loss gave less than ideal results as shown below.



Similarly, the images for I/L= 1 and 0 are presented in the same format as above.





2. GANS

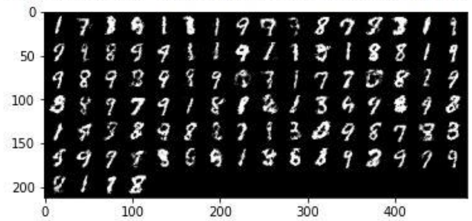
The generator of the GANS was defined as follows:

```
class Generator (nn.Module):
    def __init__ (self):
        super(Generator, self).__init__()
        # TODO initialize layers
        self.L1 = nn.Linear(100, 128)
        self.L1_ = nn.Linear(128,256)
        self.L2 = nn.Linear(256, 512)
        self.L3 = nn.Linear(512, 600)
        self.L4 = nn.Linear(600, 784)
        self.bn0 = nn.BatchNorm1d(num_features=128)
        self.bn1 = nn.BatchNorm1d(num_features=256)
        self.bn2 = nn.BatchNorm1d(num_features=512)
        self.bn3 = nn.BatchNorm1d(num_features=600)

    def forward (self, Z):
        # TODO execute layers and return result
        Z = F.relu(self.L1(Z))
        Z = self.bn0(Z)
        Z = F.relu(self.L1_(Z))
        Z = self.bn1(Z)
        Z = F.relu(self.L2(Z))
        Z = self.bn2(Z)
        Z = F.relu(self.L3(Z))
        Z = self.bn3(Z)
        Z = F.sigmoid(self.L4(Z))

        return Z
```

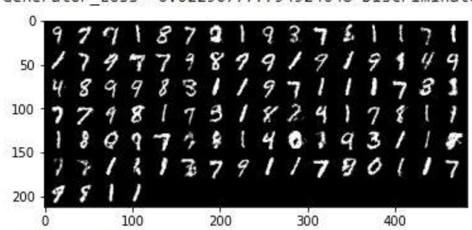
The discriminator of the GANS was defined as follows (NOTE: Batchnorm for discriminator was attempted but it failed to deliver good results):

```python
class Discriminator (nn.Module):
    def __init__ (self):
        super(Discriminator, self).__init__()
        # TODO initialize layers
        self.fc1 = nn.Linear(784, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 1)
        self.drop = nn.Dropout(p=0.5)
        self.bn0 = nn.BatchNorm1d(num_features=1024)
        self.bn1 = nn.BatchNorm1d(num_features=512)

    def forward (self, X):
        # TODO execute layers and return result
        X = self.drop(X)
        X = F.relu(self.fc1(X))
        #X = self.bn0(X)
        X = F.relu(self.fc2(X))
        #X = self.bn1(X)
        X = F.sigmoid(self.fc3(X))

        return X
```

Below is an example of the training of the GANS network. The Discriminator and Generator losses are printed after each epoch.



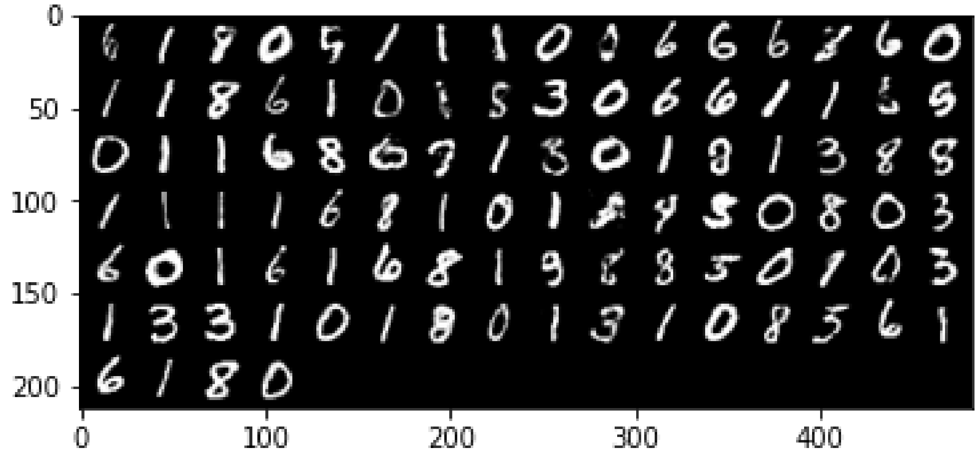
```
Epoch ::  11
Generator_Loss= 0.02921457933729345 Discriminator_Loss 0.004699573575366627
Epoch ::  12
Generator_Loss= 0.027649533800645307 Discriminator_Loss 0.005087453528967771
Epoch ::  13
Generator_Loss= 0.026691346354918048 Discriminator_Loss 0.005380202317779715
Epoch ::  14
Generator_Loss= 0.02526992661736228 Discriminator_Loss 0.005597603309696371
Epoch ::  15
Generator_Loss= 0.025032208728790282 Discriminator_Loss 0.005700738873806867
Epoch ::  16
Generator_Loss= 0.02454564935944297 Discriminator_Loss 0.005749093503301794
Epoch ::  17
Generator_Loss= 0.0242135822838003 Discriminator_Loss 0.005776125210523605
Epoch ::  18
Generator_Loss= 0.02399353002201427 Discriminator_Loss 0.005901965965466066
Epoch ::  19
Generator_Loss= 0.023872564086047085 Discriminator_Loss 0.005974472313035618
Epoch ::  20
Generator_Loss= 0.022907777794924648 Discriminator_Loss 0.006037305596199903
```



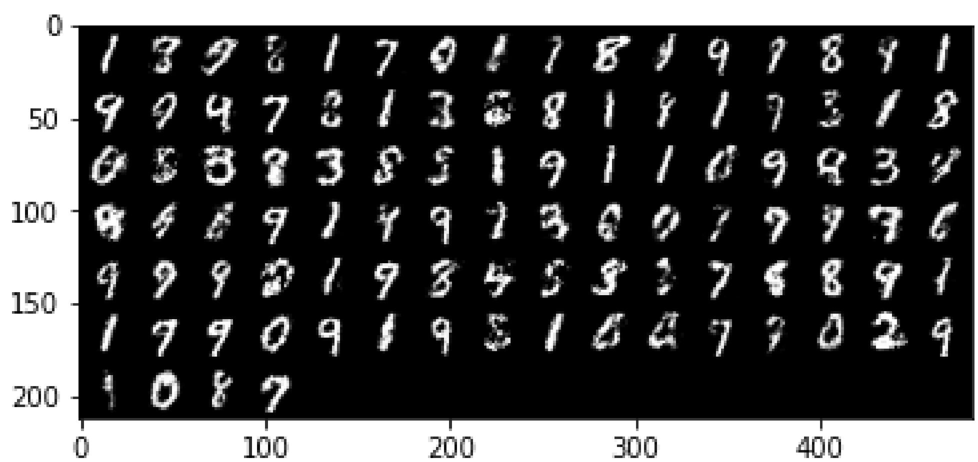
```
Epoch ::  21
Generator_Loss= 0.022928698418357155 Discriminator_Loss 0.006099061618068015
```

Several training methods were used to get the desired results.

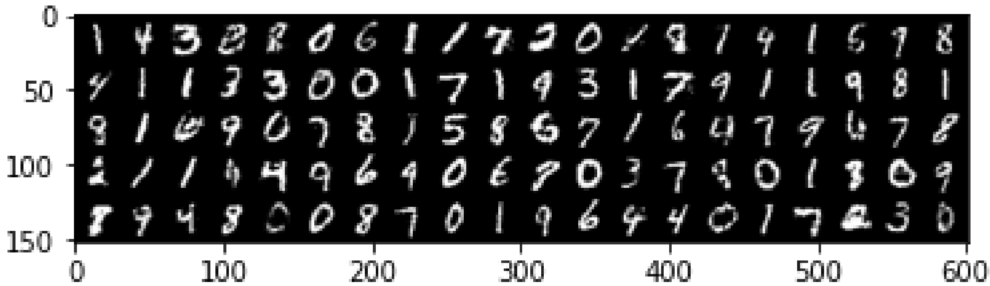
1. First, we trained discriminator for epoch numbers ending in 1-4 and epoch numbers ending in 5-9 were when generator was trained. Every 10th epoch both discriminator and generator are trained simultaneously. Below are the generated images after 200 epochs. It failed to generate 2’s and the images weren’t clear.



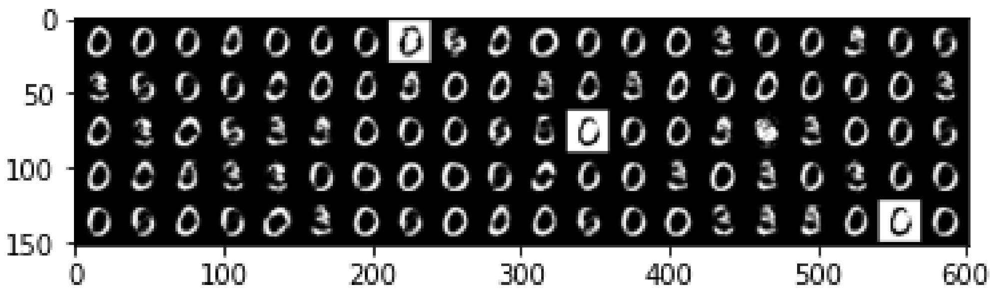
2. When no explicit learning rates were set and Discriminator was stopped from training for every 500th minibatch in each epoch, the network learned to generate numbers by the 20th epoch. We also noticed that changing the latent size of generator also matters. For us, best results were observed when the initial latent sizes were around 80-100. Below is an image of training with these parameters.

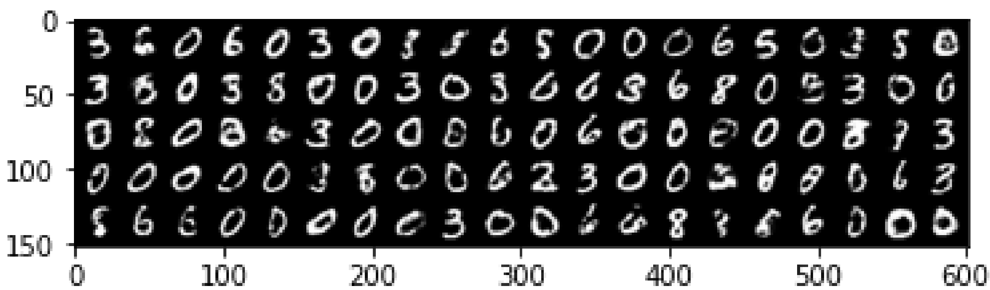


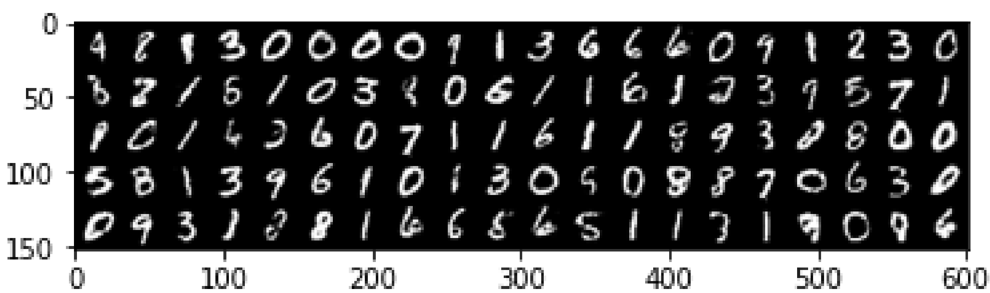
3. However, despite the network learning to create most of the digits, we observed that the digits were quite noisy. Observing the losses while training, we noticed that slowing down the convergence of Generator while dropping every 50th minibatch for 200 epochs of training showed better results. Below is the results when we set Generator Learning rate to 5e-5. This helped us generate in our opinion clearer and diverse numbers.

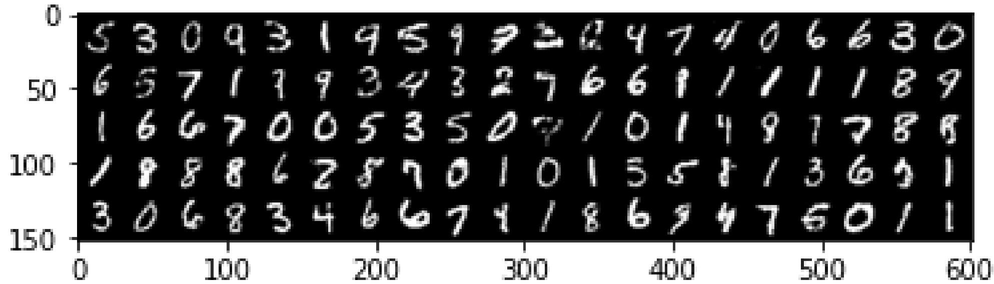


The following images portray the generator's output for every 10th , 50th , 100th and 200th epoch.

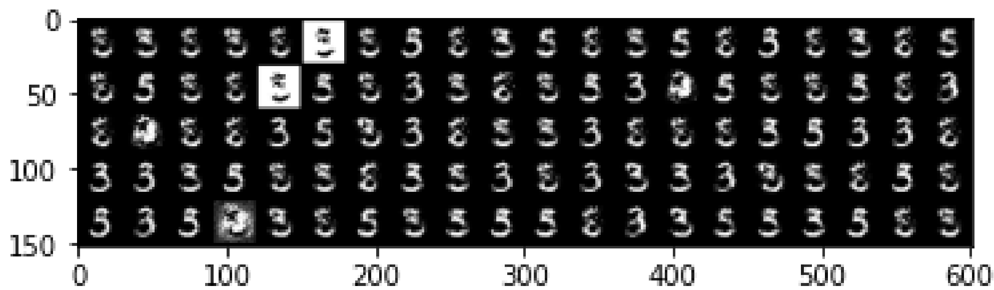


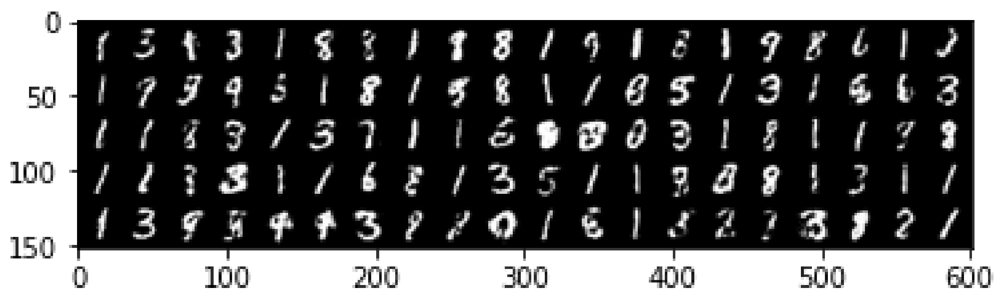


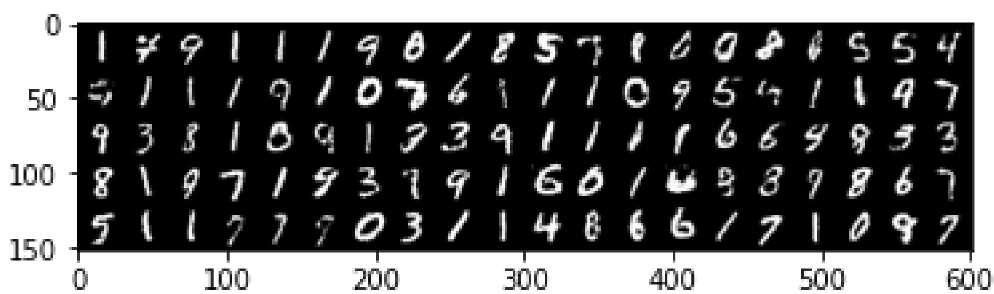


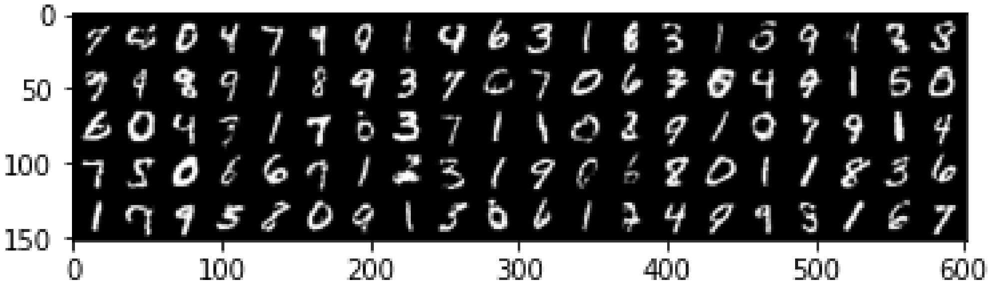


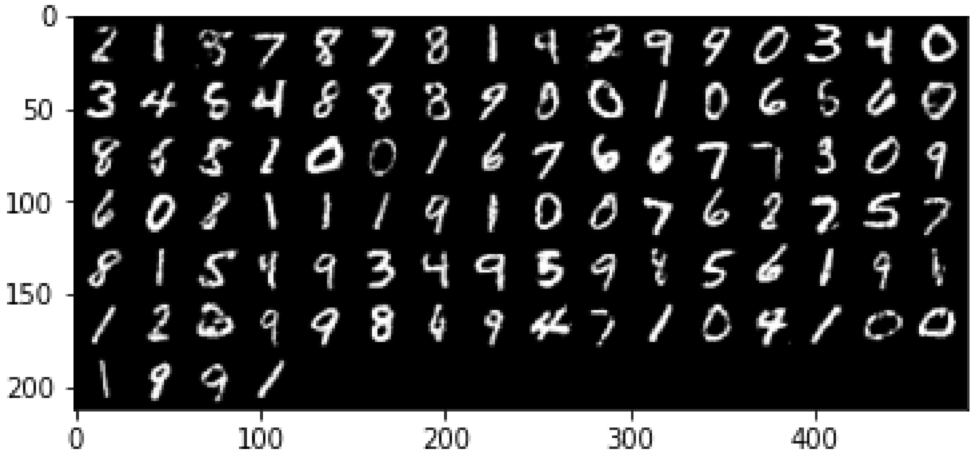
4. We also trained the GANS with skipping the Discriminator every 5th epoch and below are the 10th , 50th , 100th , 150th and 200th epoch.








0
50
100
150
0
100
200
300
400
500
600


0
50
100
150
200
0
100
200
300
400