CAP 6610: Final Project Report

UFID:0499-5847

Vasireddy Sai Datta Vara Prasad

The new breakthrough focused on neural networks is the Generative Adversarial Networks (GAN) and Variational Auto Encoder. Both are generative models used to produce data based on previously defined data for the purpose of generating data basing on the previously existing data.

MNIST DATASET:

The MNIST database is a broad handwritten digit database that is widely used to train different systems for image processing. The database is also commonly used in machine learning for teaching and research.

There is a training set of 60,000 examples in the MNIST handwritten digits collection, accessible from this list, and a test set of 10,000 examples. It is a subset of NIST's accessible wider package. The digits in a fixed-size picture have been size-normalized and oriented.

NIST's original black and white (bilevel) images have been normalized in size to fit in a 20x20 pixel box while retaining their aspect ratio. Because of the anti-aliasing strategy used by the normalization algorithm, the resulting images contain gray amounts. By measuring the center of mass of the pixels and converting the image, the images were centered in a 28x28 image to place this point at the center of the 28x28 region.

Code Snippet of downloading the MNIST data set using tensor flow backend :

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

Generative Adversarial Networks (GAN):

Generative modeling is an unsupervised machine learning task that involves the automated discovery and learning of the regularities or patterns in input data in such a way that new examples that could have been plausibly derived from the original dataset can be created or developed by the model.

GANs are an exciting and rapidly evolving field, delivering on the promise of generative models in their ability to generate realistic examples across a variety of problem areas, especially in image-to-image translation tasks such as translating summer-to-winter or day-to-night photographs, and generating photorealistic images of objects, scenes, and people that cannot even be told are fake.

For example, Naive Bayes works by summarizing the probability distribution of each input variable and the output class. When a prediction is made, the probability for each possible outcome is calculated for each variable, the independent probabilities are combined, and the

most likely outcome is predicted. Used in reverse, the probability distributions for each variable can be sampled to generate new plausible (independent) feature values.

Let us define the notations we will be using to formalize our GAN,

Pdata(x) -> the distribution of real data
X -> sample from Pdata(x)
P(z) -> distribution of generator
Z -> sample from p(z)
G(z) -> Generator Network
D(x) -> Discriminator Network

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

GAN Generator:

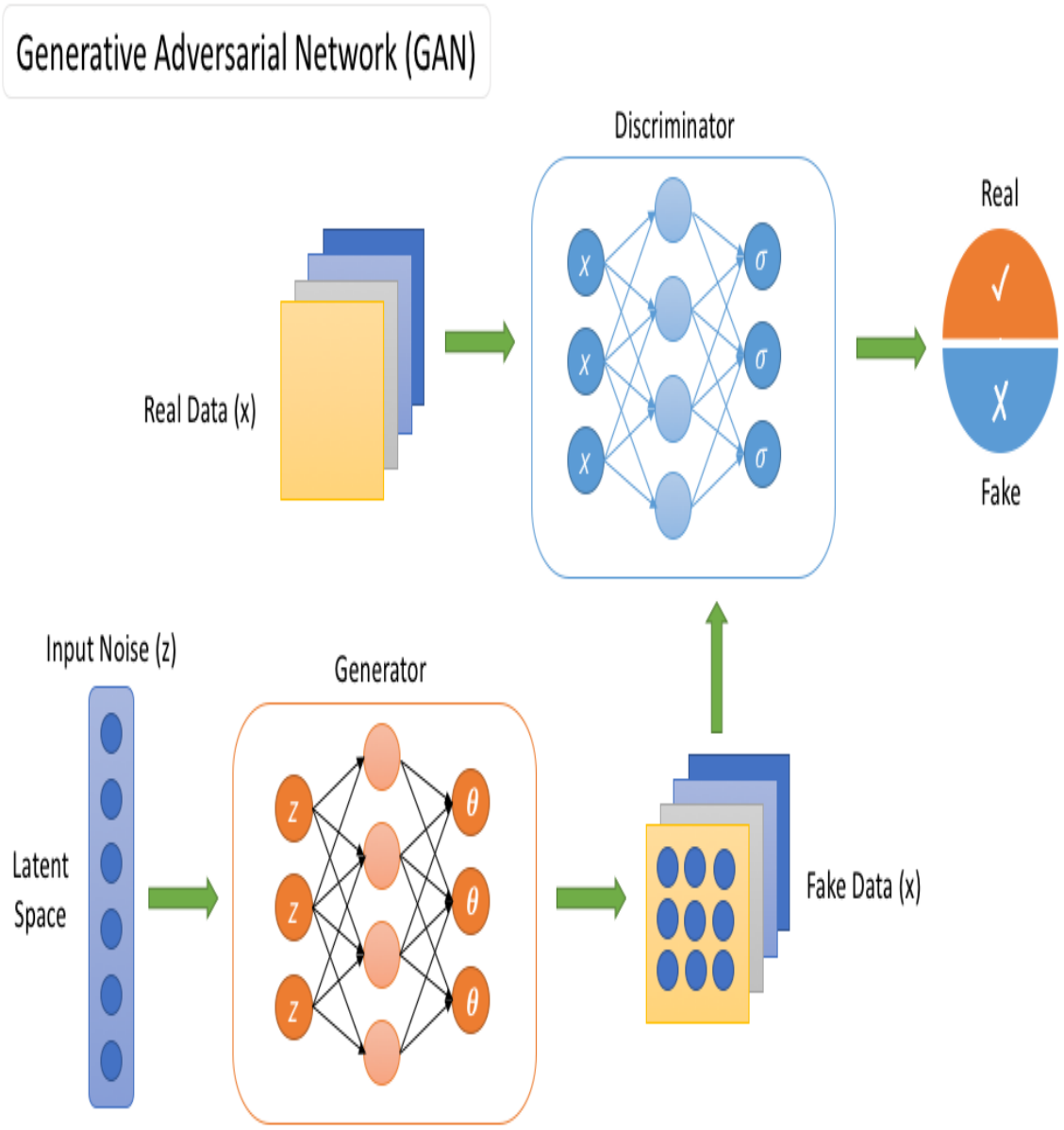
The generator model takes an irregular vector of fixed length as information and produces an example in the field. The vector is derived from a Gaussian distribution at random, and the vector is used to seed the generative loop. This multidimensional vector space will concentrate on the demanding field when the training is done, framing a compact expertise in data distribution.

The following steps address the working of the generator: The relation between latent space and output is formed by the generator. The neural network is generated at this stage, which is trained on the input to generate the image for the most given input. The generator is then trained in a way that connects both generator and discriminator. Now when the training is finished, the generator is ready for deduction.

GAN Discriminator:

The discriminator model takes a sample from the area of the input and predicts whether it is genuine or fake (produced). The model is learned from the training dataset. The produced models are yield by the generator model. When the training is completed, the discriminator model is no longer considered because the rest of the work is done by the generator**.**

The following steps discusses how the discriminator works: Initially, a convolutional neural organization is created to distinguish real and fake data. Now the fake dataset is generated by the generator using the real data as the input. The discriminator model is then trained on the real and fake datasets. The discriminator is adjusted in learning the real and fake data accordingly with the generator. if the discriminator is too acceptable, the generator will wander. In more machine learning and mathematical terms GAN can be explained as follow. Given X is the input of the data and Y is the output label for the given input. Generative models, they catch the joint likelihood p (X, Y), or just p(X) with inputs of no labels. Discriminative models catch the restrictive likelihood p (Y | X).



Comparing the output:

With the increase in the epoch count we can observe several different changes in the numbers generated and the accurate behavior of decoder. In the first epoch we can see a complete dotted plot by which we cannot find which number is everyone is. But from epoch 20 we can observe number like structures and most of the number are readable in state. From here with the increase in epoch count we can see number and their shapes getting increased progressively. We can also observe the decrease in the noise from one epoch to other epoch making the number more readable.

Generator:

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_58 (Dense)             (None, 256)               25856
_________________________________________________________________
leaky_re_lu_44 (LeakyReLU)   (None, 256)               0
_________________________________________________________________
dense_59 (Dense)             (None, 512)               131584
_________________________________________________________________
leaky_re_lu_45 (LeakyReLU)   (None, 512)               0
_________________________________________________________________
dense_60 (Dense)             (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_46 (LeakyReLU)   (None, 1024)              0
_________________________________________________________________
dense_61 (Dense)             (None, 784)               803600
=================================================================
Total params: 1,486,352
Trainable params: 1,486,352
Non-trainable params: 0
```

Discriminator:

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_58 (Dense)             (None, 256)               25856
_________________________________________________________________
leaky_re_lu_44 (LeakyReLU)   (None, 256)               0
_________________________________________________________________
dense_59 (Dense)             (None, 512)               131584
_________________________________________________________________
leaky_re_lu_45 (LeakyReLU)   (None, 512)               0
_________________________________________________________________
dense_60 (Dense)             (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_46 (LeakyReLU)   (None, 1024)              0
_________________________________________________________________
dense_61 (Dense)             (None, 784)               803600
=================================================================
Total params: 1,486,352
Trainable params: 1,486,352
Non-trainable params: 0
```

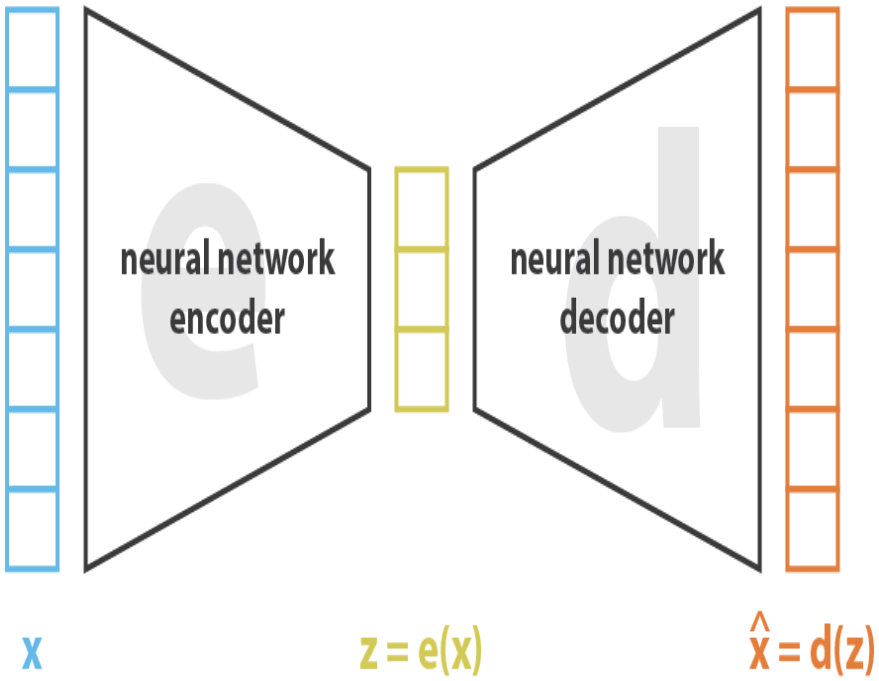
```
Epochs: 200
Batch size: 128
Batches per epoch: 468

--------------- Epoch 1 ---------------
```

Variation Auto Encoder (VAE):

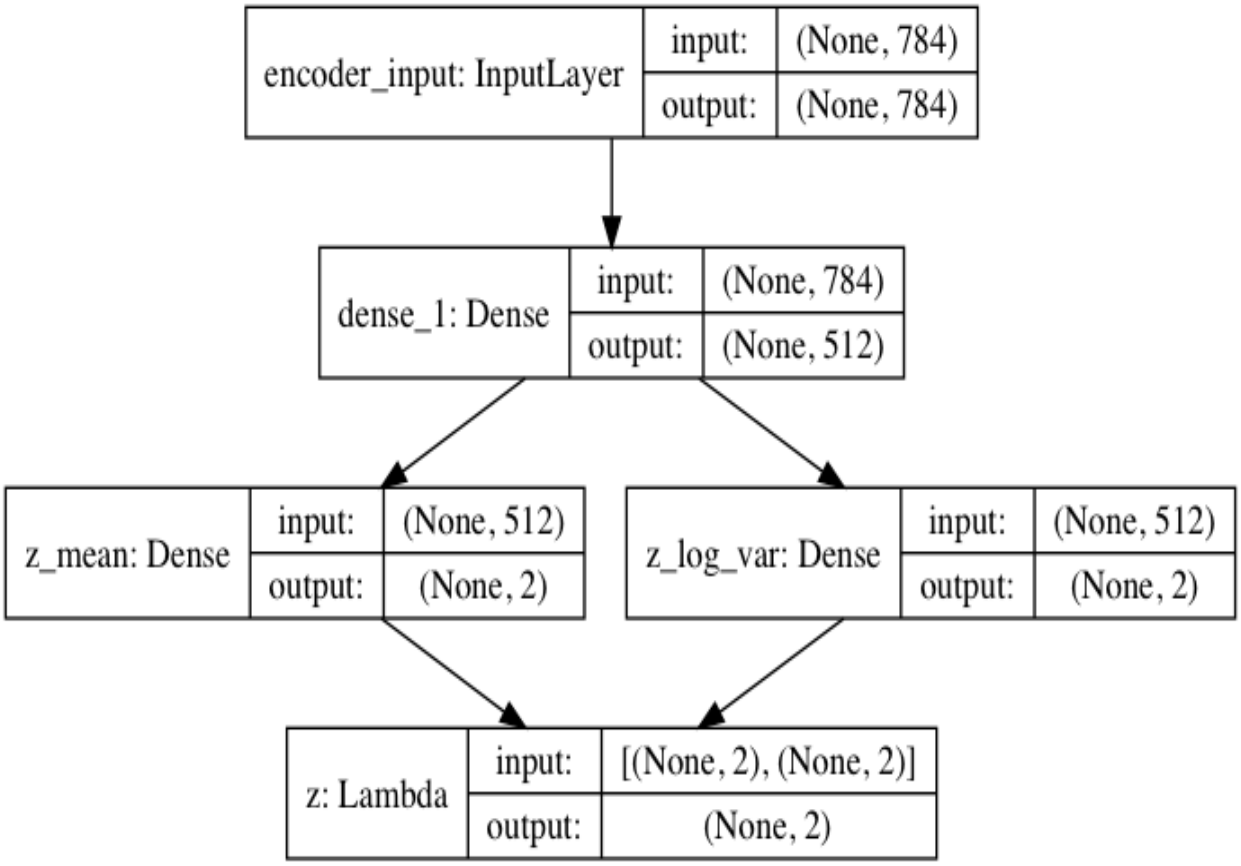
An autoencoder is a type of artificial neural network used to learn efficient data coding in an unsupervised manner. The aim of an autoencoder is to learn a representation for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”.

Variation auto encoders are generative models that estimates the Probability Density Function (PDF) of the given training data. Training this model on a natural looking image assign a high probability value to an image of a digit. Similarly, an image that is randomly generated on the other hand gets assigned with a low probability value. VAEs are directed probabilistic graphical models (DPGM) whose posterior is approximated by a neural network, forming an autoencoder-like architecture. It is different to the discriminative model which aims to learn the predictor given by the observation, generates modelling tries to simulate how the data are generated, I order to understand the underlying relations. These relations have the greatest potential of being generalized.

provides a probabilistic manner for describing an observation in latent space. Thus, rather than building an encoder which outputs a single value to describe each latent state attribute, we will formulate our encoder to describe a probability distribution for each latent attribute.



$$\text{loss} = \|x - \hat{x}\|^2 = \|x - d(z)\|^2 = \|x - d(e(x))\|^2$$



Model Overview:

Encoder:

The encoder is there to make sure it really creates values following a Normal Distribution. The returned values that will be fed to the decoder are the *z*-values. We will need the mean and standard deviation of our distributions later, when computing losses.

```
Model: "encoder"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
encoder_input (InputLayer)      (None, 784)          0
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 512)          401920      encoder_input[0][0]
__________________________________________________________________________________________________
z_mean (Dense)                  (None, 2)            1026        dense_1[0][0]
__________________________________________________________________________________________________
z_log_var (Dense)               (None, 2)            1026        dense_1[0][0]
__________________________________________________________________________________________________
z (Lambda)                      (None, 2)            0           z_mean[0][0]
                                                                 z_log_var[0][0]
==================================================================================================
Total params: 403,972
Trainable params: 403,972
Non-trainable params: 0
__________________________________________________________________________________________________
```

Decoder:

The decoder does not care about whether the input values are sampled from some specific distribution that has been defined by us. It simply will try to reconstruct the input images.

```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      (None, 2)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1536
_________________________________________________________________
dense_3 (Dense)              (None, 784)               402192
=================================================================
Total params: 403,728
Trainable params: 403,728
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "vae_mlp"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 784)               0
_________________________________________________________________
encoder (Model)              [(None, 2), (None, 2), (N 403972
_________________________________________________________________
decoder (Model)              (None, 784)               403728
=================================================================
Total params: 807,700
Trainable params: 807,700
Non-trainable params: 0
_________________________________________________________________
```

Comparing the output:

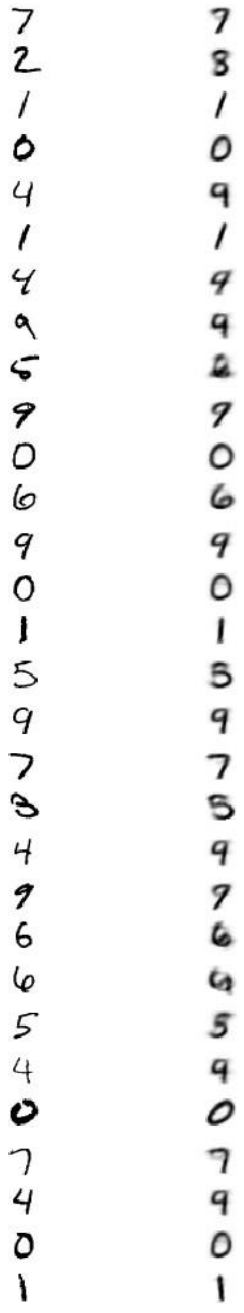
Here the output shows us the progressive improvement in the prediction of numbers from the input dataset. At epoch 1 we can observe there more wrong predictions compared to next epochs. This is a continued improvement in the performance. The predicted numbers of the original numbers also improved in shapes and reduced in noise with increase in the epochs.

Results:

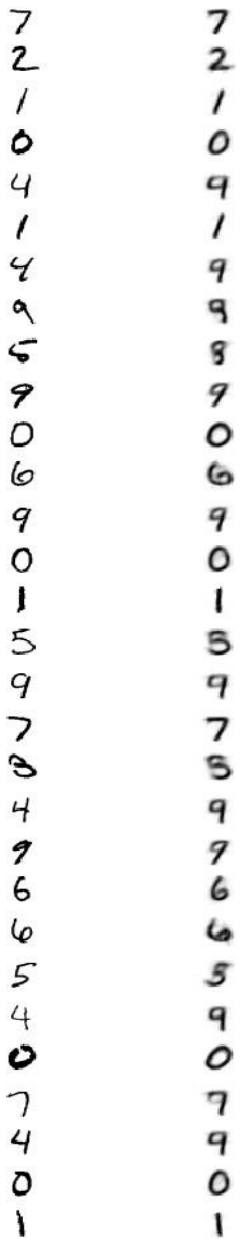
The results obtained at different epochs are like,

Epoch 5:

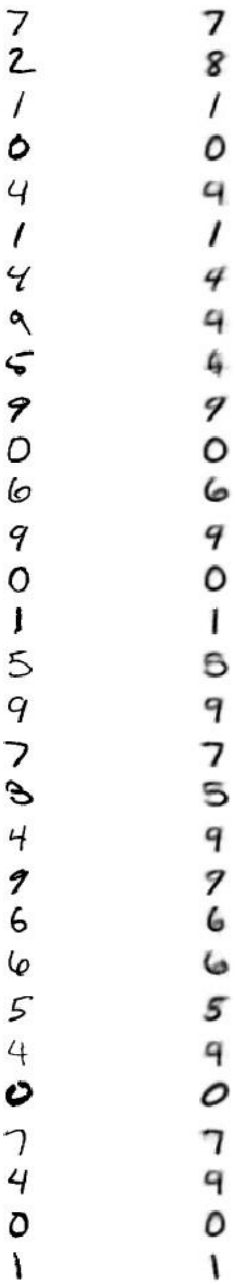
Epoch 25:



Epoch 50:



Epoch 100:



Project code:

- https://github.com/kroosen/GAN-in-keras-on-mnist/blob/master/GAN-keras-mnist-MLP.ipynb
- https://github.com/piyush-kgp/VAE-MNIST-Keras