Autoencoder: MNIST Dataset

Anup Patil

[anupsanjaypatil@gmail.com](mailto:anupsanjaypatil@gmail.com)

***Abstract-* 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 representation (encoding) for a set of data, by training the network to ignore signal “noise”. Along with the reduction side, a reconstructing side is learnt, where the Autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name*.***

I. INTRODUCTION

Autoencoders are a class of Unsupervised Networks that consist of two major networks: Encoders and Decoders. An Unsupervised Network is a network that learns patterns from data without any training labels. The Encoder generally uses a series of Dense and/or Convolutional layers to encode an image into a fixed length vector that represents the image a compact form, while the Decoder uses Dense and/or Convolutional layers to convert the latent representation vector back into that same image or another modified image.

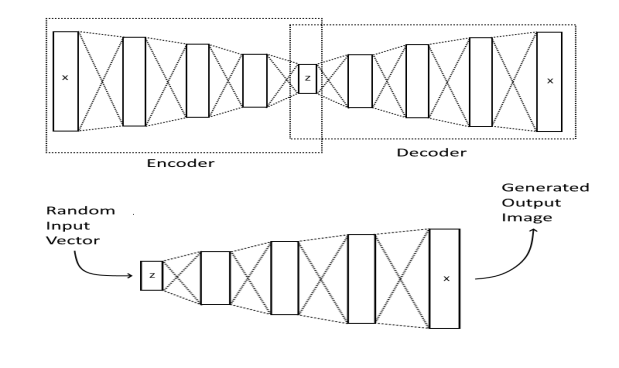


Figure 1: Basic Autoencoder architecture

The image above shows an example of a simple Autoencoder. In this Autoencoder, you can see that the input of size X is compressed into a latent vector of size Z and then decompressed into the same image of size X. To generate an image, a random input vector is given to the Decoder network. The Decoder network will convert the input vector into a full image.

II. ARCHITECTURE OF AUTOENCODER

We'll start off by building a simple Autoencoder to compress the MNIST dataset. With Autoencoder, we pass input data through an encoder that makes a compressed representation of the input. Then, this representation is passed through a decoder to reconstruct the input data. Generally the encoder and decoder will be built with neural networks and then trained on example data.

Here I have build a simple network architecture for the encoder and decoder. Then I will train an Autoencoder with these images by flattening them into 784 length vectors. The images from this dataset are already normalized such that the values are between 0 and 1.

In building simplest Autoencoder with two **hidden layer**. First we build the weights and biases for the hidden layers and output layers. Then by using corresponding weights and biases we build hidden layers and output layers. This hidden layer will be used as the compressed representation. Then, the encoder is the input layer and the hidden layer. The decoder is the hidden layer and the output layer. Since the images are normalized between 0 and 1, we need to use a **sigmoid activation on the output layer** to get values matching the input.

III. TRAINING

Here I'll write a bit of code to train the network. I'll just monitor the training loss. We initialize the variables with tf.global\_variables\_initializer(). In training Latent size is the size of the latent space: the vector holding the information after compression. This value is a crucial hyper parameter. If this value is too small, there won’t be enough data for reconstruction and if the value is too large, over fitting can occur. It is taken as the inout dim or encoded dim value in program. I found that a nice, successful latent size was 32 values long.

To train a model, you must compile it. To compile a model, you have to choose an optimizer and a loss function. Then, run the optimize and get reduced loss. For the optimizer, I chose Adam optimizer. For the loss, I chose binary cross-entropy. Binary Cross-Entropy i.e. tf.nn.sigmoid\_cross\_entropy\_with\_logits() is very commonly used with Autoencoders. In training model we use the value epochs is a hyper parameter set to 100. Generally, the more epochs the better, at least until the model plateaus out.

IV. RESULTS

After training for 100 epochs, below I've plotted f the test images along with their reconstructions

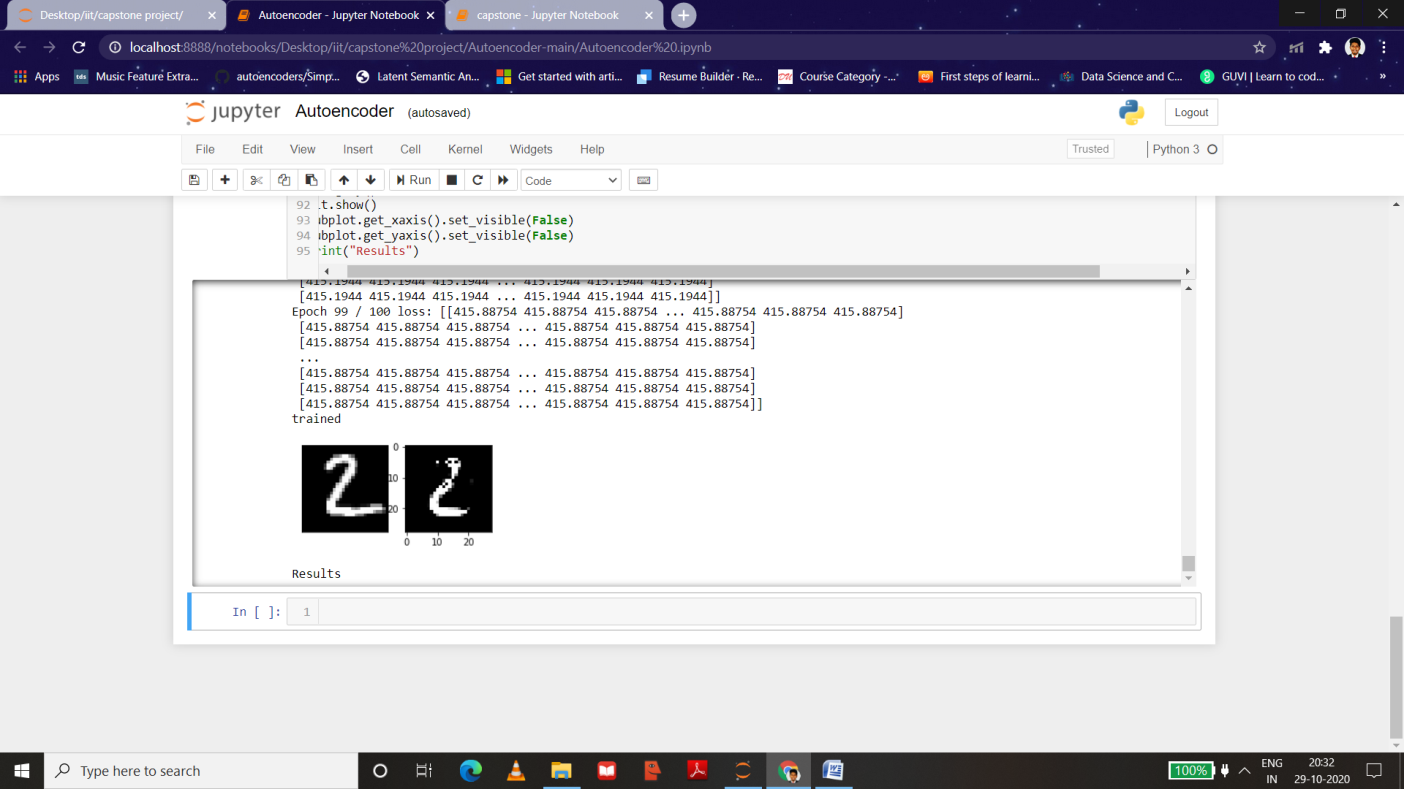


Figure 2: Result

As you can see, the results are pretty good. The Autoencoder successfully encodes and decodes the latent space vectors with pretty good quality