Gurjus Singh

November 15th, 2020

MSDS 422 – Practical Machine Learning

Assignment #9 Auto Encoder

**Data preparation, exploration, visualization**

For the Data Analysis this week, I again dealt with the MNIST dataset from Week 5 and Week 6. The MNIST data is a dataset where each row has 784 pixels/features of numbers 0-9. What my goal for this Data Analysis was to use MNIST data and do **Dimensionality Reduction similar to PCA, but with Autoencoders [1]. This is a key unsupervised learning method, which means it does not use a Target/Label to train [1]. The key difference between PCA and Autoencoders is that PCA deals with Linear Combinations when extracting features while Autoencoders deal with nonlinear combinations [1]. I used Variational Autoencoders which not only extracts features, but also makes a distribution of the images it sees and uses that to create a random sample [2].**

There was not much data prep that needed to be done. The first thing I did was imported the TensorFlow and Keras packages as before. I then checked the versions as Tensorflow 2.0 or above was required for this Analysis. I then defined variables for my layers so my model could define which later computed what statistics per layer or used what activation function. After defining explicitly, the layers I then defined a sampling function which was used to generate more data points, and then I defined the model creating the encoder, decoder and instantiating the VAE model.

After defining the model, I then loaded in the MNIST data splitting it into training and test sets after downloading. Next, I need to convert the data to float 32 and divide by 255 so it was in pixel type. I also looked the shape after each split after words to see the labels and if there were exactly 784 features. Output 1-1 showed that there were 60,000 rows with 784 columns and 60,000 labels as well in the train set. I had split up the data 60,000-10,000 as shown in 1-1.

**A picture containing text

Description automatically generated**

*Output 1-1*

I also saw if the dataset was balanced in 1-2 as I did in the last analysis, to see if the data set was balanced with all the numbers, I could see the training set was pretty balanced with the most being 1s. Same thing with the test data, although the 5’s did seem pretty low. This was good to know as well to see how our normal distribution would turn out using the Bivariate Standard Normal [2]. I also looked at the first 10 labels for training set.

Graphical user interface, application

Description automatically generated

*Output 1-2*

A close up of a keyboard

Description automatically generated

*Output 1-3*

In 1-3, I also saw what the images looked like when the pixels were stitched together. I did see some images that clearly looked the same as labels, while others looked different than their labels. After doing this it was time to train the models.

**Review research design and modeling methods**

For this assignment I focused, on using Variational Auto Encoders with Intermediate Size of 16 neurons and 64 neurons respectively. As mentioned in the data prep section above I split up the dataset into a 60,000-10,000 split. As also mentioned above, Auto Encoders are generally used for dimensionality reduction, but is also a neural network so it is two in one approach unlike PCA where you have to do dimensionality reduction first and then feed into a training/learning algorithm [2]. Another thing that Auto Encoder are especially specialized with is unlike PCA they are nonlinear when extracting features [1]. Autoencoders also utilizes the sigmoid function which is a function that varies between 0 to 1 because I need to figure out a probability for each of random numbers to show where they would be in the Monte Carlo simulation/histogram of the distribution and which number they would correspond to after doing the random sampling [2].

For the loss function of the Variational Auto Encoder there is a reconstruction loss which is used to make sure Auto Encoder “reconstructs the inputs” [3]. Another part of the loss function is the latent loss to make sure the Autoencoder is sampling for a Gaussian Normal Distribution [3]. We also take into account the regularization loss as well [2]. For this analysis these first two losses also needed to be custom made and I judged my models based on the loss because there was no accuracy score as this was unsupervised.

**Review Results and Evaluate Model**

After fitting the two models for the first model I got a loss around 144 as seen in 1-4, while my loss for model 2 was around 155 as seen in 1-5. This showed that Model 1 was performing better with 64 intermediate neurons. I also looked at the latent space which helped to generate the images and how the features were extracted in 1-6, 1-7. After looking at the latent spaces, I tried to see how the distribution was on train set and test set for both models in 1-8 through 1-11 making a bivariate plot.

Table

Description automatically generated

*Model 1 64 neurons 1-4*

*Table

Description automatically generated*

*Model 2 16 neurons 1-5*

*A close up of a computer

Description automatically generated*

*Model 1 Images 1-6*

A close up of a computer

Description automatically generated

Chart, scatter chart

Description automatically generated*Model 2 Images 1-7*

Chart, scatter chart

Description automatically generated

*Model 1 Bivar Plot Train Data 1-8 Model 1 Bivar Plot Test Data 1-9*

*Chart, histogram, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated*

*Model 2 Train Data Plot 1-10 Model 2 Test Data Plot 1-11*

For model 1 it identified the majority of labels in the Gaussian distribution were associated with 1s and 7s for the Train Data as seen in 1-8, while for model 2 it said exactly the same thing for Train Data in 1-10. As you can see for the Test Data in both models it had the same type of shape as the Test Data. The shape is basically is the beast form of compression of the data that was chosen by the autoencoder in the intermediate layer when figuring out the latent space, feature extraction, and dimensional reduction [2].

**Implementation and Programming**

In this analysis the main packages that were used were the **Tensorflow and Keras packages** for using the Autoencoders and plotting the Bivariate plots mentioned in the last section. I also utilized **numpy and matplotlib**. All the packages imported are seen below in 1-12. I first checked the packages by using attribute **tf.\_\_version\_\_** and I saw that my versions for Tensorflow was 2.3.0 and I used the same type of syntax for Keras which spit out 2.4.0. I then created variables for the dimensions for each layer **28\*28** for input layer which signified the original 784 features. The intermediate\_dim layer was 16, and 64 for both models. The Latent Dimensions were 2. I then created layers using **keras.Input, and layers.Dense** which would help compute mean and standard dev when instantiating and running the models. I then create a sampling function which would be used in the model when sampling at random to generate numbers for Gaussian Distribution based on MNIST data.

Before running the model I created the encoder, decoder which was used by using the syntax **keras.Model()** and using the inputs. I also created the latent inputs using the 64 neurons mentioned above and using **layers.Dense()** and I also used this for the outputs as well which transformed the extracted features back in 784 features which was used to generate more data.

I then instantiated the model using the same syntax above **keras.Model()** These objects were all in the **Keras and Tensorflow** packages. I then saw a summary of the model using **summary** function and using **keras.utils.plot\_model** to save a diagram of the model. These were also from the **Keras and Tensorflow** packages. I then downloaded the MNIST dataset using **Tensorflow’s load\_data** function. I then created custom loss as the loss was very different from losses I have been using in other analyses. I particularly used Tensorflow functions such as **K.mean(), K.sum(), K.square, and add\_loss()** for computing these losses as seen in 1-13 below.

After I then feature engineering by converting the features to float 32, and divided them by 255 and I took the product to reshape then using **np.product** from **numpy package.** Then after that I wanted to see the shape of the data, which using **.shape attribute.** I also saw most initial 10 labels of data by using index slicing **[0:10].** I also wanted to see most common labels using **Counter(),most\_common().** Lastly I saw the original images by using **imshow** from **matplotlib package** and saw the images by plotting their pixels on a subplots of which I saw 50 images. After seeing the initial data, I then used **.fit() function**. to train data. I used loss to compare both models. I then used **matplotlib and imshow()** function and used nested **for loops** to plot latent space. I made custom function which used **.predict** function from **Tensorflow** package to make a bivariate plot and used .**scatter from matplotlib package** to plot the data points taken from predict function. The .predict function had three columns which included the latent features which I plotted from predict function.

Graphical user interface, text, application, email

Description automatically generated

*Import Packages 1-12*

*Graphical user interface, text, application

Description automatically generated*

*Loss computation 1-13*

**Exposition, problem description, Management recommendation**

The goal as mentioned before was to use an unsupervised method in Auto Encoders on the MNIST Dataset. My results suggested by looking at 1-4 and 1-5 that Model 1 with 64 neurons in intermediate layer was performing better with a loss of 144. Model 2 with 16 neurons had a loss of 155. **Therefore, I recommend to management to choose Model 1 in 1-4 with 64 neurons in the Intermediate Layer and represented below in 1-14.** In the future I hope to try to find a way to lower the loss function even more as I felt it was still too high for my liking. Maybe getting it down to 100 or under 100 would be nicer. I do realize that I have to be cautious in taking this approach as if I significantly reduce the loss it would cause overfitting problems [1]. I also was satisfied with the fitting as it was not as time consuming as my other analysis. I also saw some beautiful bivariate charts as seen above which I like and hope to analyze them more in depth in the future.

Text, letter

Description automatically generated

*Model 1 – 1-14*

References

[1] Srinivasan, S. (2020). *Week 9 Lecture Auto Encoders* [Slides]. Canvas.

https://canvas.northwestern.edu/courses/125893/files/9374628/download?wrap=1

[2] Srinivasan, S. (2020b, November 10). *Sync Session Auto Encoder* [Slides]. Canvas.

<https://canvas.northwestern.edu/courses/125893/files/9968055?module_item_id=1703375>

[3] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed.). O’Reilly Media.