Self-Work Component Project Report

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING (ECE325)

Topic : To display deep learning model on Pima Indians onset of diabetes binary classification problem

Submitted to :-

Dr Rinki Gupta

Submitted by :-

Anurag Tripathi (A12405117001)

Department of Electronic and Communication Engineering

Amity School on Engineering and Technology (ASET)

AUUP, Noida

TABLE OF CONTENTS

Aim :- **To display deep learning model on Pima Indians onset of diabetes binary classification problem**.

Theory :-

Artificial Neural Network

The artificial neural network (ANN) is made up of multiple nodes in which a transfer function is built in. There are three types of nodes such as input node which is consisting of information in numerical form. This information is then passed throughout the network also called as hidden node. This is the second type of node. Finally the information is reached to the output node. This is the third type of node.

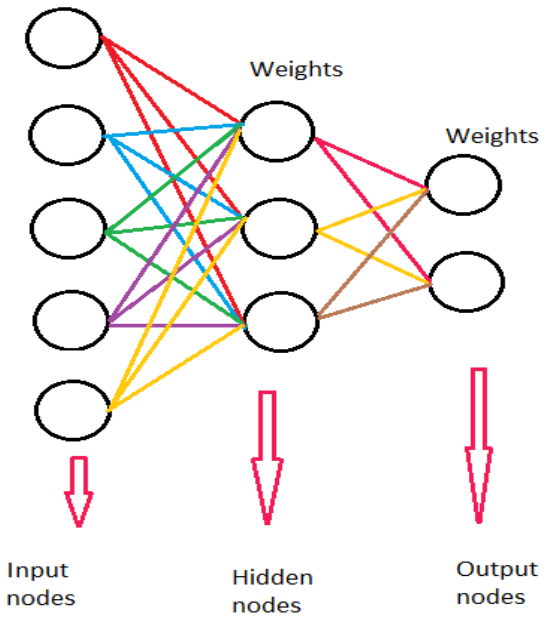

Fig.: Three layers of nodes

These nodes in ANNs can be compared with biological neurons of human brain.

All the calculations take place in the hidden layer and its role to convert the input data into meaningful outputs that can be utilized for various applications. The relationship between each layer is established through the weights that are randomly assigned to each interconnection between the layers. Each input from the node is multiplied with the weights and the weights are summed up to form the transfer function. This further is passed to the activation function present in the nodes of the hidden layers. The classified output is received at the output neuron. Artificial neurons and neurons associated with them as a rule have a randomly initiated weight that changes itself as learning advances. The weight either increments or reduces the quality of the signal at a node. The performance of the neural network is evaluated by the fraction of correct predictions over total predictions. The equation is given as follows:-

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} * 100$$

After classification, the accuracies at each epoch (training iteration) are plotted, which gives the learning curve of the neural network. Ideally, the accuracies should be low at low epochs, but should increase as epochs increase, and should finally get constant after peak value is obtained.

Program Implementation:

We are going to use the Pima Indians onset of diabetes dataset. This is a standard machine learning dataset from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years.

As such, it is a binary classification problem (onset of diabetes as 1 or not as 0). All of the input variables that describe each patient are numerical. This makes it easy to use directly with neural networks that expect numerical input and output values, and ideal for our first neural network in Keras. Keras is an open-source neural-network library written in Python.

https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv

Take a look inside the file, we see rows of data like the following:

Fig : Snap of dataset

Dataset Details :-

https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.names

- Number of Instances: 768

- Number of Attributes: 8 plus class
- For Each Attribute: (all numeric-valued)
  1. Number of times pregnant
  2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
  3. Diastolic blood pressure (mm Hg)
  4. Triceps skin fold thickness (mm)
  5. 2-Hour serum insulin (mu U/ml)
  6. Body mass index (weight in kg/(height in m)^2)
  7. Diabetes pedigree function
  8. Age (years)
  9. Class variable (0 or 1)

- Class Distribution: (class value 1 is interpreted as "tested positive for
diabetes")

1. Load Data:-

We can now load our dataset.

```
# first neural network with keras tutorial
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
...
```

Fig : Loading Data

We can now load the file as a matrix of numbers using the NumPy function loadtxt().
There are eight input variables and one output variable (the last column). We will be learning a model to map rows of input variables (X) to an output variable (y), which we often summarize as *y = f(X)*.
The variables can be summarized as follows:

Input Variables (X):

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)

Output Variables (Y):

1. Class variable (0 or 1)

Once the CSV file is loaded into memory, we can split the columns of data into input and output variables.

The data will be stored in a 2D array where the first dimension is rows and the second dimension is columns, e.g. [rows, columns].

We can split the array into two arrays by selecting subsets of columns using the standard NumPy slice operator or ":" We can select the first 8 columns from index 0 to index 7 via the slice 0:8. We can then select the output column (the 9th variable) via index 8.

```
...
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
...
```

Fig :-

Note, the dataset has 9 columns and the range 0:8 will select columns from 0 to 7, stopping before index 8.
We are now ready to define our neural network model.

2. Define Keras Model :-

Models in Keras are defined as a sequence of layers. We creates a *Sequential model* and add layers one at a time until we are happy with our network architecture.

The first thing to get right is to ensure the input layer has the right number of input features. This can be specified when creating the first layer with the **input_dim** argument and setting it to 8 for the 8 input variables.
We will use the rectified linear unit activation function referred to as ReLU on the first two layers and the Sigmoid function in the output layer.

Rectified Linear Unit (ReLU) Activation Function

ReLU is an activation function which is fast enough and has good performance. When the value of z is less than 0, the functions returns to 0. While when the value of z is greater than or equal to zero, it is equal to z. The formula can be depicted as:

$$R(z) = \max(0, z)$$

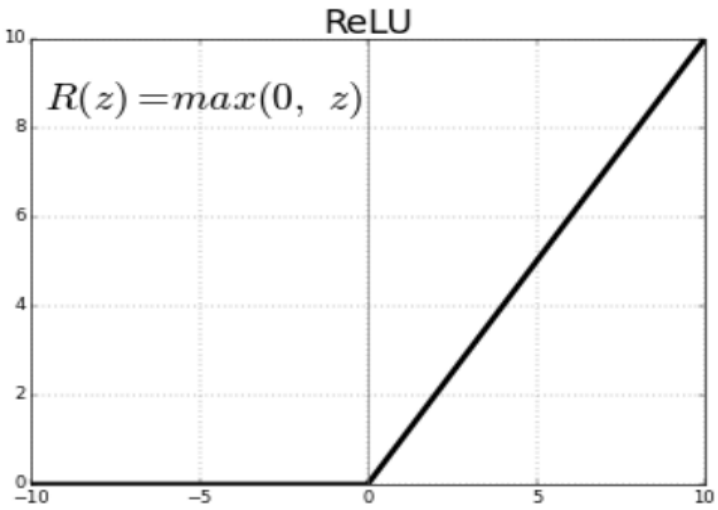

Fig.: Rectified Linear Unit Function

Sigmoid Activation Function

"S"- shaped curve also known as sigmoid curve. It lies from 0 to 1. Because of this range, it can be used in neural networks models where the probability can be predicted from the output.

$$S(x) = \frac{1}{1+e^{-x}}$$

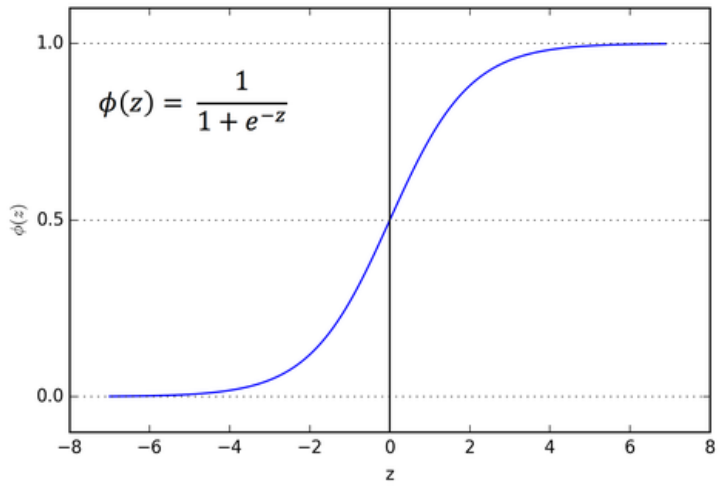

Fig.: Sigmoid Function

Now,

We can piece it all together by adding each layer:

- The model expects rows of data with 8 variables (the *input_dim=8* argument)
- The first hidden layer has 12 nodes and uses the relu activation function.
- The second hidden layer has 8 nodes and uses the relu activation function.
- The output layer has one node and uses the sigmoid activation function.

```
...
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
...
```

Fig :- Defining Keras Model

3. Compile Keras Model :-

Now that the model is defined, *we can compile it.*

Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on our hardware, such as CPU or GPU.

When compiling, we must specify some additional properties required when training the network.

We must specify the loss function to use to evaluate a set of weights. In this case, we will use cross entropy as the loss argument. This loss is for a binary classification problems and is defined in Keras as “binary crossentropy“.

We will define the optimizer as the efficient stochastic gradient descent algorithm “adam“. It automatically tunes itself and gives good results in a wide range of problems. Finally, because it is a classification problem, we will collect and report the classification accuracy, defined via the metrics argument.

```
...
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
...
```

Fig :- Compliling the keras model

4. Fit Keras Model :-

We have defined our model and compiled it ready for efficient computation.

Now it is time to execute the model on some data.

We can train or fit our model on our loaded data by calling the **fit()** function on the model.
Training occurs over epochs and each epoch is split into batches.

- **Epoch** :- One pass through all of the rows in the training dataset.
- **Batch** :- One or more samples considered by the model within an epoch before weights are updated.

The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the epochs argument.
For this problem, we will run for a small number of epochs (150) and use a relatively small batch size of 10.

These configurations can be chosen experimentally by trial and error. We want to train the model enough so that it learns a good (or good enough) mapping of rows of input data to the output classification.

The model will always have some error, but the amount of error will level out after some point for a given model configuration.

```
...
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
...
```

Fig :- Fitting the model on the dataset

5. Evaluate Keras Model :-

We have trained our neural network on the entire dataset and we can evaluate the performance of the network on the same dataset.

This will only give us an idea of how well we have modeled the dataset (e.g. train accuracy), but no idea of how well the algorithm might perform on new data.

We can evaluate our model on our training dataset using the **evaluate()** function on our model. This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics we have configured, such as accuracy.

The **evaluate()** function will return a list with two values. The first will be the loss of the model on the dataset and the second will be the accuracy of the model on the dataset.

6.Visualize Model Training History in Keras :-

One of the default callbacks that is registered when training all deep learning models is the History callback . It records training metrics for each epoch .

This includes the loss and the accuracy (for classification problems) as well as the loss and accuracy for the validation dataset, if one is set.

We can use the data collected in the history object to create plots.

The plots can provide an indication of useful things about the training of the model, such as:

- It's speed of convergence over epochs (slope).
- Whether the model may have already converged.
- Whether the mode may be over-learning the training data (inflection for validation line).

7. Tie It All Together :-

Let’s tie it all together into a complete code example .

```
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import numpy
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(4, kernel_initializer='uniform', activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10, verbose=0)
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Fig :- Complete code

8. Result:

The plots are provided below.

From the plot of accuracy we can see that the model could probably be trained a little more as the trend for accuracy on both datasets is still rising for the last few epochs.

We can also see that the model has not yet over-learned the training dataset, showing comparable skill on both datasets.

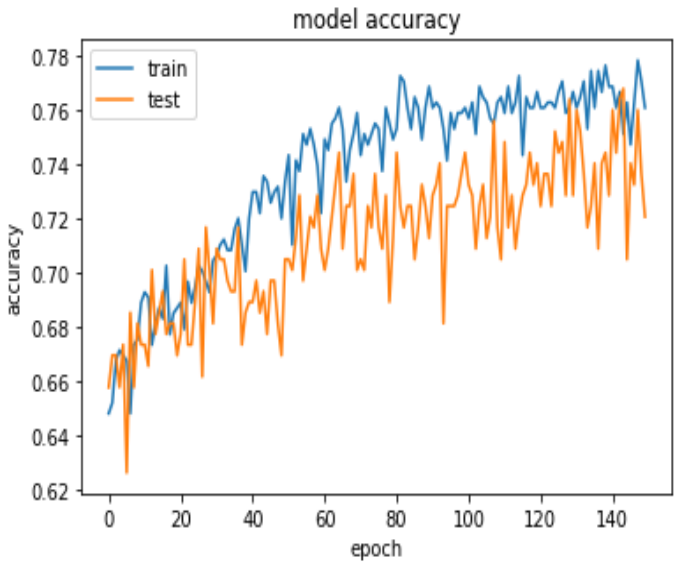

Fig :- model accuracy

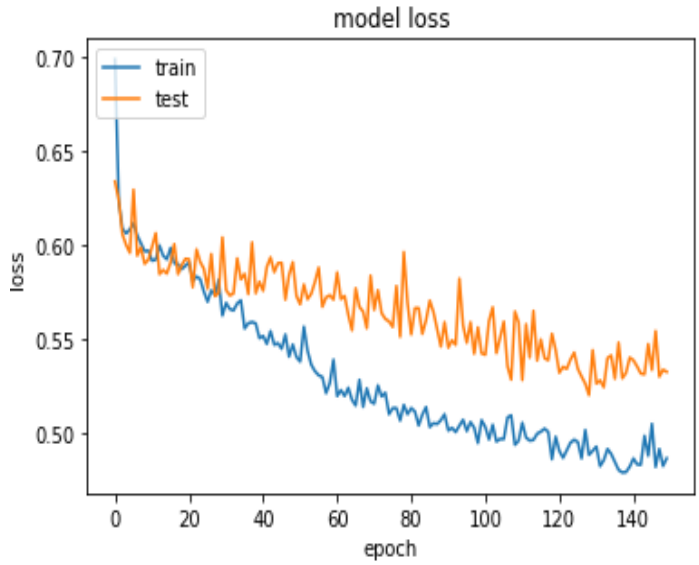

Fig :- model loss

From the plot of loss, we can see that the model has comparable performance on both train and validation datasets (labeled test). If these parallel plots start to depart consistently, it might be a sign to stop training at an earlier epoch.

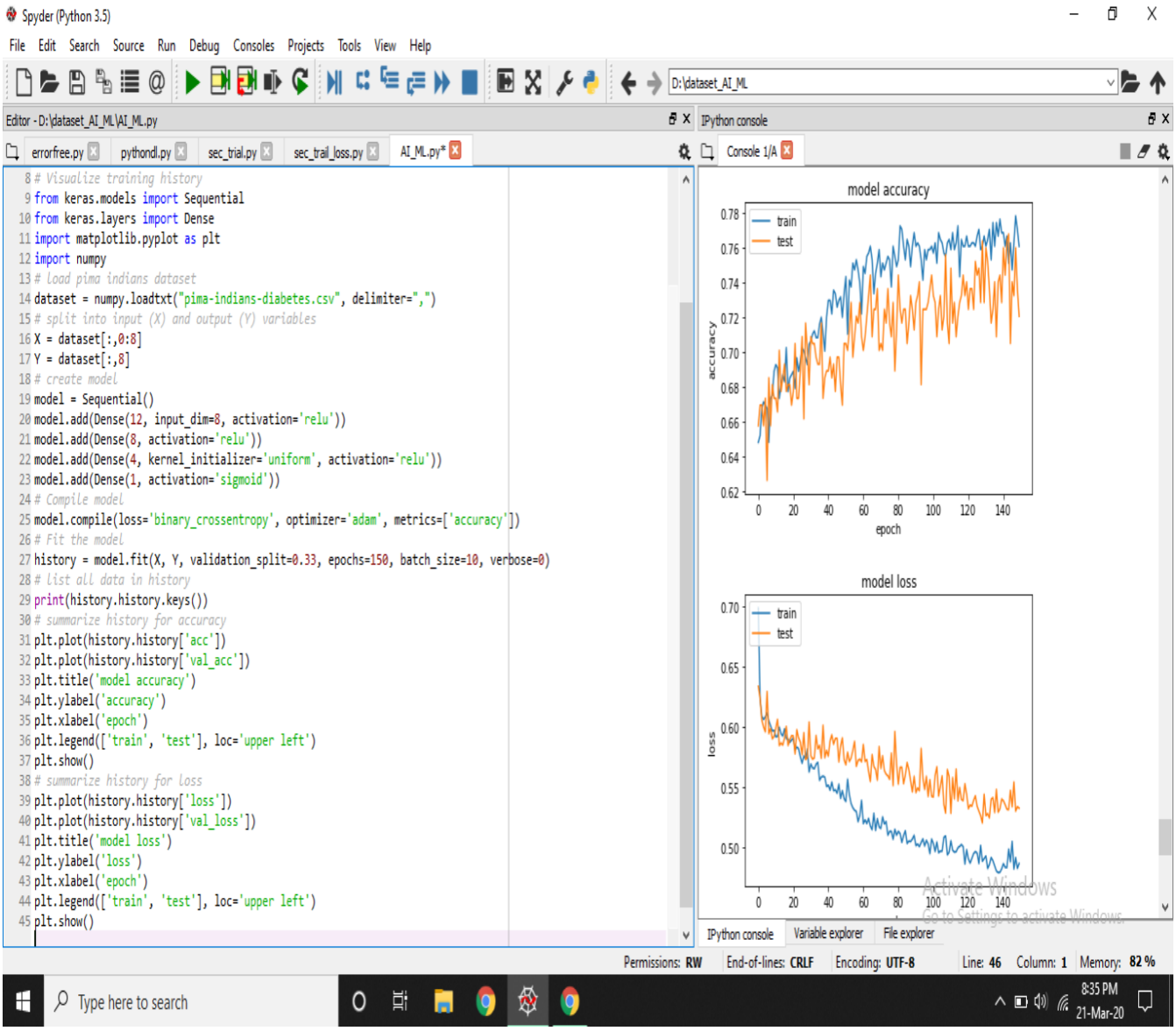

Fig :- Complete snapshot

Referance:-

1. Reduction of Overfitting in Diabetes Prediction Using Deep Learning Neural Network Akm Ashiquzzaman , Abdul Kawsar Tushar , Md. Rashedul Islam and Jong-Myon Kim , Department of CSE, University of Asia Pacific, Dhaka, Bangladesh ,Department of Electrical, Electronics, and Computer Engineering, University of Ulsan, Ulsan, Republic of Korea, 2017
2. National Institute of Diabetes and Digestive and Kidney Diseases, https://www.niddk.nih.gov/, last accessed: 2017/05/29. 18. Theano Development Team, Theano: A Python framework for fast computation of mathematical expressions. In: arXiv e-prints, vol. abs/1605.02688 (2016).
3. 19. F. Chollet, Keras, https://github.com/fchollet/keras, last accessed 2017/06/01
4. Dasgupta, J., Sikder, J., and Mandal, D.: Modeling and Optimization of Polymer Enhanced Ultrafiltration Using Hybrid Neuralgenetic Algorithm Based Evolutionary Approach, Applied Soft Computing, vol. 55, pp. 108–126 (2017).
5. Nielsen, M. A.: Neural Networks and Deep Learning http://neuralnetworksanddeeplearning.com, last accessed 2017/05/29
6. Predicting Diabetes in Medical Datasets Using Machine Learning Techniques Uswa Ali Zia, Dr. Naeem Khan, 5 MAY 2017
7.]A. Iyer, J. S and R. Sumbaly, "Diagnosis of Diabetes Using Classification Mining Techniques", IJDKP, vol. 5, no. 1, pp. 01-14, 2015.
8. N. M. S. kumar, T. Eswari, P. Sampath, and S. Lavanya, "Predictive methodology for diabetic dataanalysis in big data," Procedia Computer Science, vol. 50, pp. 203–208, 2015
9. S.Salian and G. Harisekaran, “Big Data Analytics Predicting Risk of Readmissions of Diabetic Patients,” International Journal of Science and Research, vol. 4, April 2015.