Capstone Project Report
June 2018

Time Series Classification

There are a lot of Time-Series available for classification. I have chosen EEG signals for classification as I'm interested in the use of machine learning in the medical area.

EEG signal Based Eye State Classification

Definition

1 Domain Background

The electroencephalogram (EEG) is a bio-signal just like MRI, CT, f-MRI, ECG. It is the recording of the electrical activity of the brain from the scalp. Whenever we do some activities neurons are fired in the brain which generates some current, we use electrodes on the scalp and measure that very tiny current and then amplify it. In various researches[1], it is found that our brain generates different EEG time-series patterns for different activities like moving hands, legs, eyes opening and closing etc. So if we use machine learning to recognize those patterns then we can easily classify them.

2 Problem Statement

Given EEG pattern(Time - Series), we have to classify person's eyes are open or closed using supervised machine learning approach.

3 Solution Statement

- **Statistical Feature Extraction -** Extract statistical feature like - mean, standard-deviation, Kurt, Skewness, Band Power etc and train it classification algorithm - SVM, Logistic Regression. So when a new EEG signal comes, we will first extract statistical features and then send it to classification algorithm and predict the output.

- **Using Multilayer Perceptron -** Use 14 values as feature vector and train it to the neural network to classify it.

- **Recurrent Neural Network -** Train LSTM(Long Short Term Memory) network on eeg signal and use last layer as sigmoid to predict a value of '0' and '1'.

4 Performance Metrics

For the classification task accuracy is not enough.

Example – Let's say you have 100 data point out of which 90 are of label 0 and 10 are of label 1.
If we just create a naive classifier that gives only 0 as an output for every data point then,

Accuracy = 90/100 = 0.9

But it is totally useless model; this is called as **accuracy paradox**.
This is where Precision and Recall comes in.

Precision = $\frac{TP}{TP + FP}$

Recall = $\frac{TP}{TP+FN}$

F1 Score = 2 $*$ $\frac{Precision * recall}{precision+recall}$

Where,

- **True Positives (TP):** number of positive examples, labelled as such.
- **False Positives (FP):** number of negative examples, labelled as positive.
- **True Negatives (TN):** number of negative examples, labelled as such.
- **False Negatives (FN):** number of positive examples, labelled as negative.

Analysis -

1 Datasets and Inputs

Dataset - EEG Eye State Data Set (http://archive.ics.uci.edu/ml/datasets/EEG+Eye+State)

Each row consist of 14 EEG values (means 14 time-series values) representing 14 electrodes (TF7, O2, F3, P8, T8, F4, FC6, AF3, FC5, T7, F8, P7, O1 and AF4) along with a value indicating the eye state.

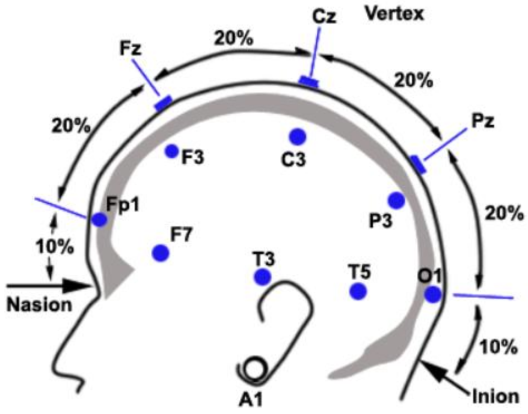


Figure: Electrodes Placement

Labels - '1' indicates eye-closed.
'0' indicates eye-open.

Input - 14 EEG values
Output - '1' or '0'
Missing Data – 0

2 Data exploration –

Sample Data –

4329.23, 4009.23, 4289.23, 4148.21, 4350.26, 4586.15, 4096.92, 4641.03, 4222.05, 4238.46, 4211.28, 4280.51, 4635.9, 4393.85, 0

As we can see first 14 values are EEG value and last 15th Value is label which is ‘0’ (eye open) here.

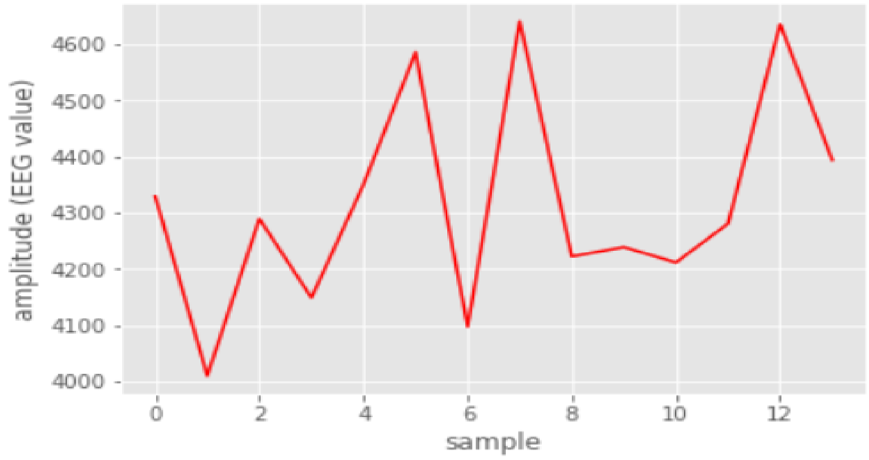
Statistical Information about data –

Eye	Open	Closed

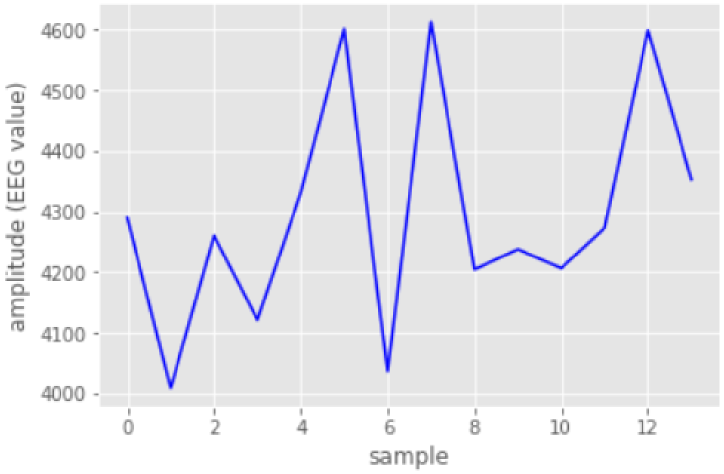
Electrode	Min	Mean	Std	Max	Min	Mean	Std	Max
AF3	4198	4305	33	4445	1030	4297	54	4504
F7	3905	4005	27	4138	3924	4013	52	7804
F3	4212	4265	20	4367	4197	4263	27	5762
FC5	4058	4121	20	4214	2453	4123	27	4250
T7	4309	4341	18	4435	2089	4341	29	4463
P7	4574	4618	18	4708	2768	4620	28	4756
O1	4026	4073	24	4167	3581	4071	18	4178
O2	4567	4616	18	4695	4567	4615	34	7264
P8	4147	4202	18	4287	4152	4200	17	4586
T8	4174	4233	19	4323	4152	4229	33	6674
FC6	4130	4204	24	4319	4100	4200	27	5170
F4	4225	4281	18	4368	4201	4277	36	7002
F8	4510	4610	32	4811	86	4601	59	4833
AF4	4246	4367	34	4552	1366	4356	52	4573

3 Visualization Of Time Series –

Eye open series –



Eye closed series –



As from graph we can see in Eye open (mean value = 4316.59) state values are greater than Eye closed (mean value = 4302.08) state. **This observation helps us to build intuitions to extract statistical features from this time series.**

4 Algorithm and Techniques –

1) – Statistical Feature Extraction – As we know to compare two time-series data we can extract statistical features –

a) – Mean –

$$\overline{X} = \frac{\sum X}{N}$$

b) – Standard Deviation –

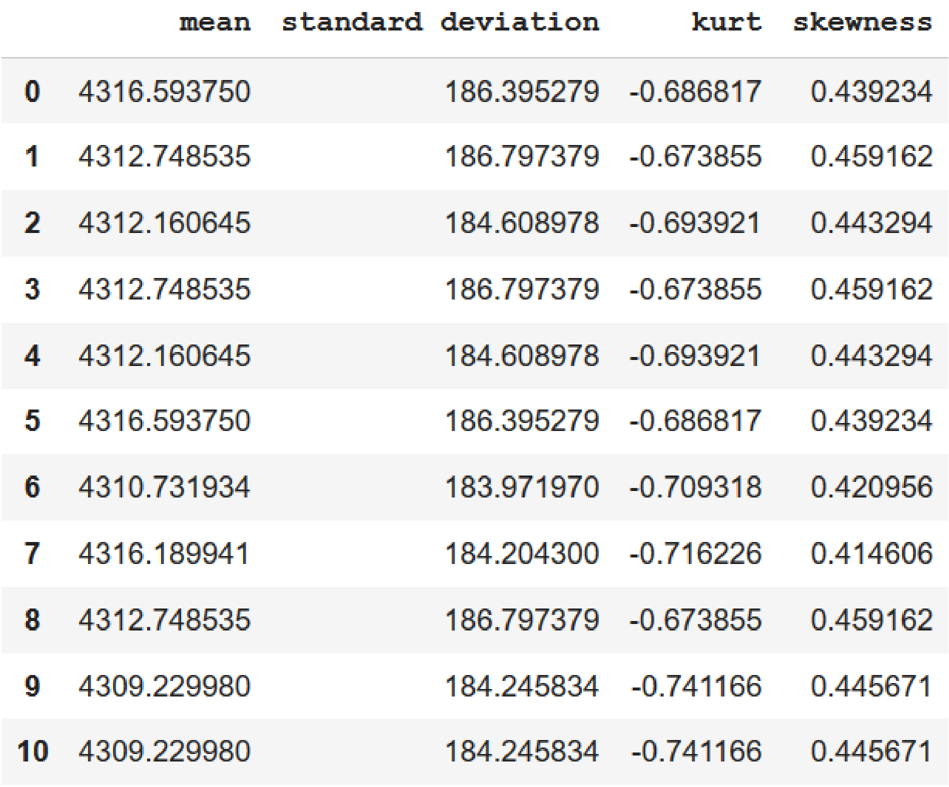
$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{n}}$$

c) – Skewness –

$$SKEW = \frac{\sum_{t=1}^{n}(y(t) - \mu)^3}{n\sigma^3}$$

d) – Kurtosis –

$$KURT = \frac{\sum_{t=1}^{n}(y(t) - \mu)^4}{n\sigma^4} - 3$$



	mean	standard deviation	kurt	skewness
0	4316.593750	186.395279	-0.686817	0.439234
1	4312.748535	186.797379	-0.673855	0.459162
2	4312.160645	184.608978	-0.693921	0.443294
3	4312.748535	186.797379	-0.673855	0.459162
4	4312.160645	184.608978	-0.693921	0.443294
5	4316.593750	186.395279	-0.686817	0.439234
6	4310.731934	183.971970	-0.709318	0.420956
7	4316.189941	184.204300	-0.716226	0.414606
8	4312.748535	186.797379	-0.673855	0.459162
9	4309.229980	184.245834	-0.741166	0.445671
10	4309.229980	184.245834	-0.741166	0.445671

Figure: Extracted Features from Time-Series

2) SVM - SVM maps the input data to high dimensional feature space through *kernels*, so that it can be separated by a hyperplane.

Justification - Because of different kernels SVM will provide good accuracy in our problem. In implementation I have used ‘rbf’ kernel which was giving better results than other kernels (poly, linear).

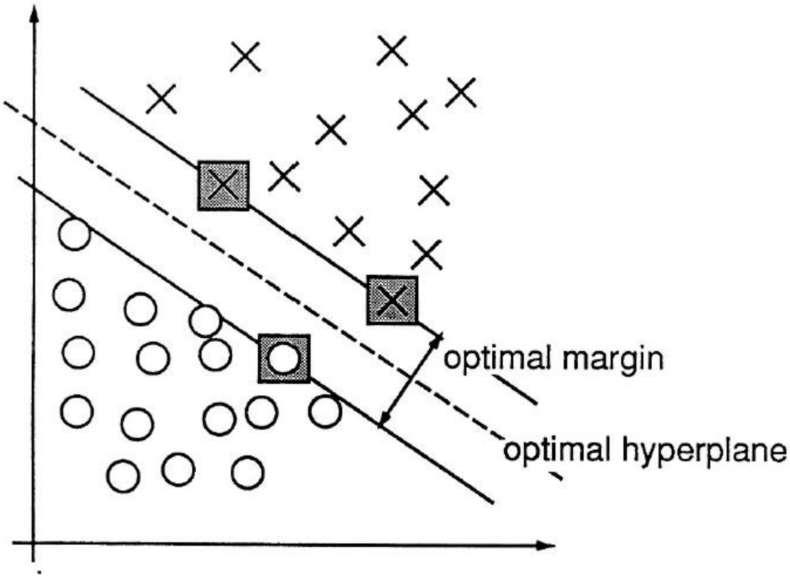
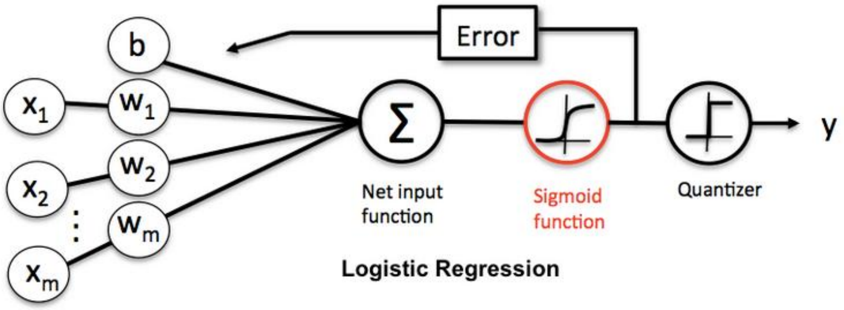


Figure: SVM

3) – Logistic Regression – Logistic Regression is another technique for classification, which use sigmoid for two class classification and uses cross-entropy function as a loss function.
Justification - Logistic Regression are used as benchmark models because of simple algorithm with non-linear sigmoid function they provide good baseline accuracy.



4) – Neural Networks - Artificial Neural Networks are inspired from our Brain Neural Networks. ANN randomly assigns weights to all layers then use backpropagation for decreasing the error rate and updating the weights.
Justification - Deep Neural Networks have specialty of function approximation on lot of different type of data. Best part of neural network is that at each layer automatically important features get extract, that’s what make neural network great in field of Machine Learning.

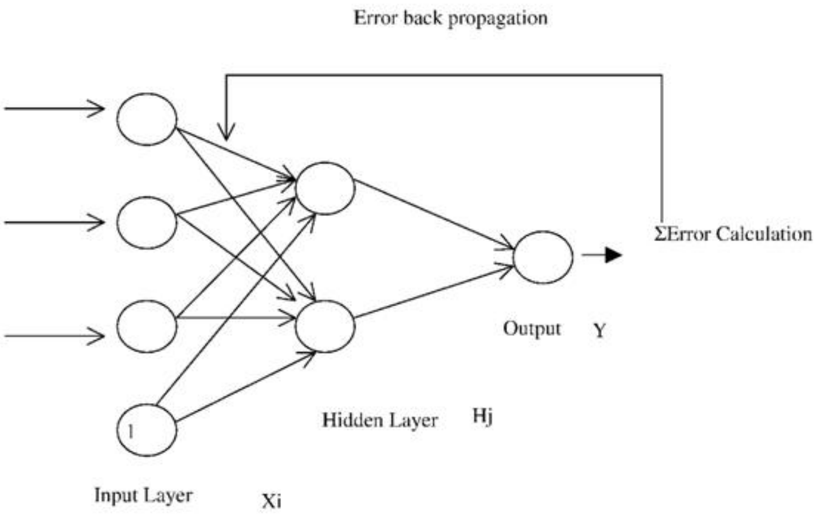
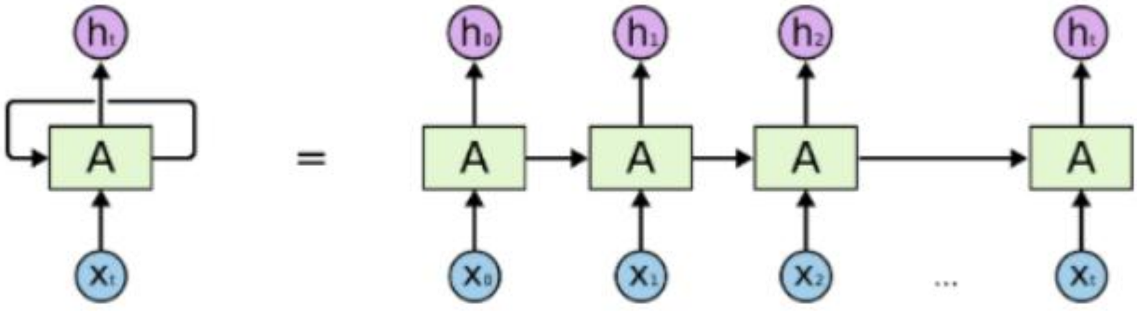


Figure: Neural Network

4) – Recurrent Neural Network – The basic idea of RNN is to make use of sequential information. In traditional neural networks the data was assumed to be independent, but as we know data points in Time Series are not dependent, future values depend on previous values. A RNN has loops in them that allow information to be carried in neurons while reading the inputs but RNN has few drawbacks like vanishing and exploring gradient problem.



An unrolled RNN

5) – LSTM (Long Short Term Memory) –To overcome vanishing gradient problem of Recurrent Neural Network, LSTM was made, which uses a different function to compute the hidden state. LSTMs also have chain structure but instead of having a single neural network layer, it uses 4 – cell, an input gate, an output gate and a forget gate as shown in figure.

Justification - LSTM will be very useful in our problem as our data is time – series data, it can capture recurrence relation of the data.

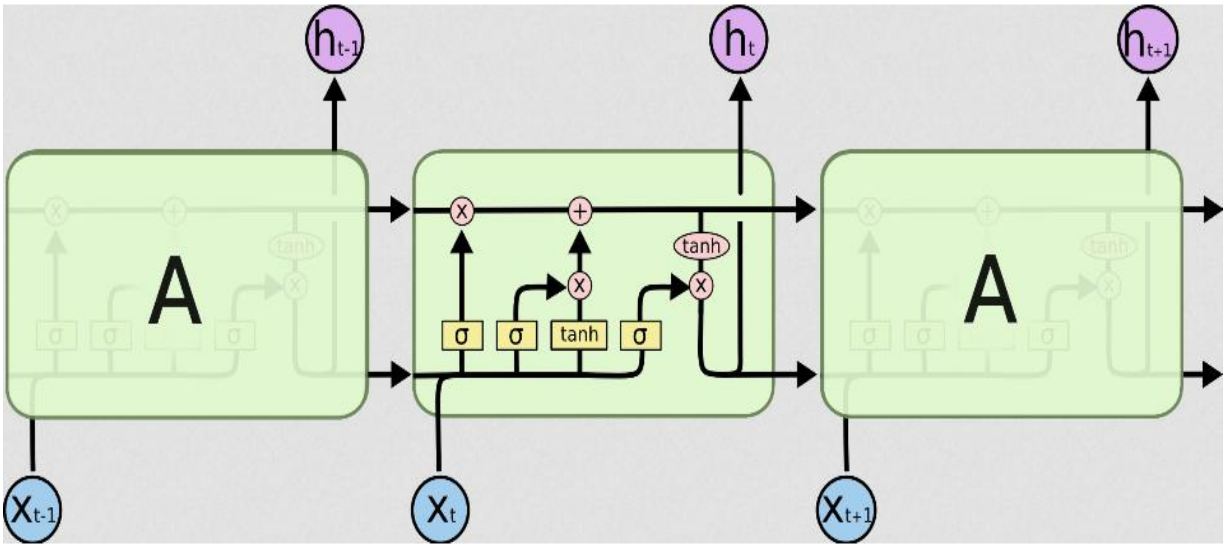


Figure : LSTM

6 Benchmark Model –

Benchmark models are very useful in every project. Benchmark models are chosen as simple algorithm that can help to set baseline accuracy.

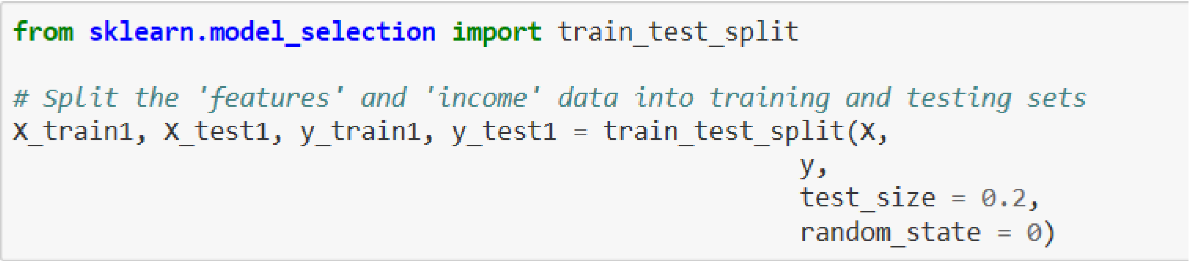
For the benchmark model I extracted statistical features (mean, Std, Skew, Kurt) from every time-series and then train it on SVM and Logistic Regression.

Methodology –

1 Pre-processing the data –

a) In this step first we will first separate data (X) and label (y) from raw dataset.

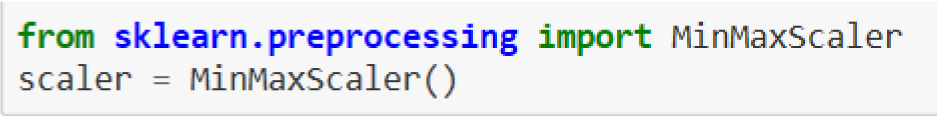
b) Then split the data into train and test in ratio of 80 and 20.



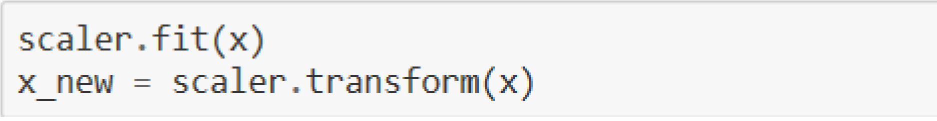
```
from sklearn.model_selection import train_test_split

# Split the 'features' and 'income' data into training and testing sets
X_train1, X_test1, y_train1, y_test1 = train_test_split(X,
                                                        y,
                                                        test_size = 0.2,
                                                        random_state = 0)
```

c) For training in neural networks we will normalize train and test data using min – max scalar normalization. This normalization is done so that neural networks can converge faster.



```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```



```
scaler.fit(x)
x_new = scaler.transform(x)
```

- **Feature Extraction –** In this step for training on SVM and Logistic Regression we will extract Statistical Features as discussed above.

- **Training on Models –** In this step we will create SVM, Logistic Regression, Neural Network, LSTM for training.

- **Compare the results –** After training on training models we will calculate Accuracy, Precision, Recall, F1 Score for different models.

Implementation –

1 Neural Network –

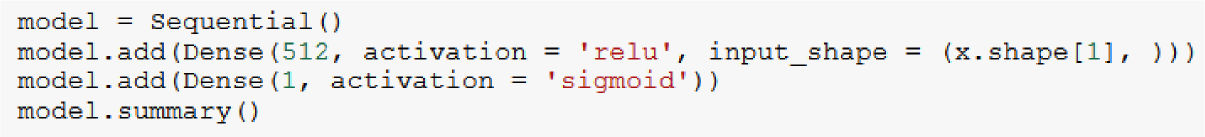
a) - For creating Neural Network Keras library is used with Tensorflow backend.

b) - To make model we have import Sequential model from Keras.
from keras.models import Sequential

c) – Then we can add Dense layers by the following code.
model.add(Dense(number_of_nuerons, activation_function, input_shape))

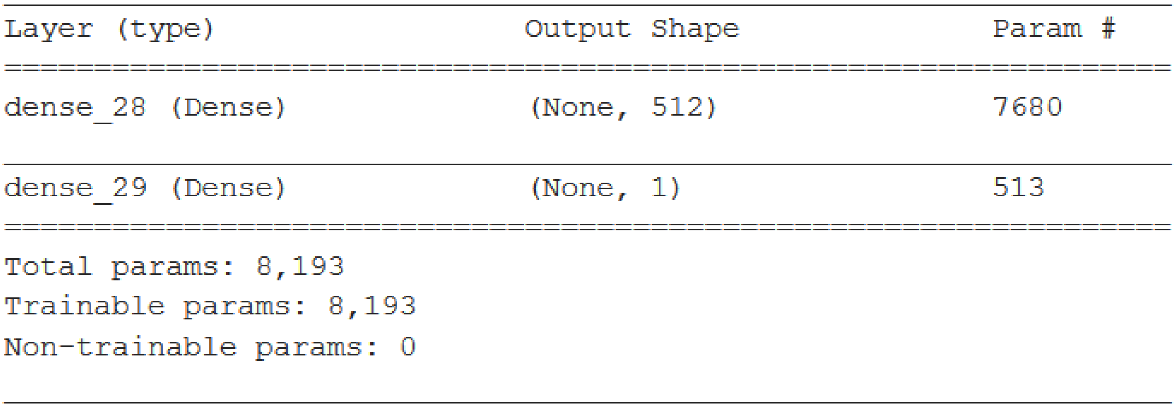
d) – Activation ‘relu’ function is used in hidden layer for faster convergence and ‘sigmoid’ is used at output layer for classification.

Whole model snippet –



```
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (x.shape[1], )))
model.add(Dense(1, activation = 'sigmoid'))
model.summary()
```

Model Summary –

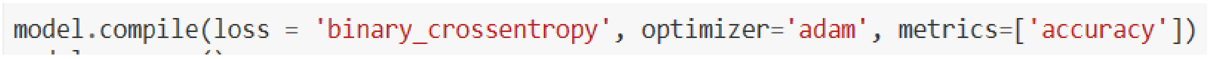


```
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 512)               7680
_________________________________________________________________
dense_29 (Dense)             (None, 1)                 513
=================================================================
Total params: 8,193
Trainable params: 8,193
Non-trainable params: 0
_________________________________________________________________
```

Optimizers – ‘Adam’ to overcome local minima.

Loss function – Binary cross entropy as we have 2 classes to classify.

Code snippet –



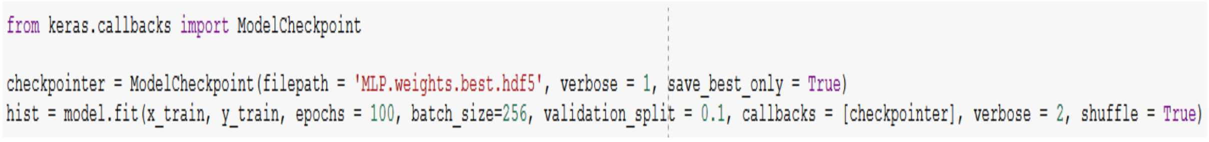
```
model.compile(loss = 'binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Training –
For training fast we divide our dataset into batches, keras provide giving batch size as parameter in training.

We can also do cross validation by further splitting our training set into train and valid set, Keras also provide this feature as parameter in training.

In my training I took –
Epochs = 100
Batch size = 256
Train and valid set split ration = 0.1

code snippet –



```
from keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint(filepath = 'MLP.weights.best.hdf5', verbose = 1, save_best_only = True)
hist = model.fit(x_train, y_train, epochs = 100, batch_size=256, validation_split = 0.1, callbacks = [checkpointer], verbose = 2, shuffle = True)
```

Refined (improved Model)

a) Increased the no. of hidden layers to 4.

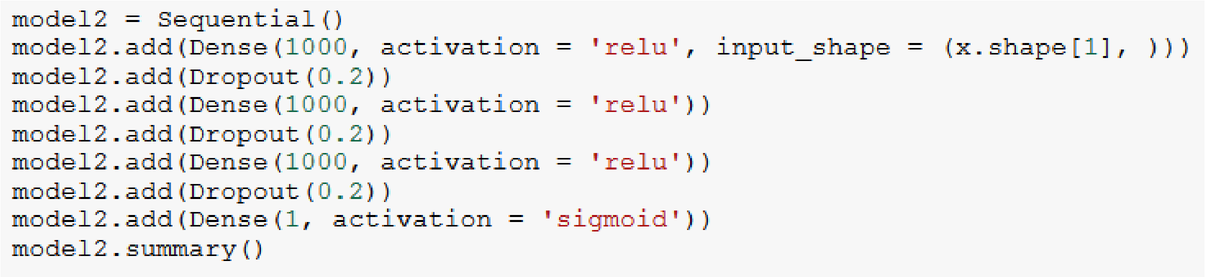
b) Increased the no. of neurons to 1000 to better approx the function of data.

c) At every layer, Dropout with 0.2 probability was set. Dropout was set to overcome the overfitting of neural network.

d) Epochs increased to 200.

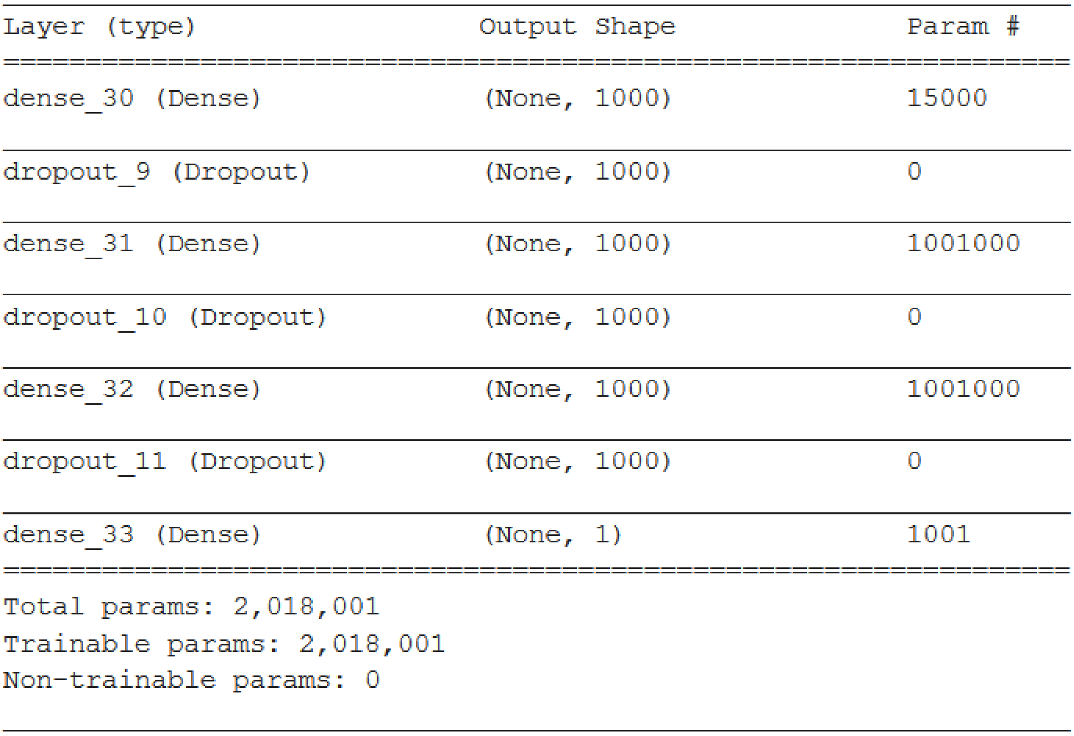
e) Batch Size increased to 512.

Whole model snippet –



```
model2 = Sequential()
model2.add(Dense(1000, activation = 'relu', input_shape = (x.shape[1], )))
model2.add(Dropout(0.2))
model2.add(Dense(1000, activation = 'relu'))
model2.add(Dropout(0.2))
model2.add(Dense(1000, activation = 'relu'))
model2.add(Dropout(0.2))
model2.add(Dense(1, activation = 'sigmoid'))
model2.summary()
```

Model Summary –

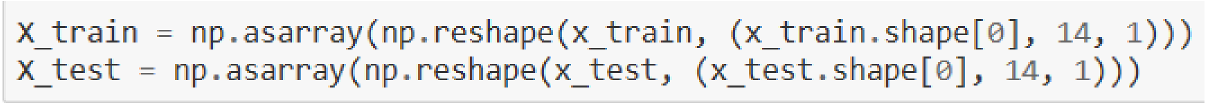


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_30 (Dense)             (None, 1000)              15000     
_________________________________________________________________
dropout_9 (Dropout)          (None, 1000)              0         
_________________________________________________________________
dense_31 (Dense)             (None, 1000)              1001000   
_________________________________________________________________
dropout_10 (Dropout)         (None, 1000)              0         
_________________________________________________________________
dense_32 (Dense)             (None, 1000)              1001000   
_________________________________________________________________
dropout_11 (Dropout)         (None, 1000)              0         
_________________________________________________________________
dense_33 (Dense)             (None, 1)                 1001      
=================================================================
Total params: 2,018,001
Trainable params: 2,018,001
Non-trainable params: 0
_________________________________________________________________
```

2 LSTM –

a) First we have to convert our dataset into 3D as LSTM takes 3D array as input.

Code snippet –



```
X_train = np.asarray(np.reshape(x_train, (x_train.shape[0], 14, 1)))
X_test = np.asarray(np.reshape(x_test, (x_test.shape[0], 14, 1)))
```

X_train.shape[0] gives us total numbers of dataset rows.

14 is used as window size as we want our LSTM to observe all 14 values at a time.

1 indicates that one feature is one observation at a time step.

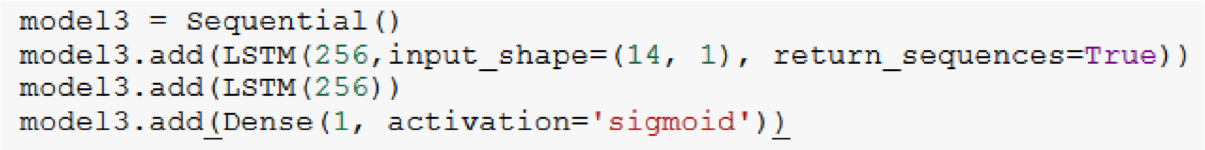
b) Create a Sequential Model in Keras.

c) Then add LSTM layer using following code snippet –

model.add(LSTM(no_of_neurons, input_shape, return_sequences = True))

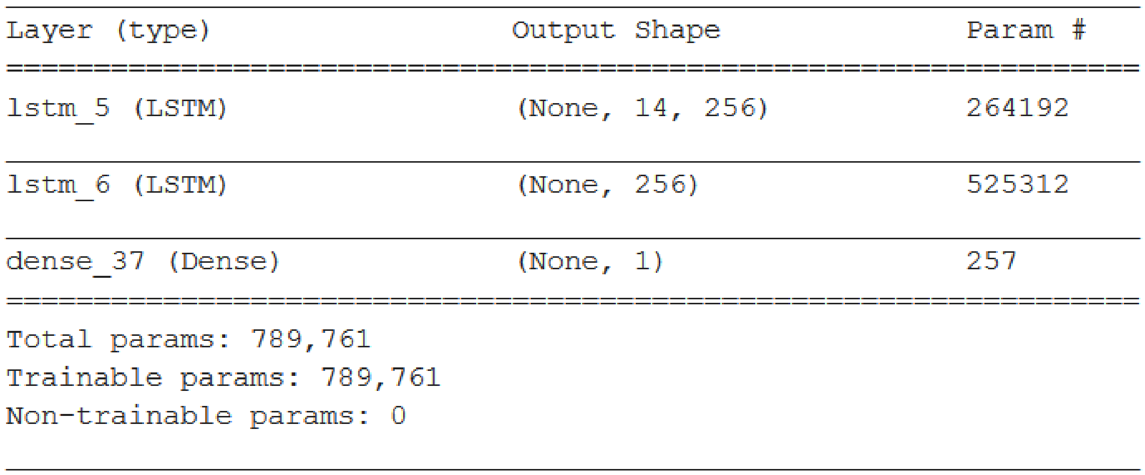
return_sequences - returns the hidden state output for each input time step.

Whole code snippet –



```
model3 = Sequential()
model3.add(LSTM(256,input_shape=(14, 1), return_sequences=True))
model3.add(LSTM(256))
model3.add(Dense(1, activation='sigmoid'))
```

Model summary –



```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 14, 256)           264192
_________________________________________________________________
lstm_6 (LSTM)                (None, 256)               525312
_________________________________________________________________
dense_37 (Dense)             (None, 1)                 257
=================================================================
Total params: 789,761
Trainable params: 789,761
Non-trainable params: 0
_________________________________________________________________
```

Refined (Improved Model) –

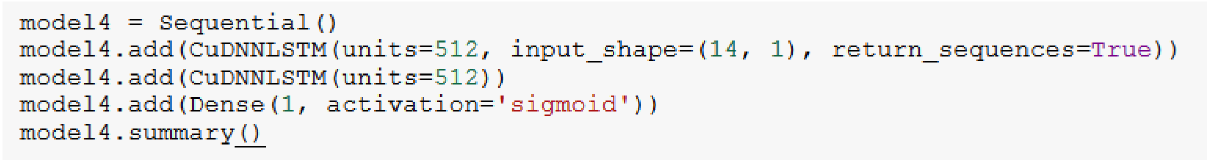
a) For decreasing the training time instead of using LSTM layer, Fast LSTM implementation backed by CuDNN is used.

b) Increased the output size of each layer to 512.

c) Increased epochs to 300.

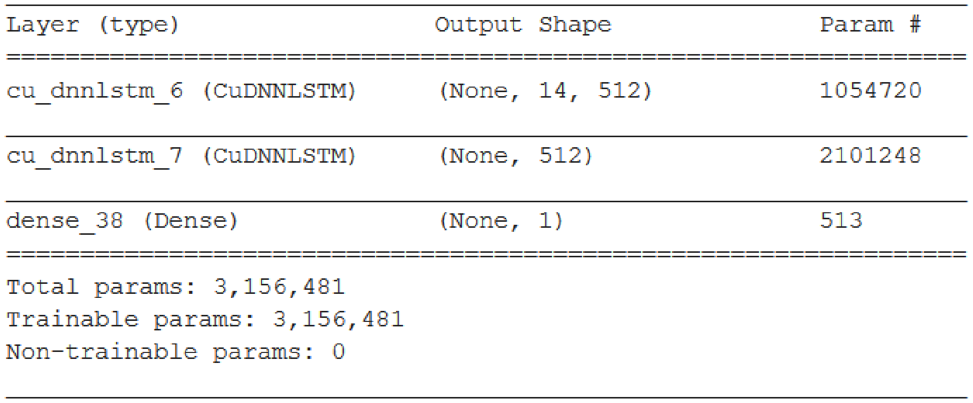
d) Increased Batch size to 512.

Code Snippet –



```
model4 = Sequential()
model4.add(CuDNNLSTM(units=512, input_shape=(14, 1), return_sequences=True))
model4.add(CuDNNLSTM(units=512))
model4.add(Dense(1, activation='sigmoid'))
model4.summary()
```

Model Summary –



```
Layer (type)                 Output Shape              Param #
=================================================================
cu_dnnlstm_6 (CuDNNLSTM)     (None, 14, 512)           1054720
_________________________________________________________________
cu_dnnlstm_7 (CuDNNLSTM)     (None, 512)               2101248
_________________________________________________________________
dense_38 (Dense)             (None, 1)                 513
=================================================================
Total params: 3,156,481
Trainable params: 3,156,481
Non-trainable params: 0
_________________________________________________________________
```

Problems faced – Main problem I faced was that 3D shape input, first I read documentation of LSTM keras and I read some blogs to understand what is meaning of 3D shaped input.

Results –

1 Model Evaluation and Validation –

Final Improved Neural Network Model –

No. of Hidden Layers – 3
No. of Neuron each layer – 1000
Hidden Layer Activation function – 'relu'
Output Layer Activation function – 'sigmoid'
Dropout probability at each layer – 0.2
Epochs – 200
Batch-Size- 512
Loss Function – Binary Cross Entropy
Optimizer – Adam

Final Improved LSTM model –

No. of CuDNNLSTM layer – 2
Output units = 512
Output Layer activation – 'sigmoid'
Epochs – 300

Batch-Size – 512
Loss Function – Binary Cross Entropy
Optimizer - Adam

2 Robustness of the model –

After adding dropout in Neural Networks its accuracy increased by +5%. I ran the program several times and every shuffled the dataset, every time Neural Networks was giving 70+ accuracy with standard deviation of 7%.

After changing to CuDNNLSTM layer training time drastically decreased by 2 times and this model was giving accuracy of 65 with standard deviation of 6%.

3 Justification –

Benchmark Model –

SVM –

Accuracy = 0.6989319092122831
Precision = 0.7059190031152648
Recall = 0.7248880358285349
F1 Score = 0.7152777777777777

Logistic Regression –

Accuracy = 0.5240320427236315
Precision = 0.5404607206142942
Recall = 0.5854126679462572
F1 Score = 0.5620393120393121

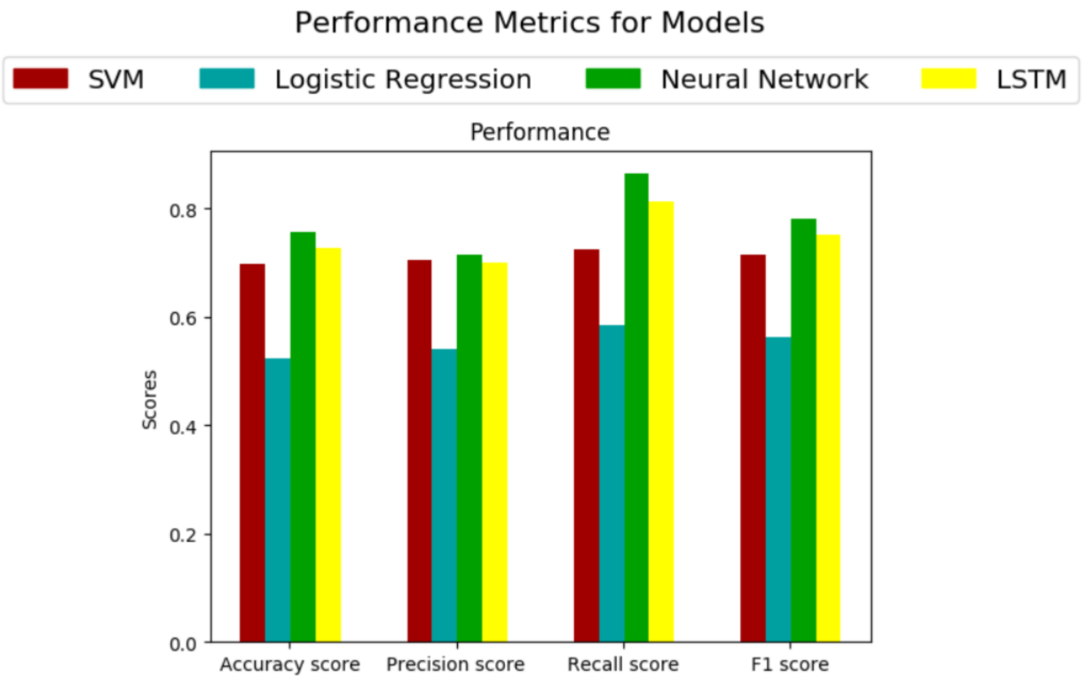
Neural Network –

Accuracy = 0.7557788944723618
Precision = 0.7142857142857143
Recall = 0.8642892521050025
F1 Score = 0.7821604661586732

LSTM –

Accuracy = 0.7281407035175888
Precision = 0.6999573196756296
Recall = 0.8122833085685983
F1 Score = 0.7519486474094451

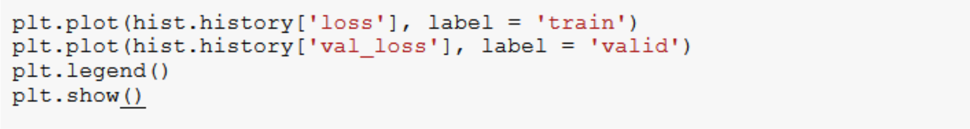
As we can see from results Neural Network and LSTM are performing better than Benchmark models.



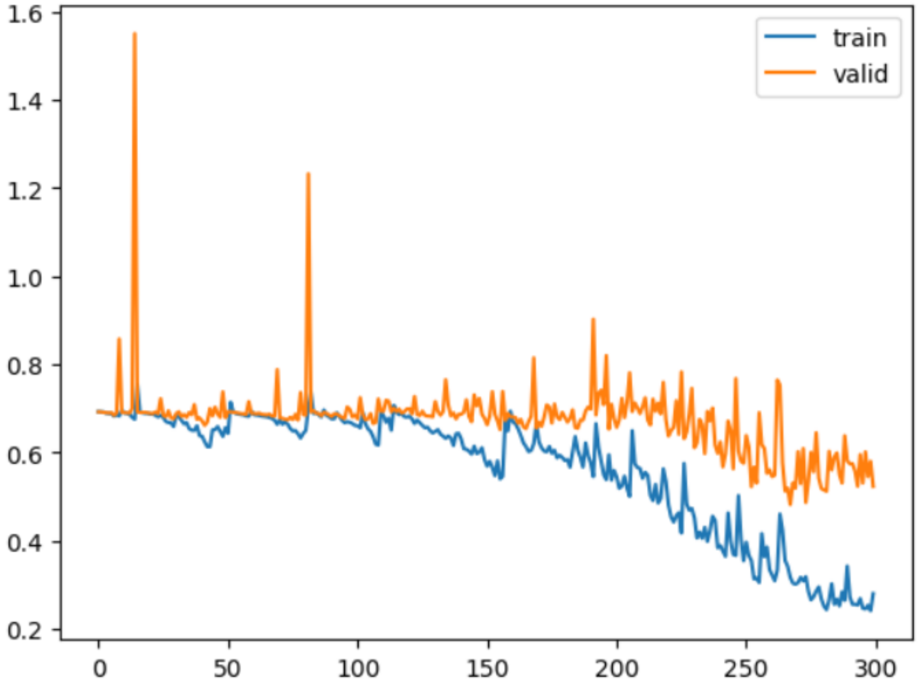
Conclusion –

1 Free-Form Visualization –

In training any neural network very important topic is its convergence of loss function. Convergence helps us to know whether the model is under-fit or over-fit. The below figure shows the learning curve of the training process of LSTM model. We can observe that loss for training and validation set is decreasing as epochs are increasing. From the graph we can say our model is neither over-fitted nor under-fitted.



```
plt.plot(hist.history['loss'], label = 'train')
plt.plot(hist.history['val_loss'], label = 'valid')
plt.legend()
plt.show()
```



REFLECTION –

To repeat the results follow this steps –

- **Pre-processing the data –**
  - First separate data (X) and label (y) from raw dataset.
  - Split the data into train and test in ratio of 80 and 20.
  - Normalize the data.
  - Extract Statistical Feature.
- Develop Benchmark Model –
  - Train SVM and Logistic Regression on extracted features.
  - Evaluate the performance.
- Create Neural Network and LSTM
  - Train it on the dataset.
  - Evaluate the performance.

I chosen the EEG classification after reading some new on Brain Computer Interaction. The project was fun to create and I learned a lot.

Major problems I faced was training on LSTM as it was new to me and I have to go through a lot of videos and blogs to build intuitions of LSTM.

IMPROVEMENT

The main reason for LSTM and Neural Network not able to accuracy greater than 75% is **EEG signals are Non – Stationary signals** means there frequency keep on changing.

To further improving the model we need to use signal processing techniques and take it to spatial domain (Fourier Transform) or Time – Frequency Domain to better capture those features.

References –

1. Sanei, Saeid, and Jonathon A. Chambers. EEG signal processing. John Wiley & Sons, 2013.

2. Rosler, Oliver, and David Suendermann. *A first step towards eye state prediction using eeg*. Proc. of the AIHLS (2013).

3. Neha Jain, Sandeep Bhargava, Savita Shivani, Dinesh Goyal. *Eye State prediction using eeg by supervised learning*. International Journal of Science, Engineering and Technology.