JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY
SECTOR 62

MINOR PROJECT REPORT:

EARTHQUAKE PREDICTION USING LONG SHORT TERM MEMORY APPROACH

Name of Group Members	Enrollment Number	Batch
Pranab Sharma	16103055	B9
Rohit Singh	16103258	B9
Manya Agrawal	16103313	B9

ACKNOWLEDGEMENT

This project consumed huge amount of work, research and dedication. Implementation would not have been possible if we did not have support, supervision and assistance of many individuals, therefore we would like to extend our sincere gratitude to all of them.

We are pleased to acknowledge Mr Mahendra Gurve and Mrs Ankita Wadhwa for their invaluable guidance, support and devoting their time and knowledge in the implementation, during the course of this project work.

Our thanks and appreciations also goes to our colleagues in developing the project and people who have willingly helped us out with their abilities.

CONTENTS

INTRODUCTION

Earthquake is a phenomenon in which the tectonic plates of the lithosphere rub against each other to release energy, thereby resulting in tremors on the crust. The epicentre of an earthquake, if lies under human establishment, can cause casualties and huge damage to infrastructure. If it lies near coastal regions, it can lead to tsunamis and loss of life.

Unlike hurricanes or cyclones, earthquakes are independent from weather conditions and therefore it is tough for geologists to predict the earthquakes. Before earthquake occurs, it does not lead us to many variables which can be extracted from nature, to help us predict its outcome. Therefore it becomes a challenge to predict the earthquakes that can happen in future. If the earthquakes could be predicted beforehand, it could prevent economic and human losses.

Geologists keep a record of the earthquakes that have occurred in the past, with all the relevant information pertaining to the event; few of them being magnitude of the earthquake, latitude/longitude of the epicentre, occurrence depth, tectonic plates involved, etc.

We are proposing an implementation of an algorithm(Long Short-Term memory) that can help us to predict the earthquakes around the Indian tectonic plate and the Eurasian tectonic plate , using previous data (both spatial and temporal) that has been collected by geologists and we check the accuracy of our results by comparing the actual data with the predicted outputs using the two models.

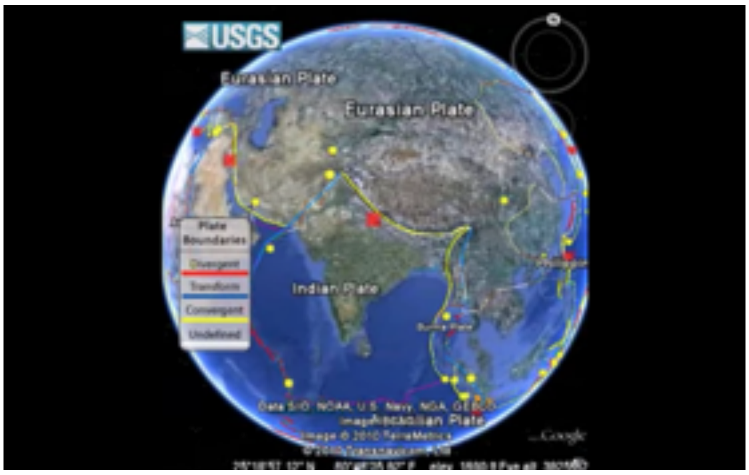
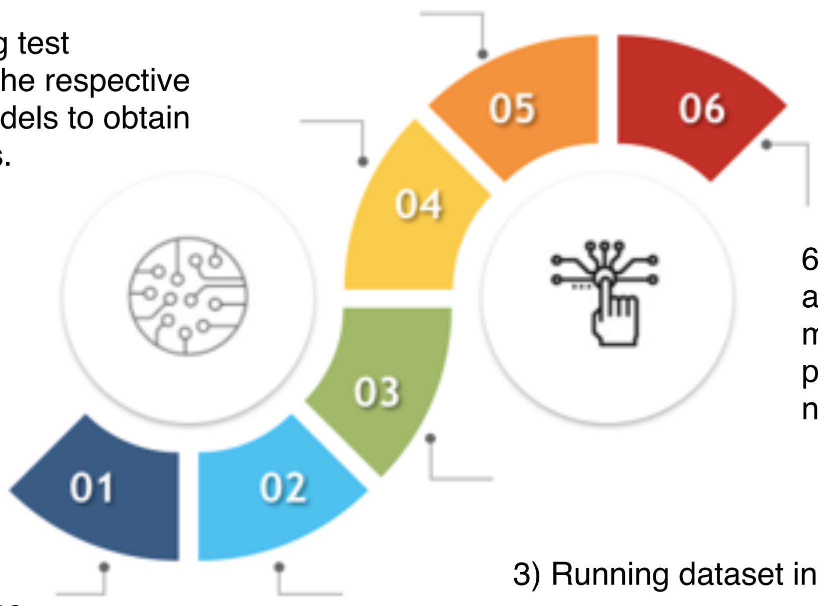


Image credits: EZSnips (Youtube)

OBJECTIVE

Our primary objective is to forecast the earthquakes in a given region using the dataset, utilising all the variables available to us by observing nature. We break down the primary objective into more tangible targets as follows:



5) Displaying the predicted earthquake regions over world map.

4) Inputting test data over the respective trained models to obtain predictions.

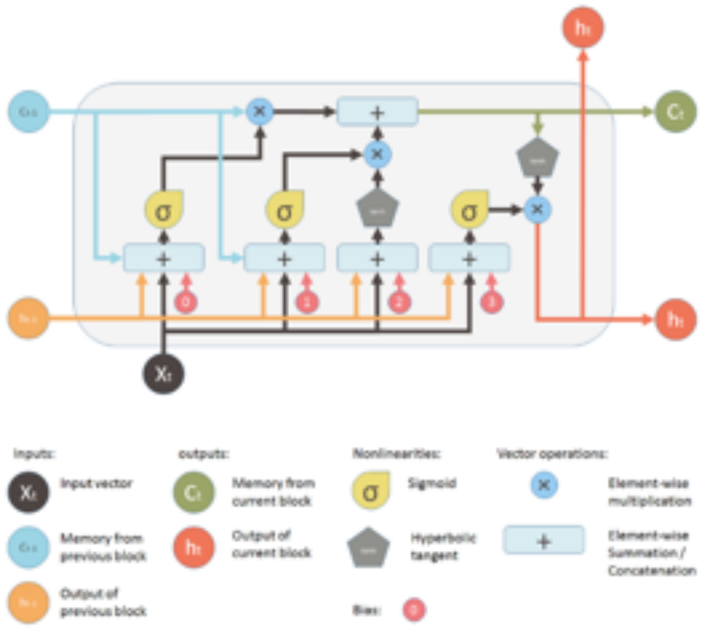
6) Comparing accuracies of both models. (with true positive and true negative outcomes)

1) Preprocessing the data as per requirement. Assigning the coordinates to different grid sectors across the map, forming buckets.

2) Forming the input vector, with frequencies across grid, for LSTM models(1D and 2D, respectively) cell.

3) Running dataset into both the models for optimum number of epochs till the loss is standardised, therefore training them.

BACKGROUND STUDY AND FINDINGS



WORKING PRINCIPLE OF LSTM CELL

I. Aforementioned diagram is the basic structure of a LSTM cell. Matrix multiplication of memory from previous block [C(t-1)] with output of forget gate provides us the intermediate new memory. This in turn is matrix added with the output of new memory gate to obtain the new memory [C(t)] generated by the cell. This is done as per our requirements of how much previous memory we want the cell to forward.

II. The first gate from left is called *FORGET GATE*. The inputs are added and applied to the sigmoid function to obtain the output of this gate, which lies between 0 and 1 (closer to 0 means forget, closer to 1 means don't forget).

III. The next gate is called *NEW MEMORY GATE*. Output of the forget gate is matrix multiplied with the new memory formed (tanh{h[t-1]+X(t)+b}): tanh() function helps to regulate the network. This is the output of the gate as discussed in (I).

IV. The final gate is called *OUTPUT GATE*. Outputs of forget gate and new memory are matrix multiplied to finally form the output of our cell (h(t)). C(t) and h(t) will be fed forward as inputs to the next cell.

Observations wrt Long Short-Term Memory Algorithm:

Upon examining the working principles and its possible application of LSTM we conclude-

- Input of LSTM must always be 3D. (units, input shape, and return sequences (Input data must be reshaped if it isn't in 3D format))
- Output of LSTM must always be in 2D or 3D (Batch size, units, return sequences)

Observations wrt. Earthquake prediction:

- We propose to make prediction on earthquakes by taking advantages of the spatio-temporal correlations among them. The intuition is that earth is connected, and hence the seismic activities in one location will naturally lead to seismic activities in other locations
- The seismic activities tend to have certain patterns in the time domain.

MODELS OF LSTM

2D INPUT LSTM: For each set of 6 months (temporal inclusion), each coordinate in train_set is assigned a grid number (spatial inclusion) separately. The gird number along with frequency of earthquakes in the grid becomes the new base for hot vector. The new array is reshaped into 3D and then fed in the LSTM model. The output is 2D and is fed to the next epoch.

1D INPUT LSTM: Since the input variable must be one, we choose the spatial approach. For each location (latitude and longitude), the magnitudes for the input vector. This is processed and fed into the LSTM cell. Resulting output is fed in the next LSTM cell till the epoch is over.

In each of the models, the LSTM gates have been implemented by the following functions: (explained in depth under *designing in detail*)

$$i_t = \sigma(Wix.xt + Wih.ht-1 + Wic.ct-1 + bi)$$

$$f_t = \sigma(Wfx.xt + Wfh.ht-1 + Wfc.ct-1 + bf)$$

$$c_t = f_t \times c_{t-1} + i_t \times \phi(W_{cx}.x_t + W_{ch}.h_{t-1} + b_c)$$

$$o_t = \sigma(W_{ox}.x_t + W_{oh}.h_{t-1} + W_{oc}.c_t + b_o)$$

$$h_t = o_t \times \phi(c_t)$$

<u>WHY WAS RNN NOT SUFFICIENT?</u>

During back propagation, the RNN suffers from *<u>VANISHING GRADIENT PROBLEM</u>*. The gradient is a value used to update the neurons weights. In RNNs the gradient shrinks as it back propagates through time. When the value becomes too small, it doesn't contribute too much in learning. Therefore the neurons that get smaller gradient don't learn. This means RNN suffers from short term memory.

*new weight = weight - learning_rate*gradient*

2.0999.   =. 2.1.   - 0.001 (almost equal to 2.1)

<u>OUTCOMES:</u>

We are able to predict the earthquakes using both our models with the following accuracies:

- Accuracy of 1D LSTM Model: True Positive:

  True Negative:

- Accuracy of 2D LSTM Model: True Positive:

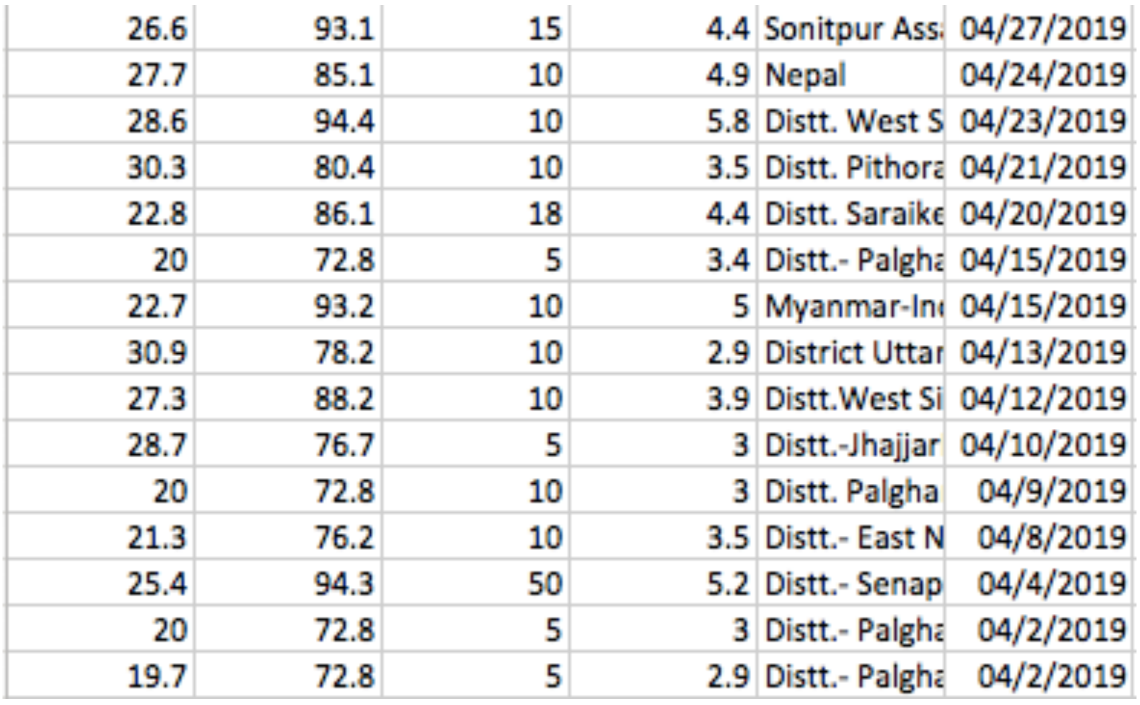
  True Negative:

Therefore we find that 2D LSTM Model is a better method to predict the earthquake (implied from its Spatio-Temporal application in the model)

DESIGNING IN DETAIL

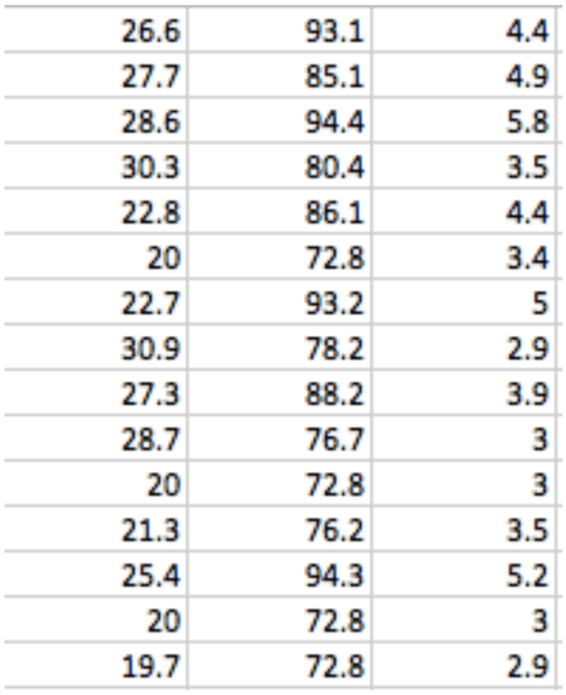
PHASE 1: PROCESSING THE DATA

The dataset file retrieved from an online source consists of Earthquakes occurred in the past around the Indian Tectonic plate.



LATITUDE	LONGITUDE	DEPTH	MAGNITUDE	REGION	DATE
26.6	93.1	15	4.4	Sonitpur Ass	04/27/2019
27.7	85.1	10	4.9	Nepal	04/24/2019
28.6	94.4	10	5.8	Distt. West S	04/23/2019
30.3	80.4	10	3.5	Distt. Pithora	04/21/2019
22.8	86.1	18	4.4	Distt. Saraike	04/20/2019
20	72.8	5	3.4	Distt.- Palgha	04/15/2019
22.7	93.2	10	5	Myanmar-In	04/15/2019
30.9	78.2	10	2.9	District Uttar	04/13/2019
27.3	88.2	10	3.9	Distt.West Si	04/12/2019
28.7	76.7	5	3	Distt.-Jhajjar	04/10/2019
20	72.8	10	3	Distt. Palgha	04/9/2019
21.3	76.2	10	3.5	Distt.- East N	04/8/2019
25.4	94.3	50	5.2	Distt.- Senap	04/4/2019
20	72.8	5	3	Distt.- Palgha	04/2/2019
19.7	72.8	5	2.9	Distt.- Palgha	04/2/2019

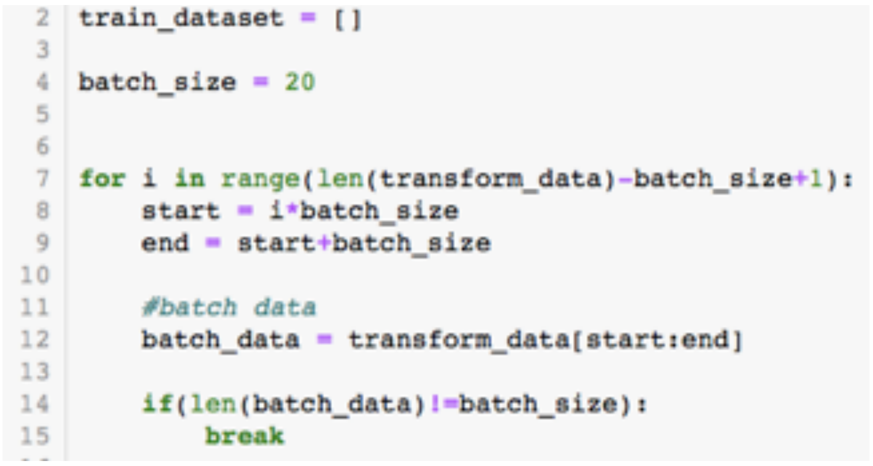
In order to make this dataset suitable for our processing, we retain the columns relevant to us (Latitude, Longitude and Magnitude) while dropping the rest of the information.



LATITUDE	LONGITUDE	MAGNITUDE
26.6	93.1	4.4
27.7	85.1	4.9
28.6	94.4	5.8
30.3	80.4	3.5
22.8	86.1	4.4
20	72.8	3.4
22.7	93.2	5
30.9	78.2	2.9
27.3	88.2	3.9
28.7	76.7	3
20	72.8	3
21.3	76.2	3.5
25.4	94.3	5.2
20	72.8	3
19.7	72.8	2.9

FOR 1D LSTM IMPLEMENTATION:

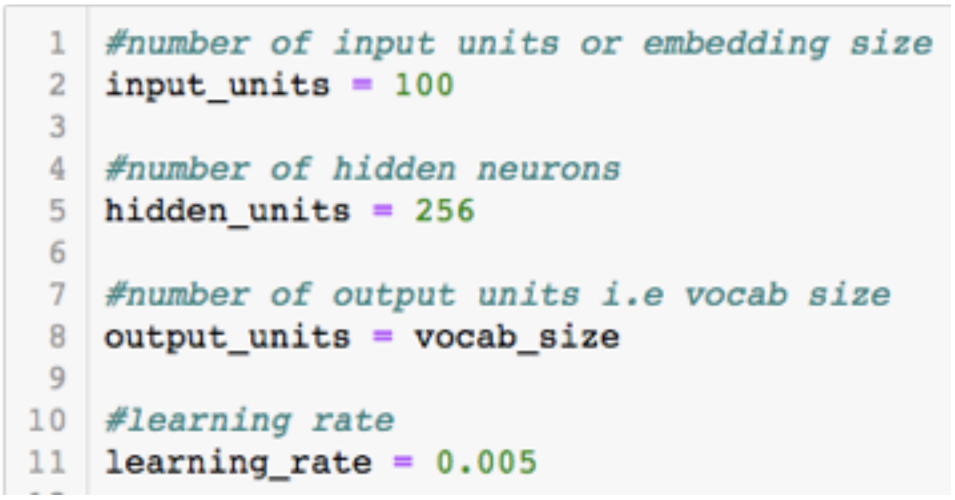
Batch_size is the window of our input which is fed in the LSTM during training. It has been taken as 20 in our code. Start and end is defined for each batch.



```
train_dataset = []

batch_size = 20


for i in range(len(transform_data)-batch_size+1):
    start = i*batch_size
    end = start+batch_size

    #batch data
    batch_data = transform_data[start:end]

    if(len(batch_data)!=batch_size):
        break
```

The dataset is divided to train data and test data, 80% belonging to train data and 20% belonging to test data. The resultant batches in a list will be carried to our model, acting as train data.

After multiple runs on the model, following inputs are finalised for the LSTM cell, pertaining optimum results.

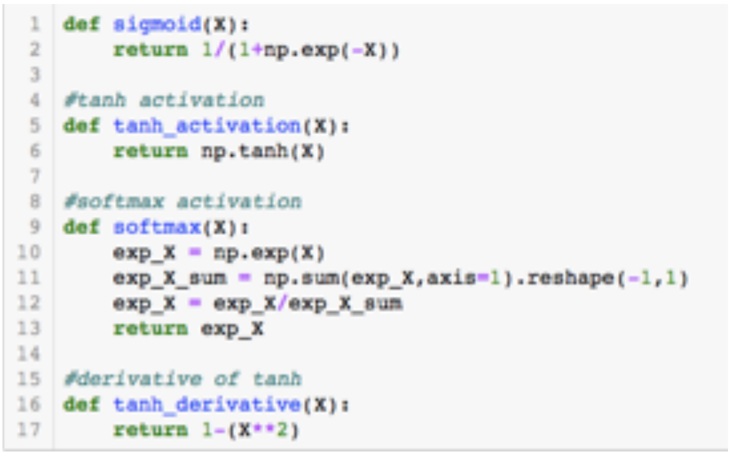


```
#number of input units or embedding size
input_units = 100

#number of hidden neurons
hidden_units = 256

#number of output units i.e vocab size
output_units = vocab_size

#learning rate
learning_rate = 0.005
```

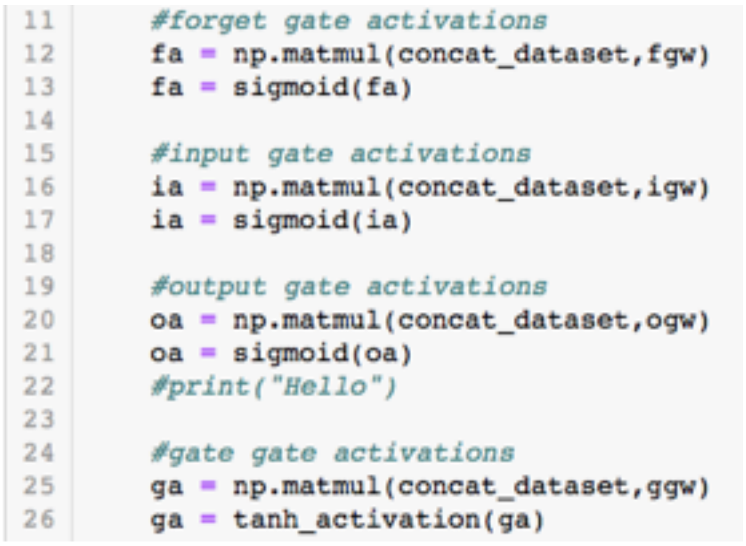
<u>Implementing the LSTM cell:</u>

Our LSTM cell, as aforementioned will have four gates. Forget Gate, Input Gate, Output Gate and Gate Gate. The parameters are initialised of these gates, including their weights. The weights are initially initialised as per mean and standard deviation values (0 and 0.1).

Under def lstm_cell, these parameters are taken through function signature and gate activations are calculated as per algorithm mentioned in the research paper taken as reference. These gate activation values are normalised by using sigmoid and tanh functions to ensure that the values do not overpower or under power the other values in our hot vector.



```python
def sigmoid(X):
    return 1/(1+np.exp(-X))

#tanh activation
def tanh_activation(X):
    return np.tanh(X)

#softmax activation
def softmax(X):
    exp_X = np.exp(X)
    exp_X_sum = np.sum(exp_X,axis=1).reshape(-1,1)
    exp_X = exp_X/exp_X_sum
    return exp_X

#derivative of tanh
def tanh_derivative(X):
    return 1-(X**2)
```



```python
    #forget gate activations
    fa = np.matmul(concat_dataset,fgw)
    fa = sigmoid(fa)

    #input gate activations
    ia = np.matmul(concat_dataset,igw)
    ia = sigmoid(ia)

    #output gate activations
    oa = np.matmul(concat_dataset,ogw)
    oa = sigmoid(oa)
    #print("Hello")

    #gate gate activations
    ga = np.matmul(concat_dataset,ggw)
    ga = tanh_activation(ga)
```

These activation values are stored in a list to be used in back propagation. Using the values calculated with the formulae below, the output values i.e. h(t) and c(t) are calculated. during forward propagation.

$$i_t = \sigma(Wix.xt + Wih.ht{-}1 + Wic.ct{-}1 + bi)$$

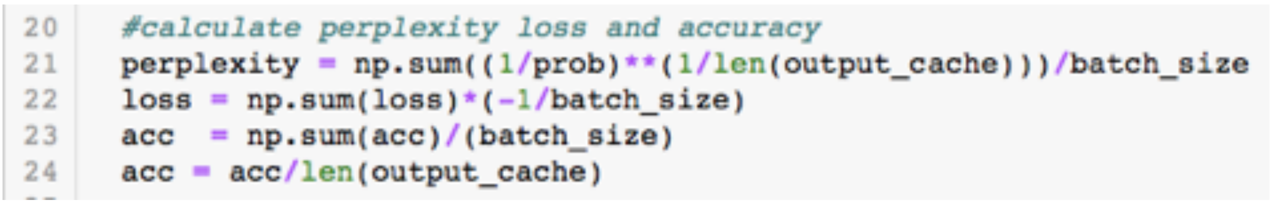
$$f_t = \sigma(Wfx.xt + Wfh.ht{-}1 + Wfc.ct{-}1 + bf\ )$$

$$c_t = f_t \times c_{t-1} + i_t \times \phi(W_{cx}.x_t + W_{ch}.h_{t-1} + b_c)$$

$$o_t = \sigma(W_{ox}.x_t + W_{oh}.h_{t-1} + W_{oc}.c_t + b_o)$$

$$h_t = o_t \times \phi(c_t)$$

Before calling the training function, we must define the parameters by defining them,



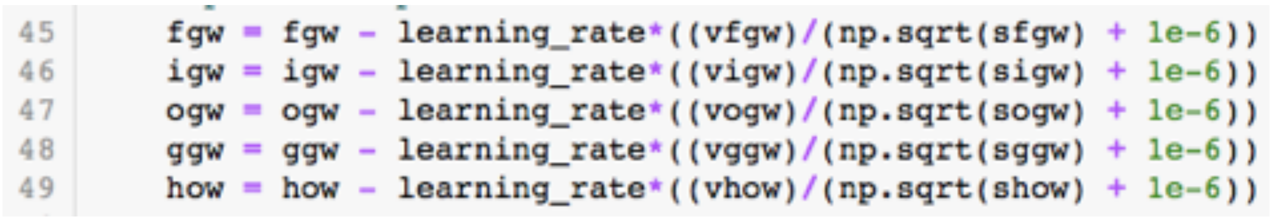
```
#calculate perplexity loss and accuracy
perplexity = np.sum((1/prob)**(1/len(output_cache)))/batch_size
loss = np.sum(loss)*(-1/batch_size)
acc  = np.sum(acc)/(batch_size)
acc = acc/len(output_cache)
```

1. Loss, perplexity, accuracy of each batch.

2. Storing errors of each gate in LSTM cell along with activation and input errors and storing them in a list lstm_error[].

3. Calculating single LSTM cell derivatives along with each output cell derivative. We have defined two functions for them respectively.

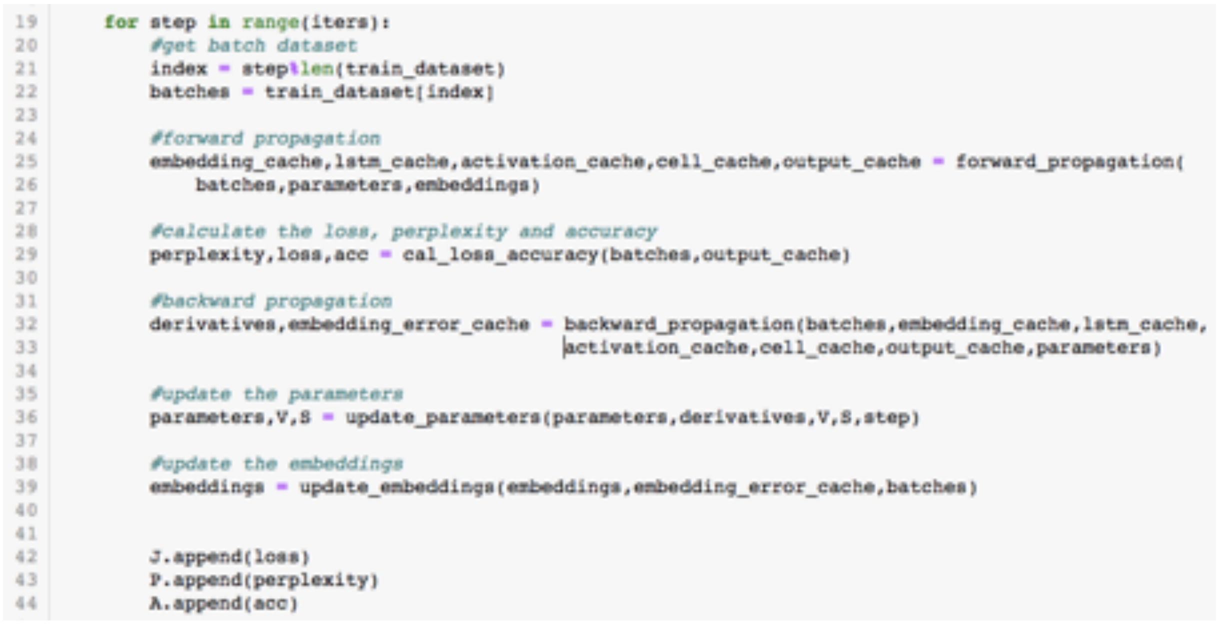
Updation of each gate parameters is done as follows:



```
    fgw = fgw - learning_rate*((vfgw)/(np.sqrt(sfgw) + 1e-6))
    igw = igw - learning_rate*((vigw)/(np.sqrt(sigw) + 1e-6))
    ogw = ogw - learning_rate*((vogw)/(np.sqrt(sogw) + 1e-6))
    ggw = ggw - learning_rate*((vggw)/(np.sqrt(sggw) + 1e-6))
    how = how - learning_rate*((vhow)/(np.sqrt(show) + 1e-6))
```

The def train function calls for all the relevant functions described above along with other intermediate functions. First, the train set is sent to for forward propagation. One batch passes through the LSTM cell, the weights are updated and activation functions are tapped, producing the first h(t) and c(t) of the cell. As other batches continue to propagate in the cell, each of these outputs are fed for the back propagation. Simultaneously, we calculate the loss, perplexity and accuracy for each batch.

After calculating the accuracy, loss and perplexity we begin our predictions. The untapped test_data is now used for the prediction. The LSTM cell has been trained as per our requirements, pertaining to optimum results. The test data is now fed in the trained cell to obtain the predicted values of earthquake for given magnitudes. These outputs are reshaped to and processed with the train_data latitude and longitudes to produce the final output.



```
    for step in range(iters):
        #get batch dataset
        index = step%len(train_dataset)
        batches = train_dataset[index]

        #forward propagation
        embedding_cache,lstm_cache,activation_cache,cell_cache,output_cache = forward_propagation(
            batches,parameters,embeddings)

        #calculate the loss, perplexity and accuracy
        perplexity,loss,acc = cal_loss_accuracy(batches,output_cache)

        #backward propagation
        derivatives,embedding_error_cache = backward_propagation(batches,embedding_cache,lstm_cache,
                                            activation_cache,cell_cache,output_cache,parameters)

        #update the parameters
        parameters,V,S = update_parameters(parameters,derivatives,V,S,step)

        #update the embeddings
        embeddings = update_embeddings(embeddings,embedding_error_cache,batches)


        J.append(loss)
        P.append(perplexity)
        A.append(acc)
```

FOR 2D LSTM IMPLEMENTATION:

We observed that majority of earthquake prediction models out there employed only specific straightforward parameters to predict events to some extent. Drawing a tangent to the work of Qianlong Wang et al. and building on it we proposed a method to efficiently predict such events in large regions. We took advantage of the fact that a large amount of data is available in the public sector to train neural networks over.

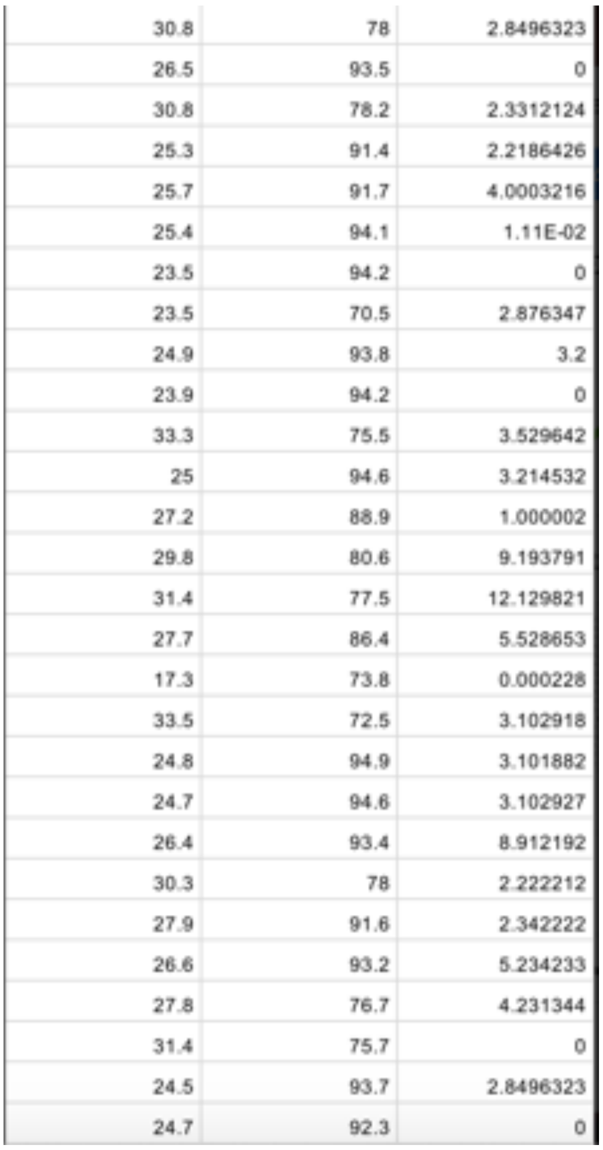
We start by selecting an area of interest essentially an area 30° wide in Longitude and 20° wide in Latitude over the subcontinent of India and sub dividing it into smaller sub sectors to finely determine event prone zones. In our work we decided to go with a 6 by 6 grid that is 36 smaller sub sectors. Now that we have a spatial template we statistically count the event frequency in each subsector in an appropriate timeframe. Each of the hot vectors obtained will together form the input for our LSTM cell.

The internal working and functioning of our LSTM cell model has been shown above.

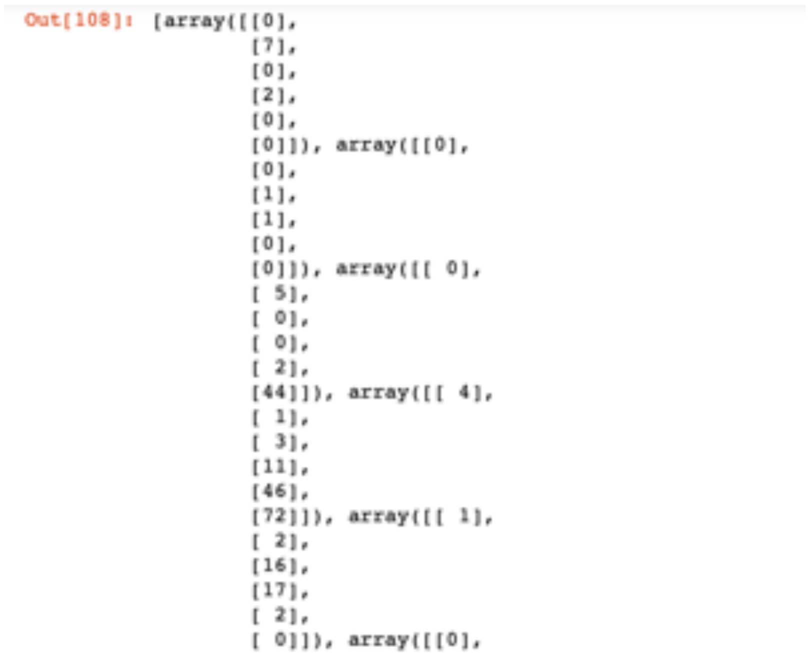
We trained our model and calculated frequencies in a one month window. After the statistical computations we obtained a two dimensional matrix which contains frequencies of earthquake events in each sub sector for every month.

OUTPUTS AND RESULTS

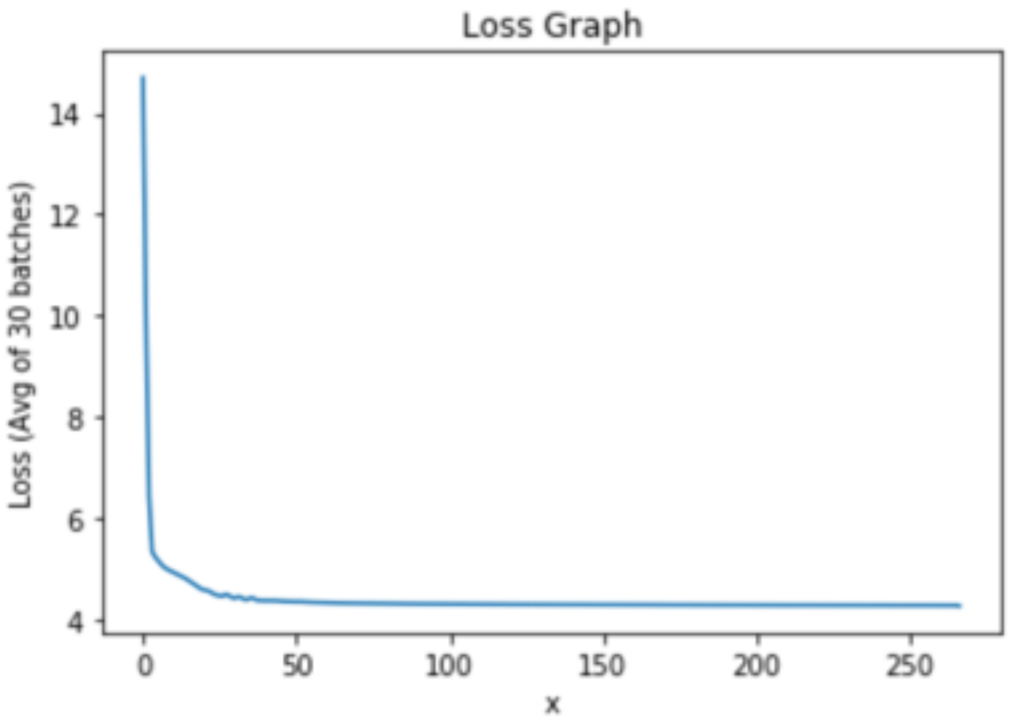
1D LSTM OUTPUT



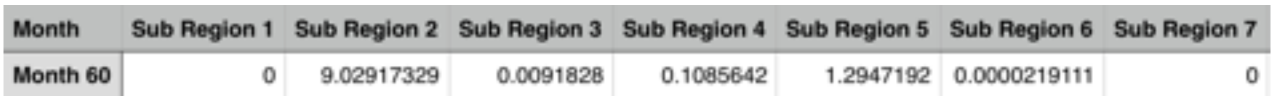
LATITUDE	LONGITUDE	MAGNITUDE
30.8	78	2.8496323
26.5	93.5	0
30.8	78.2	2.3312124
25.3	91.4	2.2186426
25.7	91.7	4.0003216
25.4	94.1	1.11E-02
23.5	94.2	0
23.5	70.5	2.876347
24.9	93.8	3.2
23.9	94.2	0
33.3	75.5	3.529642
25	94.6	3.214532
27.2	88.9	1.000002
29.8	80.6	9.193791
31.4	77.5	12.129821
27.7	86.4	5.528653
17.3	73.8	0.000228
33.5	72.5	3.102918
24.8	94.9	3.101882
24.7	94.6	3.102927
26.4	93.4	8.912192
30.3	78	2.222212
27.9	91.6	2.342222
26.6	93.2	5.234233
27.8	76.7	4.231344
31.4	75.7	0
24.5	93.7	2.8496323
24.7	92.3	0



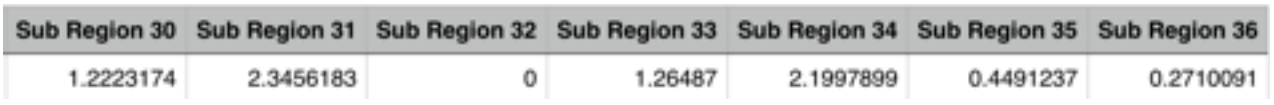
```
Out[108]: [array([[0],
                  [7],
                  [0],
                  [2],
                  [0],
                  [0]]), array([[0],
                  [0],
                  [1],
                  [1],
                  [0],
                  [0]]), array([[ 0],
                  [ 5],
                  [ 0],
                  [ 0],
                  [ 2],
                  [44]]), array([[ 4],
                  [ 1],
                  [ 3],
                  [11],
                  [46],
                  [72]]), array([[ 1],
                  [ 2],
                  [16],
                  [17],
                  [ 2],
                  [ 0]]), array([[0],
```



2D LSTM OUTPUT

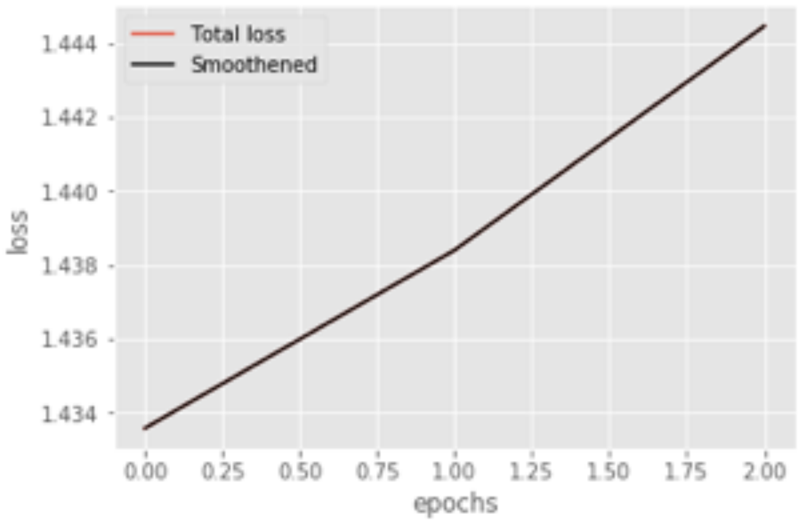


Month	Sub Region 1	Sub Region 2	Sub Region 3	Sub Region 4	Sub Region 5	Sub Region 6	Sub Region 7
Month 60	0	9.02917329	0.0091828	0.1085642	1.2947192	0.0000219111	0



Sub Region 30	Sub Region 31	Sub Region 32	Sub Region 33	Sub Region 34	Sub Region 35	Sub Region 36
1.2223174	2.3456183	0	1.26487	2.1997899	0.4491237	0.2710091

```
[[[-0.47826087 -0.18757835 -0.61083552 -0.96       -1.        ]
  [-0.15942029  0.90988216 -0.67270498 -0.72853333 -0.9998302 ]
  [-0.24637681  0.49532576 -0.09307359 -0.912      -0.99980396]
  ...
  [-0.2173913   0.05216519  0.39971295 -0.78053333 -0.30792738]
  [-0.30434783 -0.01832444 -0.72834104 -0.9344     -0.30765436]
  [-0.53623188  0.3359361   0.8021292  -0.96       -0.30745133]]

 [[-0.39130435 -0.02589634 -0.73698976 -0.97893333 -0.30736613]
  [-0.30434783  0.51717468  0.81327685 -0.93066667 -0.30687417]
  [ 0.07246377  0.87851284 -0.71250209 -0.704      -0.30684894]
  ...
  [ 0.04347826  0.83363301  0.16931423 -0.86266667  0.00811868]
  [-0.30434783  0.95801426 -0.67021069 -0.90666667  0.00813215]
  [-0.33333333  0.82296644  0.17514353 -1.          0.00816689]]

 [[-0.07246377  0.83198539  0.16442785 -0.87173333  0.00834289]
  [-0.01449275  0.95810022 -0.65803653 -0.65386667  0.00847998]
  [-0.1884058   0.26118414  0.62906193 -0.97333333  0.00849402]
  ...
  [-0.01449275  0.56351589  0.84726325 -0.9808      0.58662717]
  [-0.2173913   0.52789856  0.83216283 -0.9504      0.58662868]
  [-0.27536232  0.54889502  0.82038812 -1.          0.5866305 ]]

 [[-0.24637681  0.95768473 -0.65664307 -0.6136      0.58663203]
  [-0.27536232  0.61835309  0.88791037 -1.          0.58664053]
  [ 0.30434783  0.54082883  0.81131672 -0.9912      0.58664561]
  ...
  [-0.1884058   0.47661449 -0.26752504 -0.97306667  0.99989206]
  [-0.39130435  0.92322791 -0.70713264 -0.49866667  0.99989763]
  [-0.24637681 -0.89872846  0.30490682 -1.          1.        ]]]
```



```
lr decreased to 0.125
Epoch 2 avg. batch loss 1.44446
lr decreased to 0.25
Epoch 1 avg. batch loss 1.4384
lr decreased to 0.5
Epoch 0 avg. batch loss 1.43358
Training set loss:
```

OUTCOMES

We are able to predict the earthquakes using both our models with the following accuracies:

- Accuracy of 1D LSTM Model: True Positive:

  True Negative:

- Accuracy of 2D LSTM Model: True Positive:

  True Negative:

Therefore we find that 2D LSTM Model is a better method to predict the earthquake (implied from its Spatio-Temporal application in the model)

LITERATURE SURVEY

REFERENCES

[1] M. Akhoondzadeh and F. J. Chehrebargh. Feasibility of anomaly occurrence in aerosols time series obtained from modis satellite images during hazardous earthquakes. *Advances in Space Research*, 2016.

[2] G. Asencio-Corte´s, F. Martınez-A´lvarez, A. Morales-Esteban, and J. Reyes. A sensitivity study of seismicity indicators in supervised learning to improve earthquake prediction. *Knowledge-Based Systems*, 101:15–30, 2016.

[3] A. Boucouvalas, M. Gkasios, N. Tselikas, and G. Drakatos. Modified- fibonacci dual lucas method for earthquake prediction. In *Third International Conference on Remote Sensing and Geo information of the Environment*, pages 95351A–95351A. International Society for Optics and Photonics, 2015.

[4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[5] S. Kannan. Innovative mathematical model for earthquake prediction. *Engineering Failure Analysis*, 41:89–95, 2014.

[6] M. Last, N. Rabinowitz, and G. Leonard. Predicting the maximum earthquake magnitude from seismic data in israel and its neighboring countries. *PloS one*, 11(1):e0146101, 2016.

[7] C.Liand, X.Liu.An improved pso-bp neural network and its application to earthquake prediction. In *Control and Decision Conference (CCDC), 2016 Chinese*, pages 3434–3438. IEEE, 2016.

[8] S. Narayana Kumar and K. Raja. A bp artificial neural network model for earthquake magnitude prediction in Himalayas, India. *Circuits and Systems*, 7(11):3456, 2016.

[9] A. Panakkat and H. Adeli. Recurrent neural network for approximate earthquake time and location prediction using multiple seismicity indicators. *Computer-Aided Civil and Infrastructure Engineering*, 24(4):280– 292, 2009.

[10] T. Tieleman and G. Hinton. Lecture 6.5-rms prop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[11] X. Wang, L. Gao, S. Mao, and S. Pandey. Csi-based fingerprinting for indoor localization: A deep learning approach. *IEEE Transactions on Vehicular Technology*, 2016.