# LSTM XOR Project Report

**Jayaram Kuchibhotla**
jayaramkuchibhotla@gmail.com

## Abstract

This project report describes the implementation and results of the LSTM model which outputs the parity of the input sequence of binary bits. It has the following sections 1.Introduction 2.Description of the datasets 3.Explanation of model architecture and implementation 4. Results 5.Improvements and Conclusion

## 1 Introduction

RNN(Recurrent Neural Networks) models are useful for training sequential data in the cases where the previous information has to be remembered to predict the output .The basic problem that is solved by RNN is prediction of the next word given few words in a meaningful sentence. Information of the immediate past word is just not sufficient to predict the current word. The context is understood only when the model has knowledge of past few words in the sentence. When it comes to practical implementation, RNN suffers from the problem of vanishing or exploding gradients for data inputs having long sequences. This problem is solved to an extent by LSTM (Long Short Term Memory) models and hence these are preferred over RNN for modeling sequencential data.

In the current project, parity has to be predicted (if the count of ones is odd ,output is 1 .It is 0 if the count of ones is even). Mathematically , the bitwise operation XOR between all the bits in data helps to output the parity. The LSTM model has to be trained to approximately implement this XOR functionality

## 2 Description of the datasets

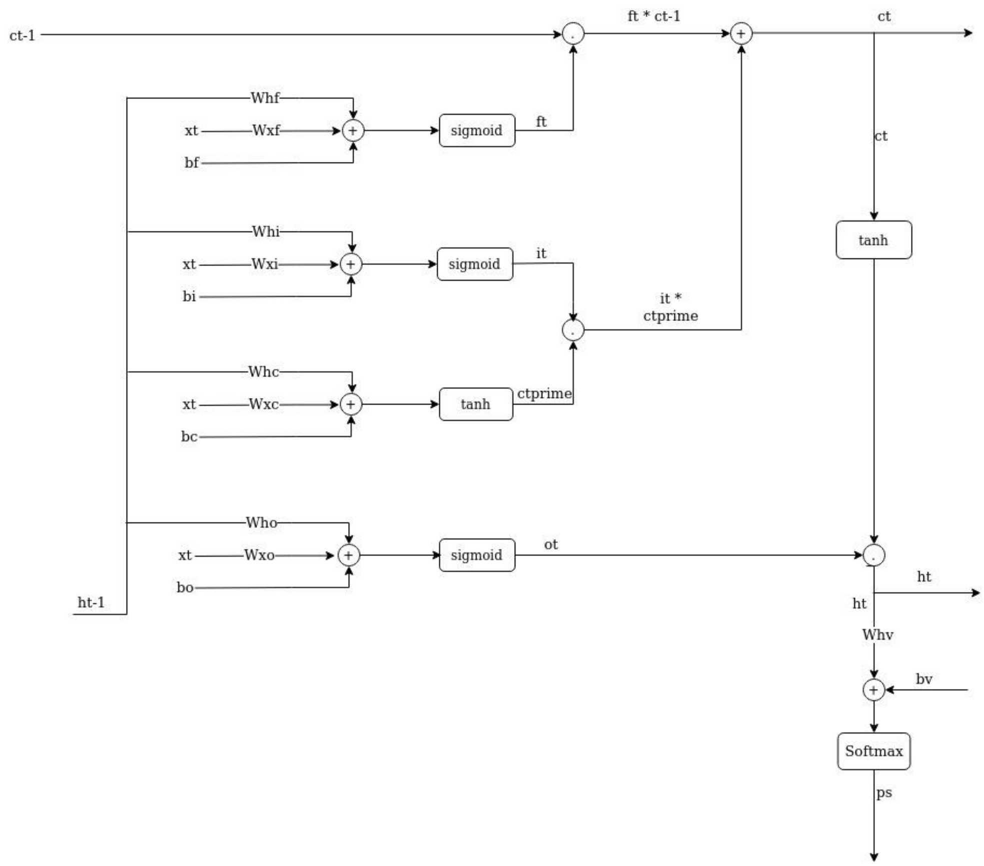
Two types of datasets are used as input to the model in 2 different instances of execution.

Type1 : 100000 binary strings with consistent length value 50 in both input and output. Type2 : 100000 binary strings where each data sample length can have any value between 1 to 50 in input and the corresponding output. In both these data sets , the output data bit at index i is the parity of the data bits in the input sample from index 0 to i-1. They are split into 80000 training samples and 20000 test samples.

Train dataset split into batches : The 80000 data samples in type1 are divided into 2500 batches each having size of 32. In type2 , 80000 samples are divided into 2521 batches each having variable size that is <=32. The reason for this variability is that the data in type 2 has variable lengths and the batches are divided in such a way that each batch has equal length data samples. This helped in easier implementation of Mini batch gradient descent using 5 different processes. Training and Testing are done separately with Type1 and Type2 batches .Results obtained are independent of each other

When Type1 dataset is used for program execution , each process handled 500 batches of data and in execution with Type2 dataset, 501 batches of data are handled by each process. The 2521th batch having 30 samples of data was not used for training . The train data is equally split among 5 processes and distribution of left over data(30 samples in 2521th batch) is not handled . An effective way is to have right multiple of processes distributing data equally or distribute the left over data equally among the fixed number of processes. This is left for future improvement work

# 3 Explanation of model architecture and implementation




The LSTM model has an LSTM cell which is repeated for 'n' times where n is the number of timesteps in the input data sample. The previous cell hidden state 'ht-1' and previous cell state 'ct-1' (cell state is current long-term memory of the network) are fed as input to the current cell along with the current time step data input

The above diagram shows the operations in one LSTM cell. It has 3 different gates 1.Forget gate(ft) 2.Input gate(it) 3.Output gate(ot) along with current cell state (ctprime).Each of these units have separate weights and biases associated with it . The weights are in dot product with the hidden state inputs and the input data for that particular time step . '+' symbol inside a circle is an addition operation while '.' symbol inside a circle is the pairwise multiplication operation. Outputs from these 4 units are passed through non linear operations such as tanh or sigmoid as shown in the diagram above

<u>Forget gate</u> : If the previous hidden state value (ht-1) and the current time step input (xt)are given as input , it outputs values which, when passed through the sigmoid function, gives the matrix values ft in the range [0,1] .ft*ct-1 operation helps in forgetting(making them zeros) few values and in preserving remaining values in ct-1. Forget gate's weights Wfh,Wfx and bias bf will be optimized so as to forget the right values of ct-1 to contribute to the correct output

<u>Input gate:</u> Current time step input (xt) and previous hidden state value (ht-1) are given as inputs to this gate ,uses weights Whi,Wxi and bias bi, the output value is passed through sigmoid function to give values 'it' in the range [0,1] .Current cell state (ctprime) is result of tanh operation of factor which is again the combination of the current time step input(xt) value and previous hidden state value(ht-1) using weights Whc,Wxc and bias bc .The use of ctprime

is to modify the information in xt given the ht-1 values . Tanh range is (-1,1), the impact of values close to lower bound is reduced while the impact of values closer to upper bound is remained as it is.

The operation it*ctprime nullifies the effect of few values in current cell state ( ctprime) and let other values pass through as they are . This is due to the usage of sigmoid non linear activation in input gate ‘it’.

ft*ct-1+it*ctprime operation produces the next cell state value ‘ct’. ct is group of values which are obtained by retaining and forgetting certain values from current cell state and previous cell state.

<u>Output gate:</u> Current time step input (xt) and previous hidden state value (ht-1) are given as inputs to this gate ,uses weights Who,Wxo and bias bo , the output values are passed through sigmoid function to give values ‘ot’ in the range [0,1] . Tanh operation on the next cell state(ct) constrains the values to the range (-1,1) i.e. reduces the impact of few values and while other values are passed on as they are before.

The operation ot*tanh(ct) filters out only the needed values from ct and gives them as next hidden state (ht) values. Thereafter, the ht values passed through a linear layer with weight Whv and bias bv and the resultant outcome is passed through a softmax layer to get the output probabilities.

The ‘n’ number of cells are cascaded one after another getting the values ct-1 and ht-1 from previous cell.

<u>Weights and bias initialization:</u> Based on references[1][2][3] and[4] , the weights which are in dot product with hidden state values from previous cell, in every gate and current cell state are obtained by ‘orthogonal initialization’. The number of units in hidden layer is 100 and dimension of these matrices are (100,100). The remaining eights are consequence of ‘xavier uniform’ initialization. Weights which multiply with input data have the dimension (batch size ,100) and the weight values in output linear layer have the dimension ( 100,2)

Except bias in output linear layer all other bias values have the shape (1,100) and the shape of bias in output layer is (1,2)

Forget gate bias values are set to ones and other bias values are set to zeros.

The input data and weights have rows which is equal to batch size but the bias values have single row . Addition of bias values was possible due to array broadcasting

<u>Training</u>: All batches of data are made available in every process . The start and end index of the group of batches for a process is calculated based on the rank of that process and in this way the group of batch data used differs from process to process. Mini batch gradient descent is used for optimization andtraining is done in parallel for each batch of data in the groups and the process can be split into 3 main steps a. Forward propagation b. Back propagation c. Optimization

a. Forward Propagation :

Before forward propagation , the weights and biases are made sure to be same across all the processes. MPI methods Send and Recv were used to get the latest parameters from process with rank 0 to the remaining processes. The previous cell state and previous hidden state values are initialized to zeros which would be used by forward propagation for 1$^{st}$ timestep.

The input and output data consists of two types of characters i.e. ‘0 and ‘1’ . For these 2 classes , softmax classification can be used which necessitates one hot encoding of the input data. Input character ‘0’ at a time step is encoded as [1,0] and character ‘1’ is encoded as [0,1]. The input and output data at every time step in the whole batch is encoded in this manner .

The same weight and bias values are used in the loop for n times where n is the number of time steps, to predict the output probabilities.

The input batch data of size (batch size, 2) is transformed into output probabilities of size (batch size,2) in the forward propagation

Cross entropy loss function is used to calculate the loss and the loss values of all the samples are summed up to form batch loss based on reference[6]

The output result values are returned as dict from the forward propagation function

Back Propagation : The gradients of weights and biases are declared in as class members as they have same dimension as their corresponding parameters. In the back propagation function ,at the beginning,gradients of biases are declared with row size equaling to the batch size. After the back prop is complete for a batch ,these 2 steps are followed, a.the gradients of biases are summed and assigned to bias gradients which are declared before as class members. b. The weight and bias gradients are sent from every process to process with rank 0. Here, all the gradients are reduced i.e. summed to form batch gradients that are used for optimization. Reference[7] was used to arrive on this idea of implementation

References [8],[9],[10] ,[11],[12] and figure above were used to derive and implement the back propagation

Optimization: Adam optimizer is implemented from scratch using reference [13] and except the learning rate, remaining hyper parameters for optimizer in the current implementation are same as in the Adam optimizer paper.

Prediction: Two types of output predictions are used .
a. One method is to check if the last bit of the predicted data matches with the last bit of the output test data simple
b. Another method is to check if every bit in the predicted data matches with every bit in the relevant time step of the output test data sample

Accuracy Calculation :
Accuracy = No. of correctly predicted data samples /No. of output data samples *100

# 4 Results

The batch loss values from all the processes are collected in process with rank 0 and averaged to get loss value for one iteration. The averaged batch loss from every iteration is summed up to get the loss for the epoch

Training and prediction for Fix length batch data :
Initially,when uniform random weights were used instead of Xavier uniform and orthogonal initialized weights , the loss value continued to increase for each epoch(Epoch:0 loss 1176 to epoch:20 loss 8843) .When the latter methods of weight initialization were used , loss decreased steadily . This shows the exploding gradients problem has occurred when uniform random weights are used. LSTM models do not solve the vanishing and exploding gradient problem of RNN completely ,rather they mitigate or delay this problem . Also, there exists at least one path in LSTM through which gradients can explode . This problem could have happened in the current implementation and resolved after the correct weight initialization.

Thereafter , it took 6 runs to achieve the convergence. In all these runs , the learning rate parameter and epochs are changed , other hyper parameters such as no. of units in hidden layer =100 , batch size =32 and random seed = 0 remain the same.

Run1:
Learning rate = 0.0005 , epochs =12
Loss does not decrease significantly even after 5 epochs
Training stopped

Run2:
Learning rate = 0.001 (same as in Adam optimization paper) ,epochs =12
Loss increases significantly for every epoch
Training stopped

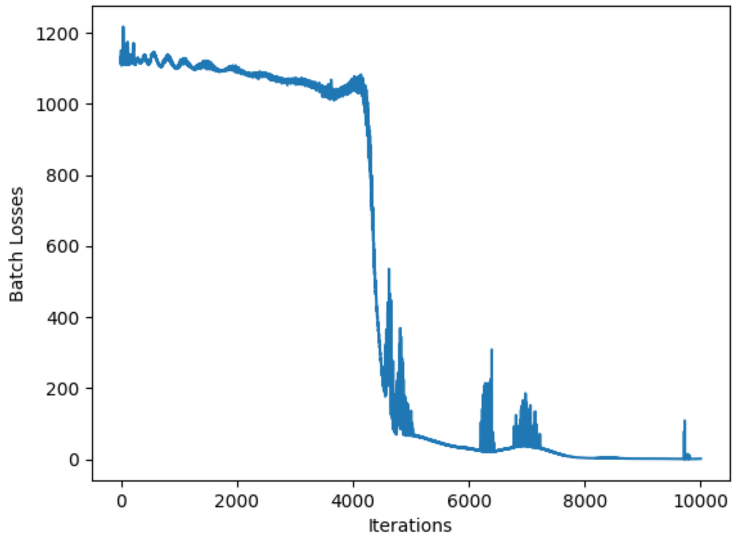
Run3:
Learning rate = 0.00075 ,epochs =12
Last epoch loss : 36
min batch Loss : 29.415
correct_prediction_count using last bit =19616
Prediction Accuracy using last bit =98.080000
correct_prediction_count for full length =7910
Prediction Accuracy for full length =39.550000

Run4:
Learning rate = 0.00080 ,epochs =12
Last epoch loss = 1780
min_batch_loss =1087.689
correct_prediction_count using last bit =9922
Prediction Accuracy using last bit =49.610000
correct_prediction_count for full length =0
Prediction Accuracy for full length =0.000000

Run5:
Learning rate = 0.00076 , epochs =12
Last epoch loss = 1960
min_batch_loss =110.529
correct_prediction_count using last bit =10195
Prediction Accuracy using last bit =50.975000
correct_prediction_count for full length =0
Prediction Accuracy for full length =0.000000

Run6 and Final result for fixed length data**:**
Learning rate = 0.00075 , epochs =20
Last epoch loss = 1
min_batch_loss = 1.075
correct_prediction_count using last bit = 19978
Prediction Accuracy using last bit = 99.890000
correct_prediction_count for full length = 18932
Prediction Accuracy for full length = 94.660000

Below graph is loss plot for Run 6 using number of iterations on x-axis
Number of iterations =10000 (500 values of averaged batch losses in 20 epochs)




Training and prediction for Variable length batch data :
Varying learning rates for the ranges of epochs instead of single learning rate helped in convergence . Loss curve and decrement seem to be complicated when compared to those of the fixed length data .
The number of epochs required are more in number compared to what is used for the fixed length data.
It took 27 runs to figure out the correct learning rates and no. of epochs to get prediction accuracy above 92% for variable length data.
Batch size =32 ,random seed =0 was used in all the runs below
Except for variation in handling data set, number of epochs and learning rates , no other change is made in the model for training variable length data set.

Run1:
Learning rate = 0.00075 , epochs =20
Training loss for Epoch:11 is 1909
Training stopped after 12 epochs due to high fluctuations in loss

Run2:
Learning rate = 0.00050 ,epochs =20
Training loss for Epoch:8 is 2359
Training stopped after 9 epochs due to high fluctuations in loss

Run 3:
Learning rate = 0.00040 , epochs =20
Training loss for Epoch:11 is 599
Training loss for Epoch:12 is 2280
Training stopped

Run 4:
Learning rate = 0.00035 , epochs = 20
Last epoch loss is 44
min_batch_loss :=4.391
correct_prediction_count using last bit =18878
Prediction Accuracy using last bit =94.390000

correct_prediction_count for full length =11162
Prediction Accuracy for full length =55.810000

Run 5:
Learning rate = 0.00035 ,epochs = 28
Last epoch loss is 464
min_batch_loss :=3.319
correct_prediction_count using last bit =16397
Prediction Accuracy using last bit =81.985000
correct_prediction_count for full length =9081
Prediction Accuracy for full length =45.405000

Run 6:
Learning rate = 0.00030 ,epochs = 20
Training loss for Epoch:16 is 301
Training loss for Epoch:17 is 434
Training stopped

Run 7:
Learning rate = 0.00025 ,epochs = 20
Last epoch loss is 479
min_batch_loss :=63.931
correct_prediction_count using last bit =10828
Prediction Accuracy using last bit =54.140000
correct_prediction_count for full length =2140
Prediction Accuracy for full length =10.700000

Run 8:
Learning rate =0.00035 ,epochs=30
Training loss for Epoch:29 is 2119
min_batch_loss :=3.319
correct_prediction_count using last bit =15151
Prediction Accuracy using last bit =75.755000
correct_prediction_count for full length =11532
Prediction Accuracy for full length =57.660000

Run 9:
Learning rate = 0.00034 ,epochs = 20
Training loss for Epoch:15 is 2000
Canceled after 16 epochs

Run 10:
Learning rate = 0.000349, epochs =20
Training loss for last epoch is 1138

Run 11:
Learning rate = 0.00035 for first 20 epochs and 0.00025 for next 10 epochs ,epochs =30
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 170
Last epoch loss is 213
min_batch_loss :=3.289
correct_prediction_count using last bit =18832
Prediction Accuracy using last bit =94.160000
correct_prediction_count for full length =13775
Prediction Accuracy for full length =68.875000

Run 12:
Learning rate = 0.00035 for first 20 epochs and 0.00012 for next 10 epochs , epochs =30
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 134
last epoch loss is 48
min_batch_loss :=3.199
correct_prediction_count using last bit =19350
Prediction Accuracy using last bit =96.750000
correct_prediction_count for full length =11238
Prediction Accuracy for full length =56.190000

Run 13:
Learning rate = 0.00035 for first 20 epochs and 0.00006 for next 10 epochs ,epochs =30
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 131
Last epoch loss is 16
min_batch_loss :=3.221
correct_prediction_count using last bit =19671
Prediction Accuracy using last bit =98.355000
correct_prediction_count for full length =13565
Prediction Accuracy for full length =67.825000

Run 14:
Learning rate = 0.00035 for first 20 epochs and 0.00010 for next 10 epochs , epochs =30
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 123
Last epoch loss is 23
min_batch_loss :=3.219
correct_prediction_count using last bit =19597
Prediction Accuracy using last bit =97.985000
correct_prediction_count for full length =12789
Prediction Accuracy for full length =63.945000

Run 15:
Learning rate = 0.00035 for first 20 epochs and 0.00009 for next 10 epochs , epochs =30
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 134
Last epoch loss is 22
min_batch_loss :=3.209
correct_prediction_count using last bit =19614
Prediction Accuracy using last bit =98.070000
correct_prediction_count for full length =12828
Prediction Accuracy for full length =64.140000

Run 16:
Learning rate = 0.00035 for first 20 epochs and 0.00001 for next 10 epochs , epochs =30
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 129
Training loss for Epoch:29 is 14
min_batch_loss :=3.182
correct_prediction_count using last bit =19702
Prediction Accuracy using last bit =98.510000
correct_prediction_count for full length =13592
Prediction Accuracy for full length =67.960000

Run 17:
Learning rate = 0.00035 for first 20 epochs and 0.000001 for next 10 epochs ,epochs =30
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 127
Training loss for Epoch:29 is 76
min_batch_loss :=3.930
correct_prediction_count using last bit =19238
Prediction Accuracy using last bit =96.190000
correct_prediction_count for full length =11661
Prediction Accuracy for full length =58.305000

Run 18:
Learning rate = 0.00035 for first 20 epochs and 0.00004 for next 10 epochs , epochs=30
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Training loss for Epoch:29 is 11
min_batch_loss :=2.628
correct_prediction_count using last bit =19760
Prediction Accuracy using last bit =98.800000
correct_prediction_count for full length =14664
Prediction Accuracy for full length =73.320000

Run 19:
Learning rate = 0.00035 for first 20 epochs and 0.00004 for next 31 epochs ,epochs =51
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Last epoch loss is 4
min_batch_loss :=0.836
correct_prediction_count using last bit =19894
Prediction Accuracy using last bit =99.470000
correct_prediction_count for full length =17535
Prediction Accuracy for full length =87.675000

Run 20:
Learning rate = 0.00035 for first 20 epochs and 0.00004 for next 45 epochs , epochs =65
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Last epoch loss is 50
min_batch_loss :=0.836
correct_prediction_count using last bit =19679
Prediction Accuracy using last bit =98.395000
correct_prediction_count for full length =16167
Prediction Accuracy for full length =80.835000

Run 21:
Learning rate = 0.00035 for first 20 epochs , 0.00004 from epoch no:20 to epoch no:45 and 0.000001 from epoch no:46 to epoch no: 54 , epochs = 55
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Training loss for Epoch:45 is 4
Training loss for Epoch:46 is 4
Last epoch loss is 4

min_batch_loss :=0.907
correct_prediction_count using last bit =19895
Prediction Accuracy using last bit =99.475000
correct_prediction_count for full length =17481
Prediction Accuracy for full length =87.405000

Run 22:
Learning rate = 0.00035 for first 20 epochs ,0.00004 for epoch no:21 to epoch no:45 epochs 0.00001 from epoch no:46 to epoch no: 59 ,epochs =60
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Training loss for Epoch:45 is 4
Training loss for Epoch:46 is 5
Last epoch loss is 4
min_batch_loss :=0.808
correct_prediction_count using last bit =19907
Prediction Accuracy using last bit =99.535000
correct_prediction_count for full length =17667
Prediction Accuracy for full length =88.335000

Run 23:
Learning rate = 0.00035 for first 20 epochs, 0.00004 from epoch no:20 to epoch no:45 and 0.000004 from epoch no:46 to epoch no: 64 ,epochs = 65
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Training loss for Epoch:45 is 4
Training loss for Epoch:46 is 4
Last epoch loss is 4
min_batch_loss :=0.826
correct_prediction_count using last bit =19900
Prediction Accuracy using last bit =99.500000
correct_prediction_count for full length =17643
Prediction Accuracy for full length =88.215000

Run 24:
Learning rate = 0.00035 for first 20 epochs , 0.00004 from epoch no:20 to epoch no:45, 0.00002 from epoch no:46 to epoch no: 64 ,epochs = 65
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Training loss for Epoch:45 is 4
Training loss for Epoch:46 is 5
Last epoch loss is 6
min_batch_loss :=0.741
correct_prediction_count using last bit =19890
Prediction Accuracy using last bit =99.450000
correct_prediction_count for full length =17863
Prediction Accuracy for full length =89.31500

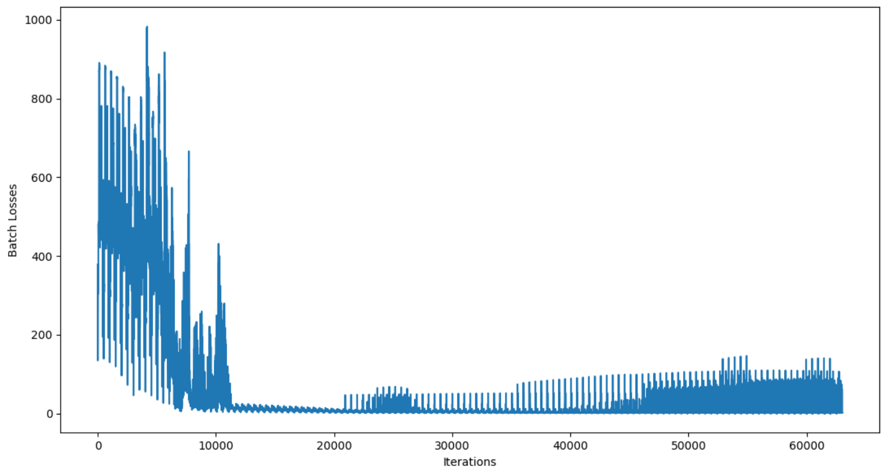
Run 25:
Learning rate = 0.00035 for first 20 epochs , 0.00004 from epoch no:21 to epoch no:45 and 0.000014 from epoch no:46 to epoch no: 64 ,epochs = 65
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Training loss for Epoch:45 is 4
Training loss for Epoch:46 is 5
Last epoch loss is 4

min_batch_loss :=0.795
correct_prediction_count using last bit =19903
Prediction Accuracy using last bit =99.515000
correct_prediction_count for full length =17753
Prediction Accuracy for full length =88.765000

Run 26:
Learning rate = 0.00035 for first 20 epochs , 0.00004 for epoch no:20 to epoch no: 45 and 0.00001 from epoch no:46 to epoch no: 79 ,epochs =80
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Training loss for Epoch:45 is 4
Training loss for Epoch:46 is 5
Training loss for last epoch is 3
min_batch_loss :=0.596
correct_prediction_count using last bit =19931
Prediction Accuracy using last bit =99.655000
correct_prediction_count for full length =18317
Prediction Accuracy for full length =91.585000

<u>Run 27 and Final epoch for variable length data:</u>
Learning rate = 0.00035 for first 20 epochs ,0.00004 for epoch no:20 to epoch no: 45 and 0.00001 from epoch no:46 to epoch no: 125 ,epochs = 125
Training loss for Epoch:19 is 44
Training loss for Epoch:20 is 132
Training loss for Epoch:45 is 4
Training loss for Epoch:46 is 5
Training loss for last epoch is 5
min_batch_loss :=0.405
correct_prediction_count using last bit =19938
Prediction Accuracy using last bit =99.690000
correct_prediction_count for full length =18838
Prediction Accuracy for full length =94.190000

Below graph is loss plot for Run 27 using number of iterations on x-axis
Number of iterations = 63000 (504 values of averaged batch losses in 125 epochs)




The point to note is that the epochs are increased from 80 to 125 and other hyper parameters are same when Run 26 and Run27 are compared . Final epoch loss is more in Run 27 by 1 but the accuracy is high by 2.6 %

## 5 Improvements and Conclusion

Following improvements can be further made which are left for future work

a. Tuning learning rate using learning rate scheduler perhaps can reduce the training time to get good results

b. MPI parallel programming with 5 processes reduced the training time of a single epoch to 14 min compared to 20 min training time of one epoch when MPI was not used. Using GPU based cloud infrastructure such as GCP or AWS can further reduce the training time to a large extent . This can help to train using efficient hyper parameter tuning methods

c. All experiments are run with seed 0 . Few other random seeds can be used in different trials.

d. Batch size 32 was used and other batch sizes such as 64,128 ,256 could be tried.

This project implementation helped to understand LSTM model and application of parallel programming using MPI

## References

[1]Orthogonal initialization for hidden to hidden , Xavier uniform initialization for rest of the weights
https://www.kaggle.com/code/junkoda/pytorch-lstm-with-tensorflow-like-initialization/notebook

[2]Weight initialization
https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/weight_initialization_activation_functions/

https://wandb.ai/sauravmaheshkar/initialization/reports/A-Gentle-Introduction-To-Weight-Initialization-for-Neural-Networks--Vmlldzo2ODExMTg

https://www.reddit.com/r/deeplearning/comments/9vjckm/what_is_the_best_way_to_initialize_lstm_weights/

[3] Orthogonal weight initialization
https://github.com/kastnerkyle/net/blob/d66d533b1ebcc25dd442bdb41431d334b617e80e/net.py#L168

https://github.com/nyu-dl/dl4mt-tutorial/blob/master/session3/lm.py

[4] Fan in Fan out for xavier uniform

https://stackoverflow.com/questions/42670274/how-to-calculate-fan-in-and-fan-out-in-xavier-initialization-for-neural-networks

https://ml-cheatsheet.readthedocs.io/en/latest/forwardpropagation.html

https://visualstudiomagazine.com/articles/2019/09/05/neural-network-glorot.aspx

[5] One hot encoding : https://datascience.stackexchange.com/questions/45803/should-we-use-only-one-hot-vector-for-lstm-input-outputs

[6]Batch Loss https://stackoverflow.com/questions/54053868/how-do-i-get-a-loss-per-epoch-and-not-per-batch

[7] Distibuted training and optimization

Diagram : https://github.com/vlimant/mpi_learn/blob/master/docs/downpour.png

MPI.AllReduce: .https://github.com/openai/baselines/blob/master/baselines/common/mpi_adam_optimizer.py

[8]Back propagation flow diagrams

https://towardsdatascience.com/backpropagation-in-rnn-explained-bdf853b4e1c2

https://github.com/christinakouridi/scratchML/blob/master/LSTM/LSTMForward_diagram.png

[9] Back propagation for batch

https://ai.stackexchange.com/questions/11667/is-back-propagation-applied-for-each-data-point-or-for-a-batch-of-data-points

[10] Modularizing code and RNN Implementation :

https://github.com/JY-Yoon/RNN-Implementation-using-NumPy/blob/master/RNN%20Implementation%20using%20NumPy.ipynb

[11] Back prop chain rule, matrix multiply dimension matching and Matrix -Matrix Multiply gradient : https://cs231n.github.io/optimization-2/

[12] Adam Optimizer

https://towardsdatascience.com/how-to-implement-an-adam-optimizer-from-scratch-76e7b217f1cc

https://stackoverflow.com/questions/70225531/time-step-in-adam-optimizer

https://arxiv.org/pdf/1412.6980.pdf

[13]Vanishing and exploding gradients in LSTM ( This problem is not entirely solved in LSTM ,rather it is mitigated and delayed)

https://www.quora.com/How-does-LSTM-help-prevent-the-vanishing-and-exploding-gradient-problem-in-a-recurrent-neural-network

[14] Cross Entropy https://www.quora.com/Whats-an-intuitive-way-to-think-of-cross-entropy

[15] Matrix and batch dimensions: https://www.quora.com/In-LSTM-how-do-you-figure-out-what-size-the-weights-are-supposed-to-be

[16] Bias gradients sum along axis =0 : https://github.com/Erlemar/cs231n_self/blob/master/assignment2/cs231n/layers.py#L116

[17] RNN Theory : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[18] LSTM Theory : http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[19] LSTM theory explanation : https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

[20] Batch size : https://ai.stackexchange.com/questions/8560/how-do-i-choose-the-optimal-batch-size

[21] Bucketing into batches of data for irregular length sequences: https://rashmi-margani.medium.com/how-to-speed-up-the-training-of-the-sequence-model-using-bucketing-techniques-9e302b0fd976

[22] Python binary string generation https://www.geeksforgeeks.org/python-program-to-generate-random-binary-string/

[23] Gradient clipping

https://stackoverflow.com/questions/44796793/difference-between-tf-clip-by-value-and-tf-clip-by-global-norm-for-rnns-and-how