Parts of Speech Tagging using Long Short Term Memory

A
Mini Project Report

by

Amitesh Sharma(SC18M004)

Department of Mathematics
Indian Institute of Space Science and Technology, Trivandrum
Semester-II

Contents

List of Figures

1 Introduction

A part-of-speech tag is a tag label which is associated with each word that marks the different part-of-speech of words and also indicates the tense, quantity ,etc. These tags are often related to search based approach in corpus which are often used to deduce some inference using tools for analysis.
Tagset is the set of all tags in the corpus which can be used for detailed description of part-of-speech including quantity, tenses, conjugations related to verbs,etc. Tagsets are usually non-identical for non-identical languages. Based on the language, same word may have non-identical part of speech which is denoted by different tags.
Various methods to solve POS tagging problem:

1. Lexical Based Methods:This is count based based method to determine the tag.

2. Rule-Based Methods: This method works on rules like decision tree and assigns the tag using the rules.

3. Probabilistic Methods: This method works by finding the probability of sequence of tag and then the tags are determined based on probability values . Conditional Random Fields and Hidden Markov Models are examples where these probabilistic approaches are applied.

4. Deep Learning Methods: POS tagging problem can be solved by RNN as the data is sequential and the network can be used for multiclass classification with tags as classes.

2 Theory

Sequence related problems are not similar to traditional ML problems. Here, the model requires selective past information and RNN based models which have memory can learn these time based dependencies. RNN based models have loops and thus information can be sent across time depending on the capacity of the model.
To solve sequential data related problems, a feedforward network can be used if entire sequence is given. However, input size of data will have to be set beforehand which will limit the capacity of the network. So, RNN and its variants will be better suited for the problem. The data from one state is passed to several time states ahead. If the distance between the time steps is less, RNN is able to learn about the dependencies. However, if the distance between the states is large, RNN is not able to learn as it is affected by exploding gradient and vanishing gradient problems.
Long Short Term Memory networks was designed especially to avoid the long term dependency problem. Their nature is to remember information for large intervals of time and are not affected by exploding gradient and vanishing gradient problems.

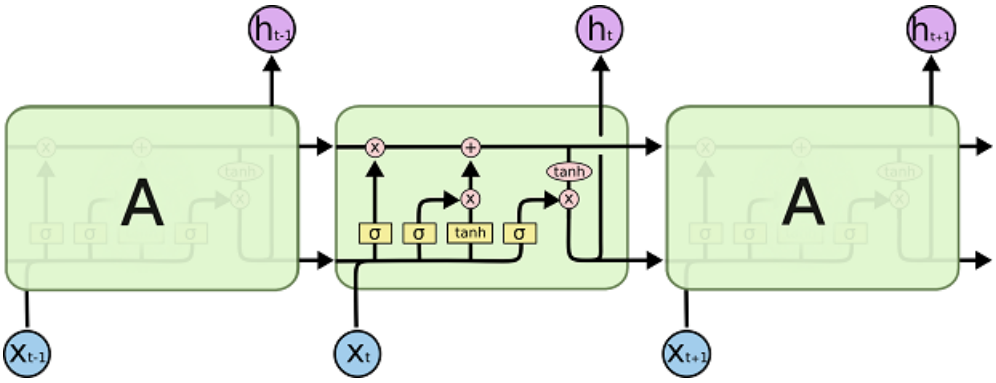

Figure 1: LSTM Architecture

The cell state is most critical to LSTM which is represented by the horizontal line and it goes through the entire network without changing the information. Initial step in LSTM decides how much information the network needs to forget. A sigmoid layer called the "forget gate layer" is used for this.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

The next step decides how much new information needs to be selectively written from the current cell state. A sigmoid layer called the "input gate layer" decides the update factor.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

A tanh layer generates a vector of new state values, $\widetilde{C_t}$, that will be added to the state. These two are combined to create an update to the state.

$$\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The previous cell state C_{t-1} is updated to current state C_t.
The previous state is multiplied by the forget factor f_t. Then $i_t * \widetilde{C_t}$ is added to get new current state values.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}$$

This output is dependent on cell state. First, a sigmoid layer decides how much amount of cell state should be present in the output. Then, the cell state is passed through tanh and multiplied by the output of the sigmoid gate.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_0)$$

$$h_t = o_t * tanh(C_t)$$

3 Implementation details

Dataset:
Penn Tree Bank Dataset and Brown Dataset is used for the problem which contains contains 3913 sentences labelled with 36 tags .
Word Embeddings:

ML and DL algorithms cannot process strings or text directly. The input needs to be converted to numerical interpretation to perform problems related to classification, regression, etc. The amount is humongous in the raw form. Thus to capture the semantic and syntactic interconnections based on contextual inference, Word embeddings are used. These are numerical representations of the raw text document that can be passed through the network.

Preprocessing steps:

- Making index for words and tags
- Adding '<UNK>','<EOS>'
- Making reverse indexes
- Converting words in sentences to number
- Mapping numbers and words
- Mapping numbers and tags
- 70%-30% Train Test split

Hyperparameters:

- Learning Rate
- Hidden layer size
- Embedding dimension

Negative Log-Likelihood (NLL) Loss:

A loss function determines the factor by which one can say how well the model is performing with respect to the data. Loss value is high if predictions are completely incorrect or far from the actual answer. If predictions are close to the actual answer, loss function has low value. As hyperparameters are tuned, algorithm try and improve the model and loss function will guide whether the algorithm is learning in the right direction.

NLL Loss function:

$L(y) = -log(y)$

This is summed for all the correct classes and taken average which gives the total loss.

Optimizer:

Stochastic gradient descent uses a single sample per iteration and as it converges, we get the optimal loss value. Stochastic gradient descent is noisy but works well if number of iterations is high. Data sample is chosen randomly for training which is passed as a batch(size is one). SGD can also be used with mini batches. In this project, SGD is used as optimizer to minimize the loss. Alternatively, Adam optimizer can be used.

LSTM and its Variants:

POS tagger module is trained using:

1. Standard implementation of LSTM cell
2. LSTM-f : implementation of LSTM cell without forget gate
   Modified Equations:
   $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
   $\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
   $C_t = i_t * \widetilde{C_t}$
   $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_0)$
   $h_t = o_t * tanh(C_t)$
3. LSTM-o : implementation of LSTM cell without output gate
   Modified Equations:
   $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
   $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
   $\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
   $C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}$
   $h_t = tanh(C_t)$
4. LSTM-i: implementation of LSTM cell without input gate
   Modified Equations:
   $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
   $\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
   $C_t = f_t * C_{t-1}$
   $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_0)$
   $h_t = o_t * tanh(C_t)$
5. LSTM-b : implementation of LSTM cell with forget gate bias set to 1
   $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
   $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + tensor(ones))$
   $\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
   $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_0)$
   $h_t = o_t * tanh(C_t)$

4 Results

Penn Tree Bank Dataset:

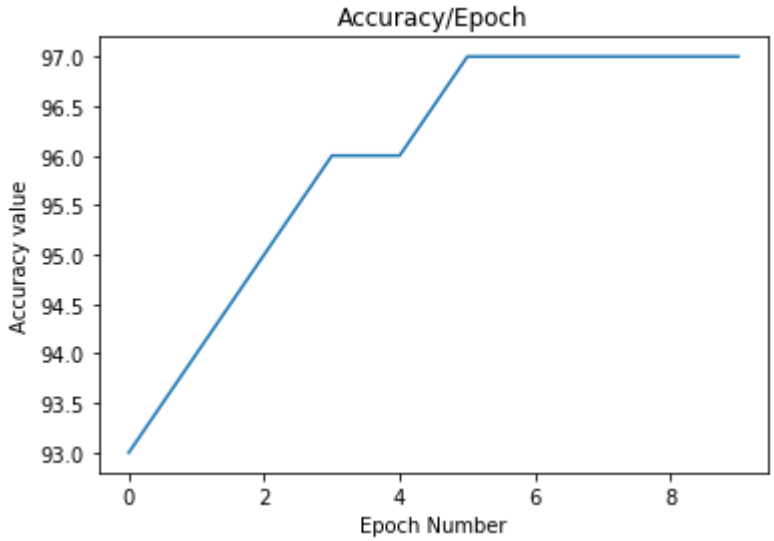

Figure 2: Standard LSTM

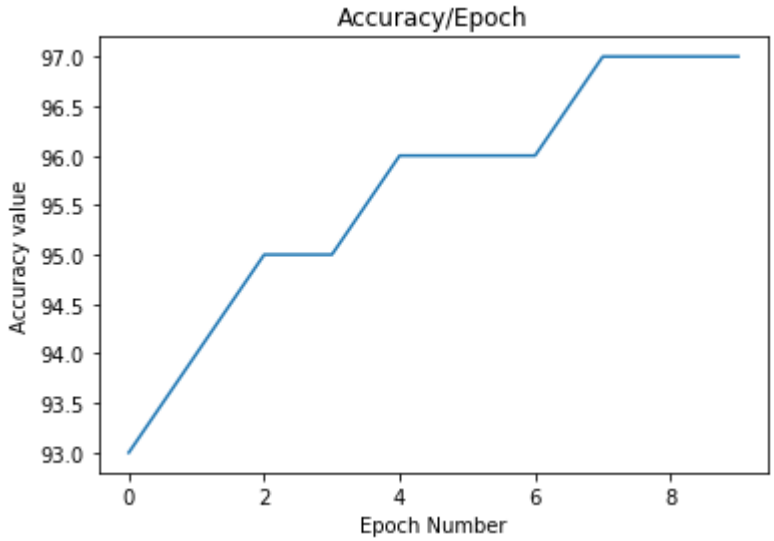

Figure 3: LSTM cell without input gate

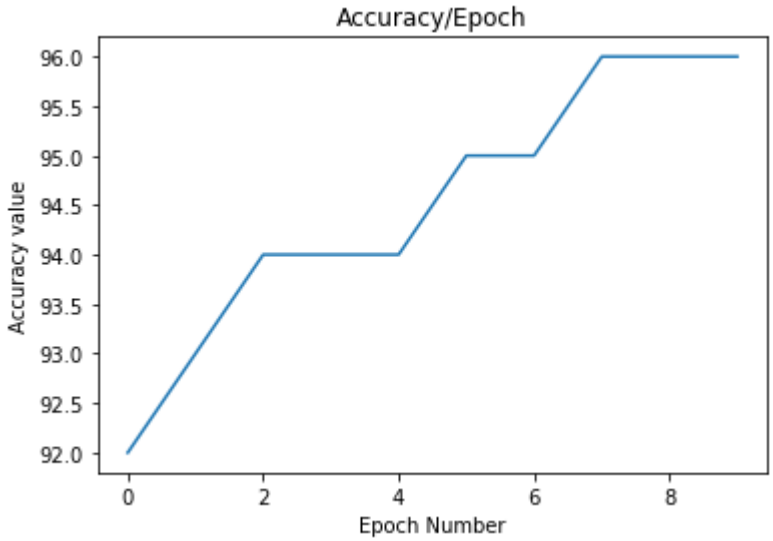

Figure 4: LSTM cell without output gate

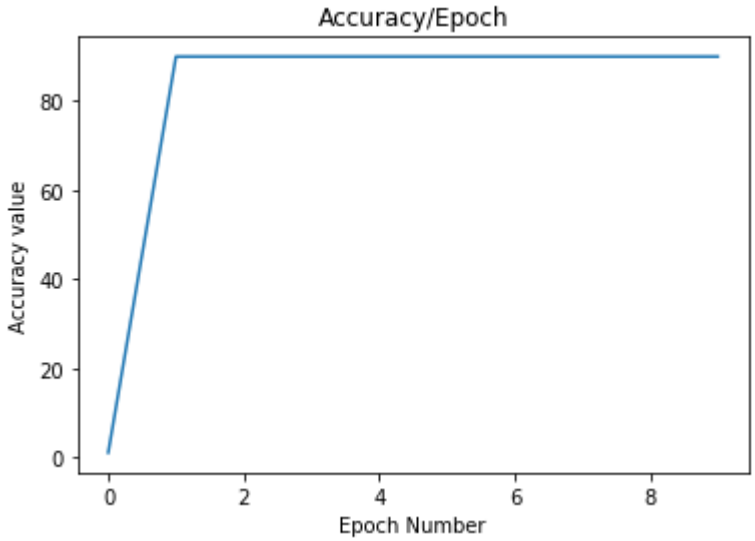

Figure 5: LSTM cell without forget gate

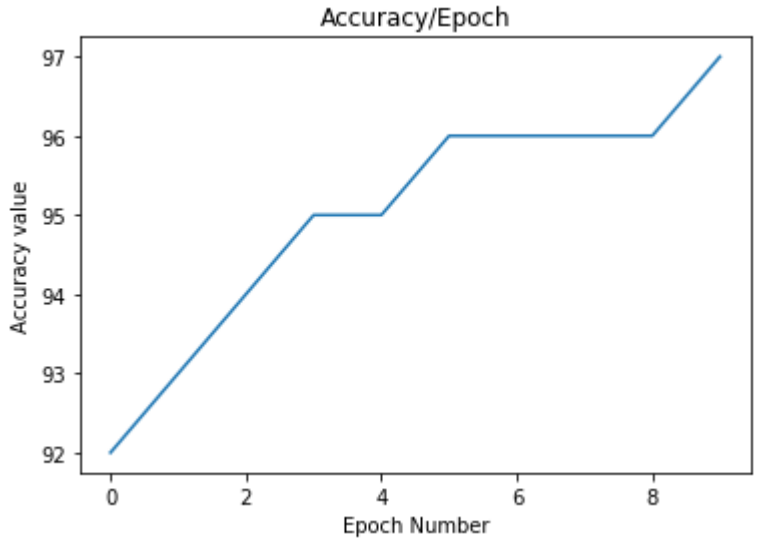

Figure 6: LSTM cell with forget gate bias set to 1

Brown Dataset:

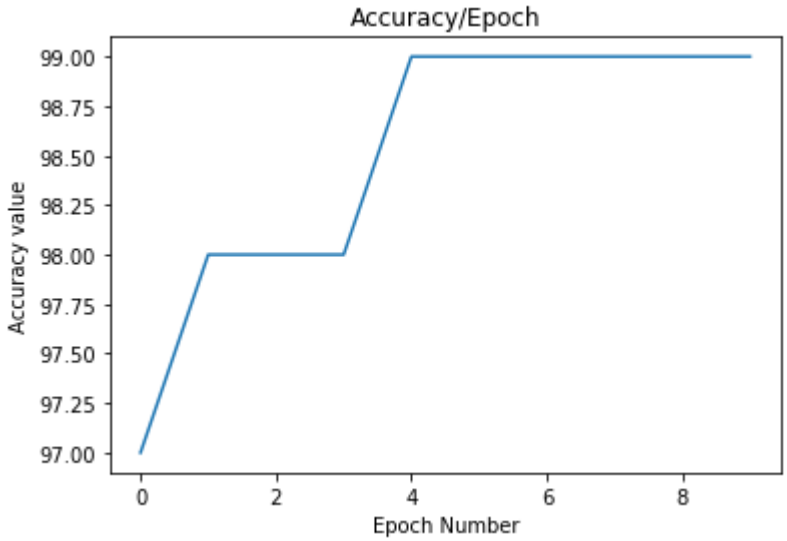

Figure 7: Standard LSTM

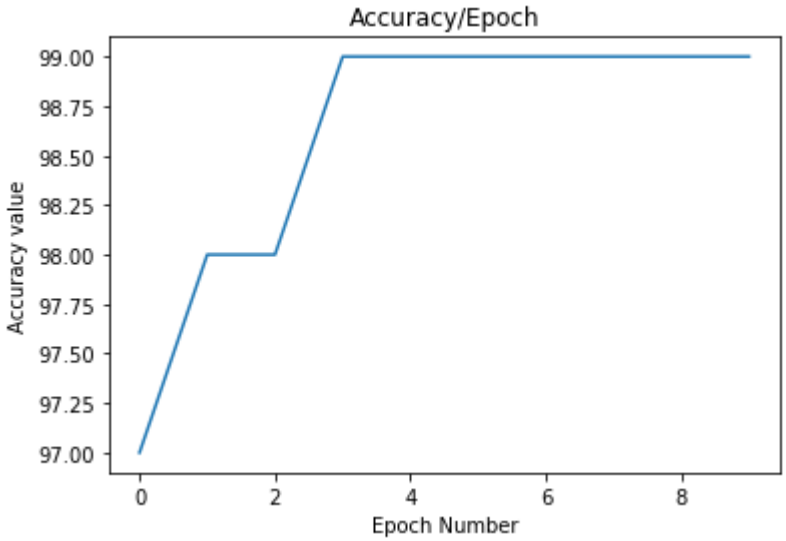

Figure 8: LSTM cell without input gate

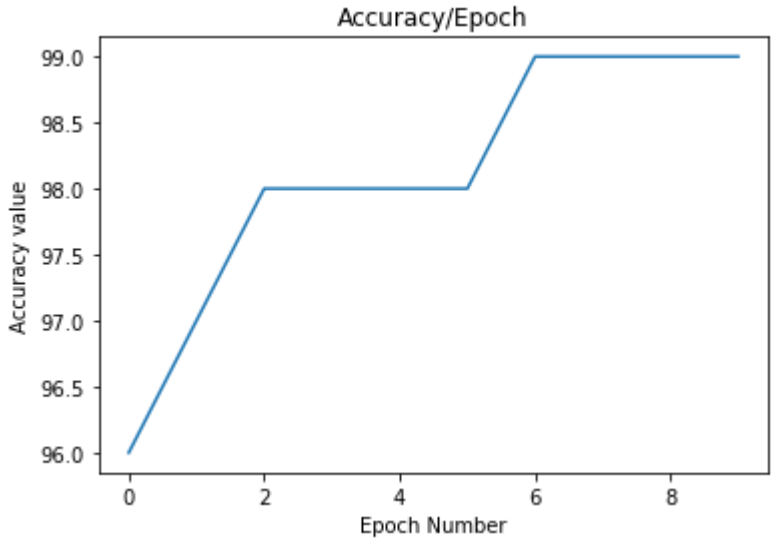

Figure 9: LSTM cell without output gate

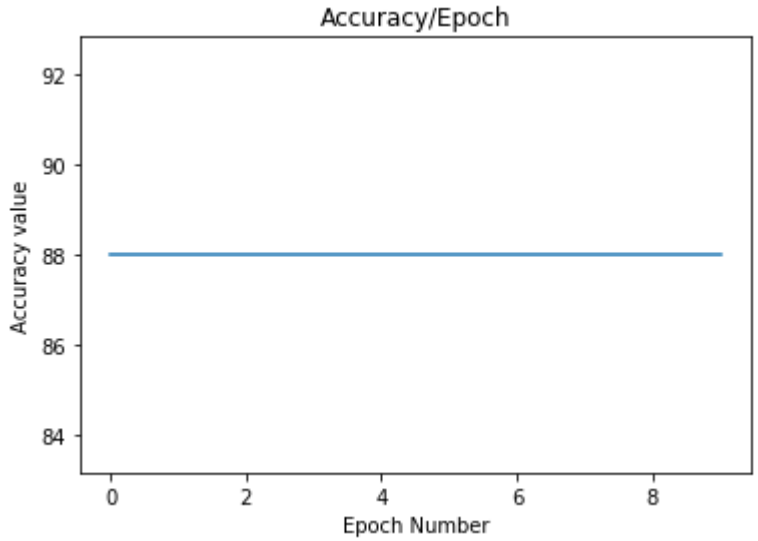

Figure 10: LSTM cell without forget gate

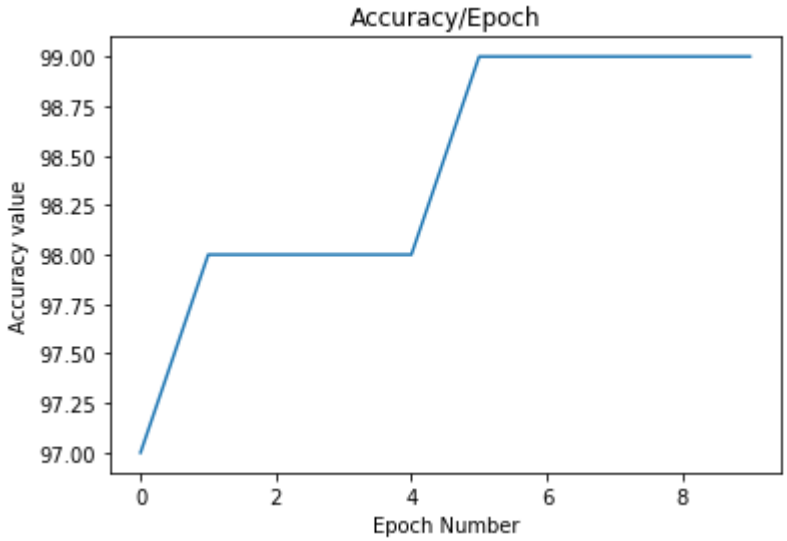

Figure 11: LSTM cell with forget gate bias set to 1

Comparative Results:

	Performance in percentage	
Model	Brown dataset	TreeBank dataset
GRU-Standard	95	91
GRU-M1	88	93
GRU-M2	95	95
GRU-M3	96	96
LSTM-Standard	99	97
LSTM-o	99	97
LSTM-f	88	86
LSTM-i	99	96
LSTM-b	99	97
RNN	80	85

Figure 12: Comparison Results

5 Conclusion

Standard implementation of LSTM cell ,implementation of LSTM cell without input gate and LSTM cell with forget gate bias set to 1 give 97 percent accuracy after 10 epochs. LSTM cell without forget gate gives the worst result as accuracy reaches 85 percent at first epoch and stays the same.

References

[1] An Empirical Exploration of Recurrent Network Architectures : Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever ; Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:2342-2350, 2015.

[2] CS7015: Deep Learning Lecture by Mitesh M Khapra (IIT M) https://www.cse.iitm.ac.in/~miteshk/CS7015.html

[3] PyTorch Documentation https://pytorch.org/docs/stable/torch.html