**Name:** Kenny Patel

**Roll No.** : 19074010

# Lab Task-4: Part-of-Speech (POS) Tagging, Chunking and Named Entity Recognition (NER) using RNN and Bi-LSTM

## Chunking

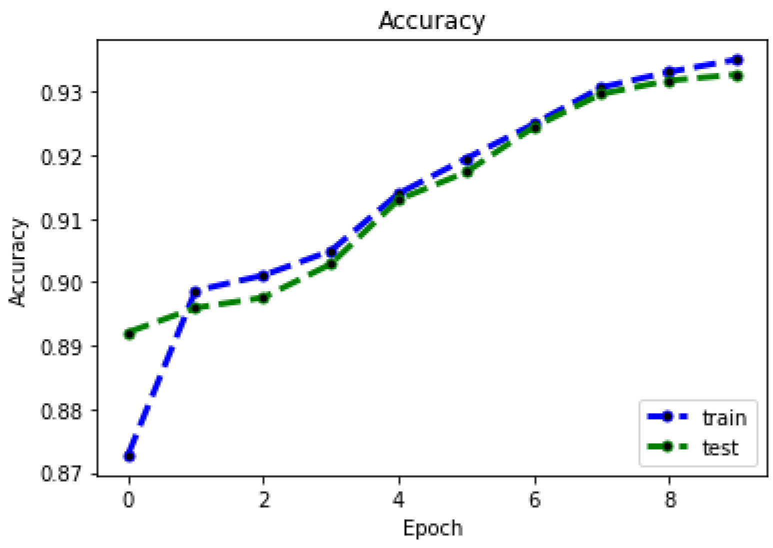
Chunking is a process of extracting phrases from unstructured text. Chunking works on top of POS tagging, it uses pos-tags as input and provides chunks as output. Similar to POS tags, there are a standard set of Chunk tags like Noun Phrase(NP), Verb Phrase (VP), etc. Chunking is very important when you want to extract information from text such as Locations, Person Names, etc. In NLP called Named Entity Extraction.

- Corpus: Training and testing data is downloaded from CoNLL shared tasks 2000.
- Data Preprocessing: Corpus consisted of three columns in which the first column was of Words and the third column, of their chunking tags. I iterated through the data, stored the first column of each row in X and the third column of each row in Y, and split it at each full stop to separate them into sentences, and stored those sentences in another list. Hence, forming a list of lists.
    - Steps:
        - Count the vocabulary(unique words in the dataset): 17257

        - Count unique tags in the dataset: 22
        - As RNN or Bi-LSTM can't take up words as input directly, they need to be vectorized. Hence, I tokenized each string in both the arrays X and Y.
        - Find out the length of the longest sequence in X: 77
        - Pad the sequences: we set a particular sentence length 80(max_length); sentences longer than max_length - truncated from the back; sentences shorter than max_length padded with zeros in front.
        - assign padded sequences to X and Y
        - use Keras' to_categorical function to one-hot encode Y
        - split the data into training and testing data
        - convert them into NumPy arrays

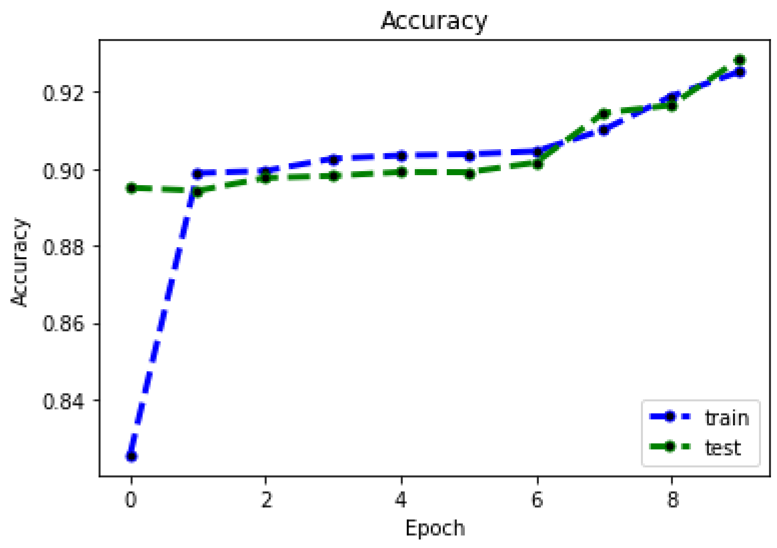
## Chunking using RNN:

- Training :
    - A model is generated by adding the embedding layer initially followed by several SimpleRNN layers and finally adding time distributed layer.
    - Compile the model
    - Train the model
    - Save the model
- Accuracy: 93.27%
- Validation_Accuracy : 93.27%
- Plot: I have plotted the accuracy of training and testing data at subsequent epochs:



## Chunking using Bi-LSTM:

- Training :
    - A model is generated by adding the embedding layer initially followed by several Bidirectional LSTM layers and finally adding time distributed layer.
    - Compile the model
    - Train the model
    - Save the model
- Accuracy : 92.5%
- Validation_Accuracy : 92.8%
- Plot: I have plotted the accuracy of training and testing data at subsequent epochs:



## POS Tagging

POS or part-of-speech tagging is the technique of assigning special labels to each token in text, to indicate its part of speech, and usually even other grammatical connotations, which can later be used in text analysis algorithms. For example, for the sentence -

*She is reading a book.*

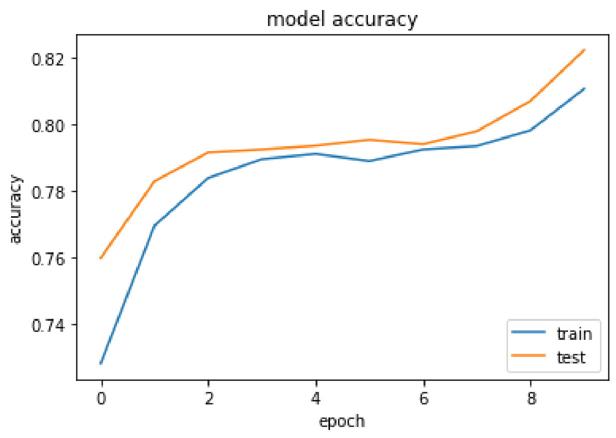
‘She’ would the POS tag of pronoun, ‘is’ would get an article tag, ‘reading’ a verb tag, ‘a’ would get an article tag and ‘book’ would get a noun tag. We can then do a search for all verbs which would pull up the word reading, and also use these tags in other algorithms.

- Corpus: Training and testing data is downloaded from CoNLL shared tasks 2000.
- Data Preprocessing : Corpus consisted of two columns in which the first column is of the Words and the second, of their POS tags. I iterated through the data, stored the first column of each row in X and the second column of each row in Y, and split it at each full stop to separate them into sentences, and stored those sentences in another list. Hence, forming a list of lists.

- Steps:
    - Count the vocabulary(unique words in the dataset) : 17258
    - Count unique tags in the dataset: 45
    - As RNN or Bi-LSTM can't take up words as input directly, they need to be vectorized. Hence, I tokenized each string in both the arrays X and Y.
    - Find out the length of the longest sequence in X: 130
    - Pad the sequences: we set a particular sentence length 100(max_length); sentences longer than max_length - truncated from the back; sentences shorter than max_length - padded with zeros in front.
    - assign padded sequences to X and Y
    - use Keras' to_categorical function to one-hot encode Y
    - split the data into training and testing data
    - convert them into NumPy arrays

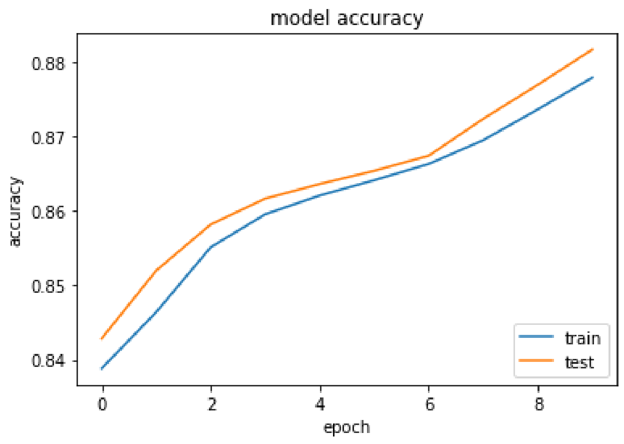
## POS-tagging using RNN:

- Training :
    - A model is generated by adding the embedding layer initially followed by several SimpleRNN layers and finally adding time distributed layer.
    - Compile the model
    - Train the model
    - Save the model
- Accuracy: 81.05%
- Validation_Accuracy : 82.2%
- Plot: I have plotted the accuracy of training and testing data at subsequent epochs:



## POS-tagging using Bi-LSTM:

- Training :
  - A model is generated by adding the embedding layer initially followed by several Bidirectional LSTM layers and finally adding time distributed layer.
  - Compile the model
  - Train the model
  - Save the model
- Accuracy: 87.7%
- Validation_Accuracy : 88.1%
- Plot: I have plotted the accuracy of training and testing data at subsequent epochs:



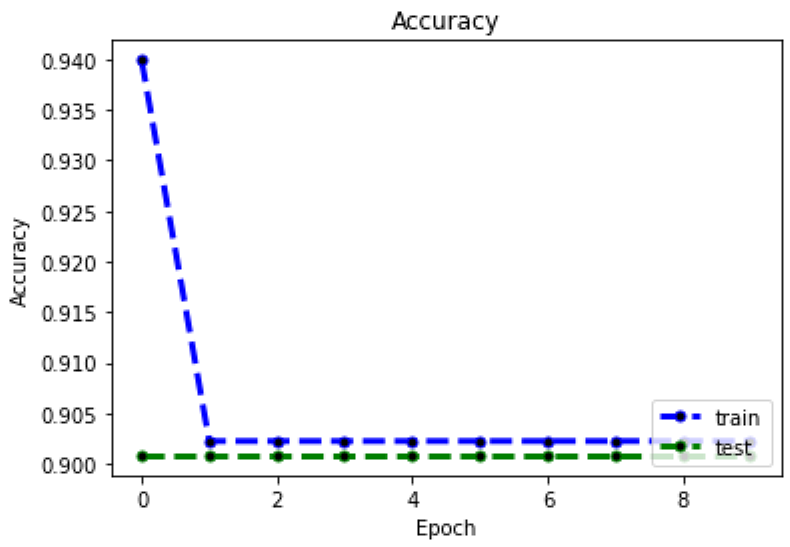
## Named Entity Recognition (NER)

Named Entity Recognition (NER) is an important basic tool in the fields of information extraction, question answering system, parsing, machine translation, and so on. It plays an important role in the practical process of natural language processing technology.

- Corpus: Training and testing data is downloaded from CoNLL shared tasks 2003.
- Data Preprocessing : Corpus consisted of four columns in which the first column is of the Words and the last, of their NER tags. I iterated through the data, stored the first column of each row in X and the last column of each row in Y, and split it at each full stop to separate them into sentences, and stored those sentences in another list. Hence, forming a list of lists.
    - Steps:
        - Count the vocabulary(unique words in the dataset): 21009
        - Count unique tags in the dataset: 8
        - As RNN or Bi-LSTM can't take up words as input directly, they need to be vectorized. Hence, I tokenized each string in both the arrays X and Y.
        - Find out the length of the longest sequence in X: 113
        - Pad the sequences: we set a particular sentence length 90(max_length); sentences longer than max_length - truncated from the back; sentences shorter than max_length - padded with zeros in front.
        - assign padded sequences to X and Y
        - use Keras' to_categorical function to one-hot encode Y
        - split the data into training and testing data
        - convert them into NumPy arrays

## NER using RNN:

- Training :
  - A model is generated by adding the embedding layer initially followed by several SimpleRNN layers and finally adding time distributed layer.
  - Compile the model
  - Train the model
  - Save the model
- Accuracy : 90.2%
- Validation_Accuracy : 90.08%
- Plot: I have plotted the accuracy of training and testing data at subsequent epochs:

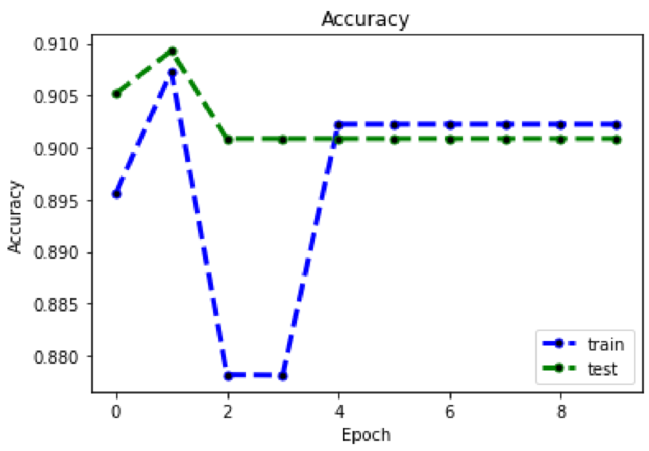


## NER using Bi-LSTM:

- Training :
  - A model is generated by adding the embedding layer initially followed by several Bidirectional LSTM layers and finally adding time distributed layer.

- Compile the model
- Train the model
- Save the model

- Accuracy : 90.2%
- Validation_Accuracy : 90.08%
- Plot: I have plotted the accuracy of training and testing data at subsequent epochs:



## Conclusion:

- RNN uses outputs of previous layers as the input of upcoming layers while Bi-LSTM consists of two LSTMs, one taking input in forward direction and the other in the backward direction, hence effectively increasing the amount of information available to the network and thereby increasing accuracy.
- Since Bi-LSTM has double LSTM, it is costly and takes up more time for training.
- CRF++ compared to RNN or Bi-LSTM gives better accuracy but takes up more time for training.