**NLP Assignment-2**
**POS Tagger**

**Task:**
The task is to provide a sequence of tags for a given sequence of words. This can be considered as a sequence modelling task.

Complete Score of model:
**Accuracy =99.65**

Hyperparameters used:
**Loss = 'categorical crossentropy'**
**Model used is sequential, bidirectional**
**Activation function used is Softmax**
**Optimizer = Adam**
**Metrics = Accuracy**

**Approach**
We will input a sentence word by word into the LSTM. We expect the LSTM to make a prediction for the tag of the word at each time step for each word.

We use a **Bidirectional LSTM layer :**
Bidirectional LSTMs (BLSTMs) were introduced to increase the amount of input information available to the network. For example, multilayer perceptrons have limitations on the input data flexibility, as they require their input data to be fixed. Standard recurrent neural network (RNNs) also have restrictions as the future input information cannot be reached from the current state. On the contrary, Bidirectional LSTMs do not require their input data to be fixed. Moreover, their future input information is reachable from the current state.
The basic idea of BLSTMs is to connect two hidden layers of opposite directions to the same output.

We substitute a bidirectional LSTM for a unidirectional one in our model's implementation. At its core, LSTM uses the hidden state to preserve data from inputs that have already been processed. Due to the fact that it has only ever received inputs from the past, a unidirectional LSTM can only preserve information from the past.
When using bidirectional, your inputs will be processed in two different directions: one from the present to the future and the other from the future to the present. This method differs from unidirectional in that information from the future is preserved in the LSTM that runs backwards, and by combining the two hidden states, you can preserve data from both the present and the future at any given time. They are able to approach a unit, therefore

**Padding the data**

Since all the sentences aren't of the same length, it creates a lot of variability. This creates a staggered matrix/tensor which isn't good for fast training in the TensorFlow paradigm. Therefore, we had to pad all the words with zeros in the beginning.
For example:
X_train : [ John, is, running ]
X_train_numberised: [ 23, 75, 234]
X_train_padded: [0, 0, 0, 0, 0, …, 0, 23, 75, 234]

**Converting tags to categorical form**
The neural network that we designed provides an output probability for each tag. For example, given the sequence: [ 'John', 'is', 'running'] It will predict the probabilities for all the possible tags for each word. So, for 'John', it will make a prediction: [ .2 .2 0.1 0.1 0.4] of the probabilities. This can be read as 20% probability of NNP 20% probability of VBX . . . Here NNP will always be index 0, VBX index 1, etc