CS6030 NATURAL LANGUAGE PROCESSING

ASSIGNMENT 1 – 27/10/2021

Developing a POS Tagger

Name: Reuel Samuel Sam
Registration Number: 2018103053
Batch: P

NOTE: Link to Github Page: https://github.com/ReuelSam/NLP-Assignment-1-POSTagger

Abstract – Natural Language Processing (NLP) is a subfield of linguistics, computer science and artificial intelligence concerned with the interactions between computers and human language. This field focuses on processing and analyzing large amounts of natural language data. Part-of-the-speech (POS) Tagging is a prominent Natural Language Processing approach that refers to categorizing words in a text (corpus) in accordance with a specific part of speech, based on the definition of the word and its context. This assignment focuses on the development of a POS-Tagger that assigns the universal tagset to the given input sentence. The model has been trained on NLTK's Brown Treebank Corpus that contains 57340 sentences. The proposed model has achieved a testing accuracy of 99.50%.

I. PROBLEM STATEMENT AND OVERVIEW

This assignment aims to tackle the NLP problem regarding the categorizing of words in a corpus in correspondence with a particular part of speech depending on the definition of the word and its context. This assignment deals with the Universal tagset of POS tags used in Universal Dependencies (UD). The model proposed for this classification uses an Embedding and LSTM layer along with 3 Dense layers. Apart from the model, due to the nature of text data, for analysis by a computer, a need for preprocessing the data was identified. This has been done by tokenizing the sentences after being untagged from the datasets. Due to the difference in length in the sentence, the sentence have been padded to have equal length.

II. DATASET DETAILS

The dataset used for the training of the dataset has been taken from NLTK's rich library – The Brown Corpus. The Brown Corpus was the first million-word electronic corpus of English, created in 1961 at Brown University. This corpus contains text from 500 sources, and the sources have been categorized by genre [1]. This dataset contains many different forms of data, however, for the purposes of this assignment only the tagged sentences have been extracted with the aforementioned Universal Tagset.

The annotation scheme is based on an evolution of (universal) Stanford dependencies (de Marneffe et al., 2006, 2008, 2014) [2], Google universal part-of-speech tags (Petrov et al., 2012) [3] and the Reusable Tagset Conversion Using Tagset Drivers (Zeman, 2008) [4]. Under this, there are 12 Tags that have been identified:

- ADJ: adjective
- ADP: adposition
- ADV: adverb
- VERB: verb
- CONJ: conjunction
- DET: determiner
- NUM: numeral
- NOUN: noun
- PRT: particle
- PRON: pronoun
- . : punctuation
- x : other

There are a total of 57340 sentences in this dataset. These have been split into train, test and validation sets of sizes 41428, 8601 and 7311 respectively. The tagged sentences are provided as a list containing tuple pairs of the word and their corresponding tags. Therefore, these sentences need to be untagged first into just the words to be used as the input and the Tags to be used as the expected output. These inputs and expected output tags are then vectorized and the output tags are then alone one hot encoded to be readable by the model. The data is padded as well to avoid issues regarding shape of input due to varying length of sentences.

III. MODULES

Python comes with a rich library of modules that make coding rather convenient. This is no different when it comes to Language Processing. Python has plenty of Language Processing that can be made use of. Following are the modules that have been imported and applied in this assignment:

- ⇨ *Warnings:* Python library to warn regarding depreciation. Used here to avoid unnecessary prints in the model creation.
- ⇨ *TensorFlow:* Open Source Machine Learning Platform
- ⇨ *Keras:* Open Source Library that provides python interface for artificial neural networks

- *Matplotlib:* Graphical Library
- *Seaborn:* Graphical Library
- *SKLearn:* Machine Library
- *Numpy:* Mathematical Library that supports array operations
- *NLTK:* NLP Library for symbolic and statistical processing for English
- *PPrint:* Python Library for pretty-printing structures in an interpretable format

Apart from Python modules, the code can be broken down into a set of modules:

- **Imports:** This section of the code is meant for processing all the imports of the necessary libraries for the execution of this project

- **Loading of Dataset:** The dataset has been loaded from Python's NLTK's collection of Treebanks. As explained in Section 2, this assignment makes use of the Brown Corpus.

- **Preprocessing of Data:** Being textual data, plenty of preprocessing is required for this dataset. The preprocessing can be further broken down into more subdivisions as follows:
    - *Untagging:* Splitting input data into raw sentences and tags rather than keeping them together.
    - *Tokenizing:* Since computers cannot process text data as it is, the sentences and tags need to be tokenized, as in converted to numbers, for it to be processed. This is done by mapping the words in a dictionary and replacing the words with the mapped key. SKLearn's tokenizer was used to achieve this.
    - *Padding:* For machine learning models, it is preferred to have equally sized data examples. However, since sentences differ in lengths, padding values are added to equal all the tweet lengths. The max length for a tweet has been chosen as the length of the longest sentence.
    - *One Hot Encoding:* The expected output is one hot encoded for the ease of passing into the model.

- **Model Creation:** The model used for this project is a Sequential Model since the output from each layer is used as the input for the next layer. The first layer in this model is an Embedding Layer that converts every word into a vector with 300 features. This layer is then followed by an LSTM layer with 32 nodes. The data is then passed to a Dense layer with 32 nodes and ReLU activation. Another Desnse layer has been fit with 16 nodes and ReLU Activation. Finally, the output is produced by another Dense layer with 13(for the 12 tags + 1 padded tag) nodes that uses Softmax Activation due to the non-binary nature of the problem statement. These layers have been elaborated and described in the next section.

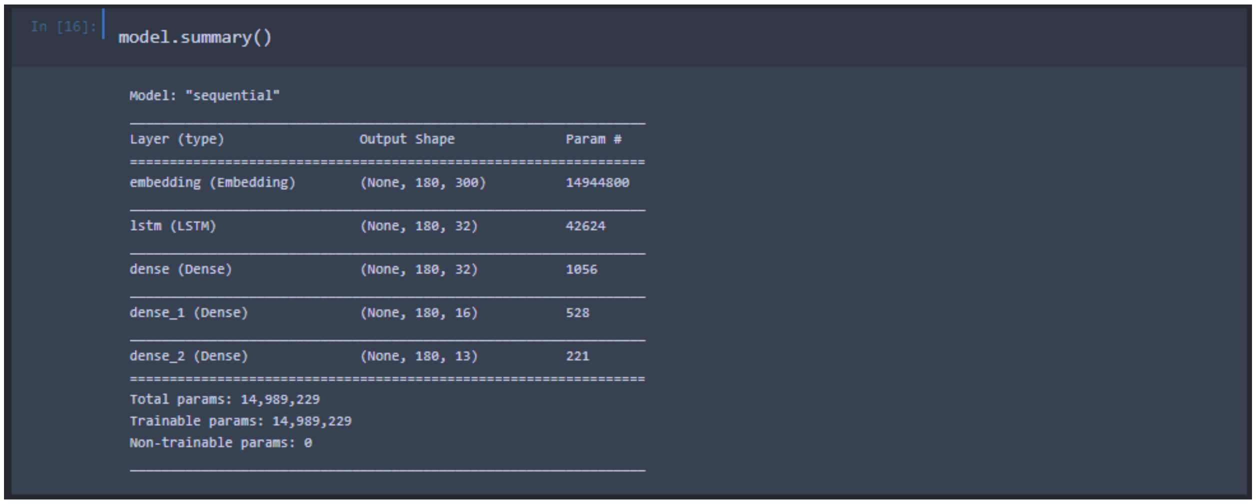
- **Hyperparameters:**
  - Epochs: 30 (Callback and Early Stopping has been implemented)
  - Optimizer: Adam
  - Learning Rate: 0.0001
  - Training Batch Size: 128
  - Loss Function: Categorical Cross Entropy
  - Max Tweet Length: 180
  - Embedding Layer Dimension = 300
  - Early Stopping Patience = 3

IV. MODEL SUMMARY

The Model Architecture adopted to address the problem has been described in the prior section. The different layers added are further elaborated here:

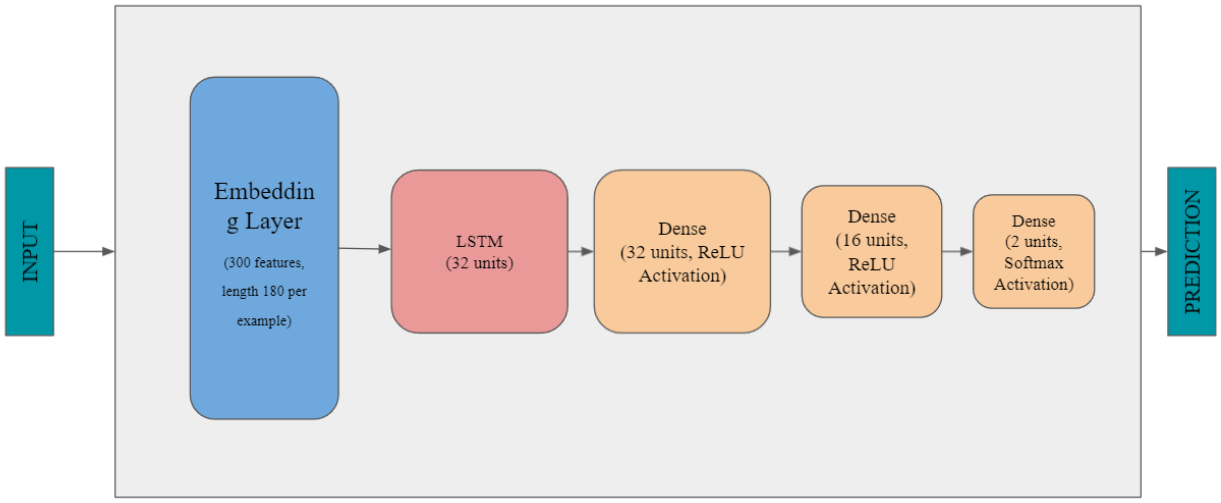
- *Embedding Layer* [5]
  - Used to gain a dense representation of words and their relative meaning.
  - Requires three arguments
    - Input Dimensions: Size of Vocabulary – Number of unique words (49816)
    - Output Dimensions: Size of feature vector – 300
    - Input Length: Size of each tweet – 120
- *LSTM Layer* [6]
  - Long Short-Term Memory
  - Variation of RNN that is effective for predicting long sequences of data
  - Two main arguments:
    - Units: 32
    - return_sequences: set to **True** indicates returning last output as the output
- *Dense Layer*
  - Three Dense Layers are used
    - Layer 1: 32 units, ReLU Activation
    - Layer 2: 16 units, ReLU Activation
    - Layer 3: 13 units, Softmax Activation

The Model Summary is given below:



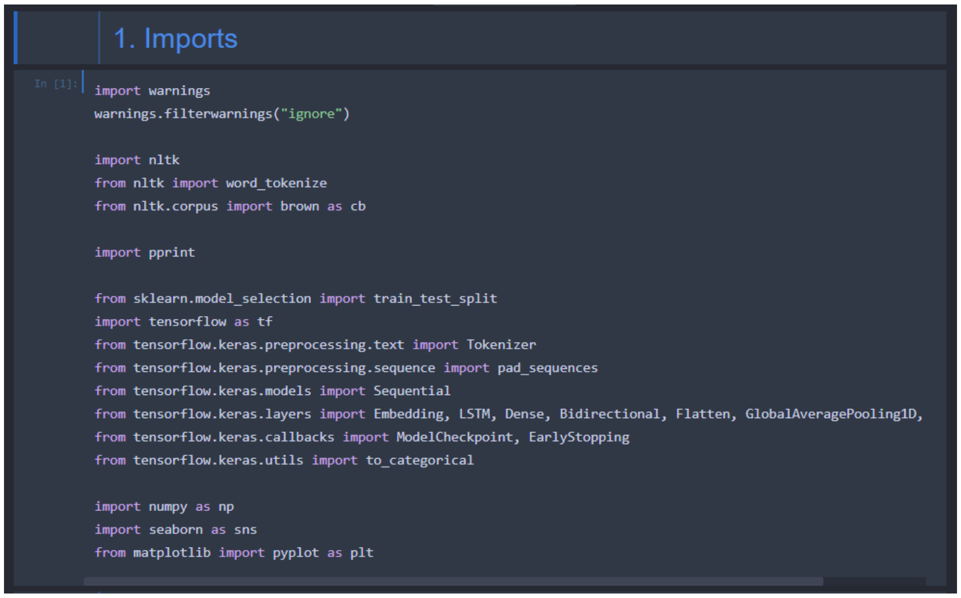
```
In [16]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 180, 300)          14944800
_________________________________________________________________
lstm (LSTM)                  (None, 180, 32)           42624
_________________________________________________________________
dense (Dense)                (None, 180, 32)           1056
_________________________________________________________________
dense_1 (Dense)              (None, 180, 16)           528
_________________________________________________________________
dense_2 (Dense)              (None, 180, 13)           221
=================================================================
Total params: 14,989,229
Trainable params: 14,989,229
Non-trainable params: 0
_________________________________________________________________
```

The Model Architecture is shown below:


INPUT
Embeddin g Layer
(300 features, length 180 per example)
LSTM (32 units)
Dense (32 units, ReLU Activation)
Dense (16 units, ReLU Activation)
Dense (2 units, Softmax Activation)
PREDICTION

V. CODING SNAPSHOTS

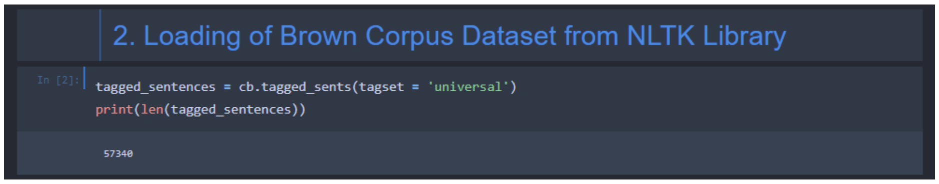
1) *Imports*



1. Imports

```
In [1]: import warnings
        warnings.filterwarnings("ignore")

        import nltk
        from nltk import word_tokenize
        from nltk.corpus import brown as cb

        import pprint

        from sklearn.model_selection import train_test_split
        import tensorflow as tf
        from tensorflow.keras.preprocessing.text import Tokenizer
        from tensorflow.keras.preprocessing.sequence import pad_sequences
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Flatten, GlobalAveragePooling1D,
        from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
        from tensorflow.keras.utils import to_categorical

        import numpy as np
        import seaborn as sns
        from matplotlib import pyplot as plt
```

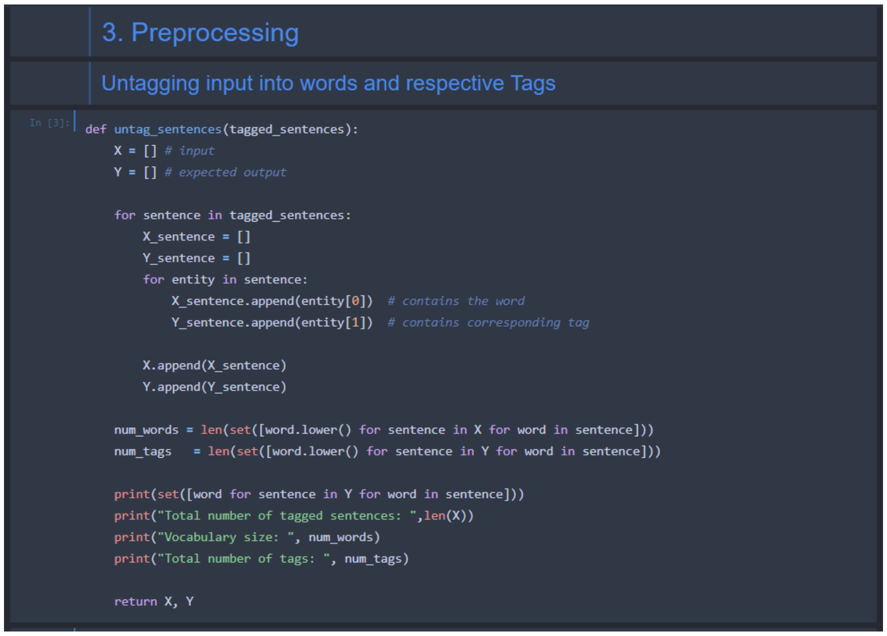
2) *Loading of data set*



2. Loading of Brown Corpus Dataset from NLTK Library

```
In [2]: tagged_sentences = cb.tagged_sents(tagset = 'universal')
        print(len(tagged_sentences))

        57340
```

3) *Data Pre-Processing*

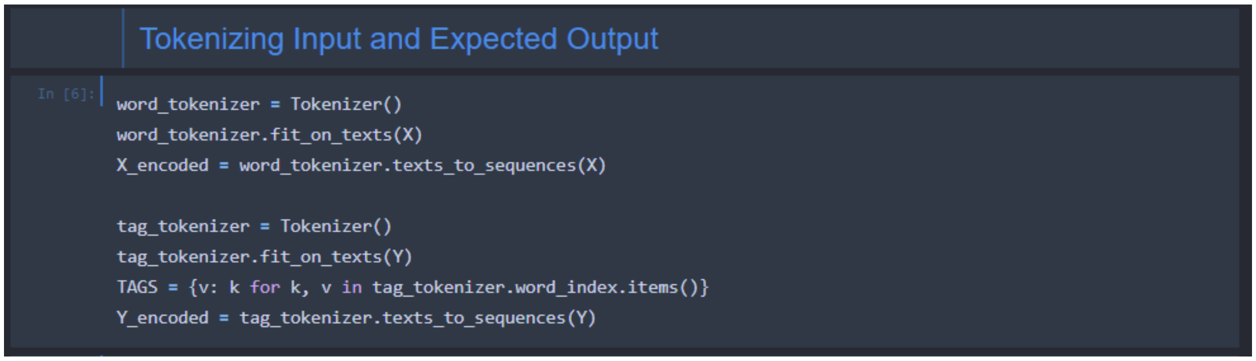


3. Preprocessing

Untagging input into words and respective Tags

```
In [3]: def untag_sentences(tagged_sentences):
            X = [] # input
            Y = [] # expected output

            for sentence in tagged_sentences:
                X_sentence = []
                Y_sentence = []
                for entity in sentence:
                    X_sentence.append(entity[0])  # contains the word
                    Y_sentence.append(entity[1])  # contains corresponding tag

                X.append(X_sentence)
                Y.append(Y_sentence)

            num_words = len(set([word.lower() for sentence in X for word in sentence]))
            num_tags  = len(set([word.lower() for sentence in Y for word in sentence]))

            print(set([word for sentence in Y for word in sentence]))
            print("Total number of tagged sentences: ",len(X))
            print("Vocabulary size: ", num_words)
            print("Total number of tags: ", num_tags)

            return X, Y
```

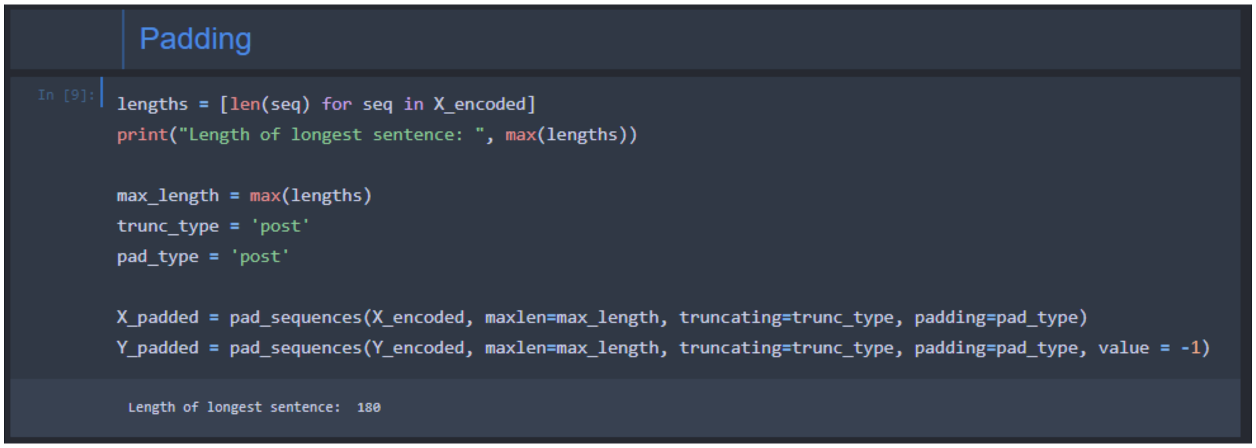
4) Tokenizing



Tokenizing Input and Expected Output

```
In [6]: word_tokenizer = Tokenizer()
        word_tokenizer.fit_on_texts(X)
        X_encoded = word_tokenizer.texts_to_sequences(X)

        tag_tokenizer = Tokenizer()
        tag_tokenizer.fit_on_texts(Y)
        TAGS = {v: k for k, v in tag_tokenizer.word_index.items()}
        Y_encoded = tag_tokenizer.texts_to_sequences(Y)
```

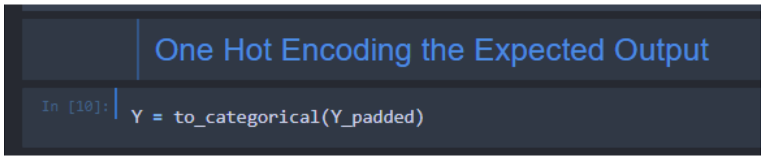
5) Padding



Padding

```
In [9]: lengths = [len(seq) for seq in X_encoded]
        print("Length of longest sentence: ", max(lengths))

        max_length = max(lengths)
        trunc_type = 'post'
        pad_type = 'post'

        X_padded = pad_sequences(X_encoded, maxlen=max_length, truncating=trunc_type, padding=pad_type)
        Y_padded = pad_sequences(Y_encoded, maxlen=max_length, truncating=trunc_type, padding=pad_type, value = -1)

         Length of longest sentence:  180
```

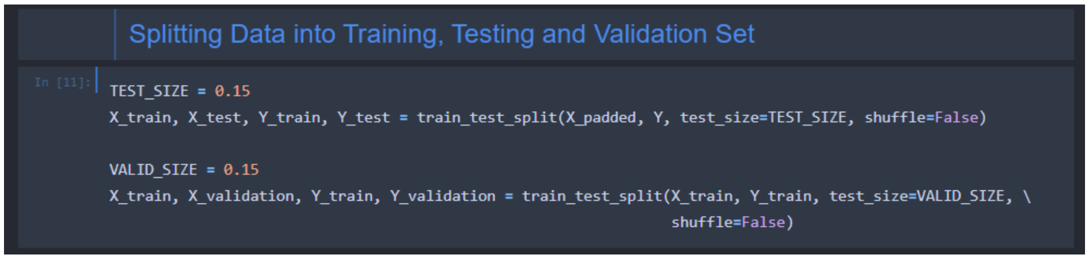
6) One Hot Encoding



One Hot Encoding the Expected Output

```
In [10]: Y = to_categorical(Y_padded)
```

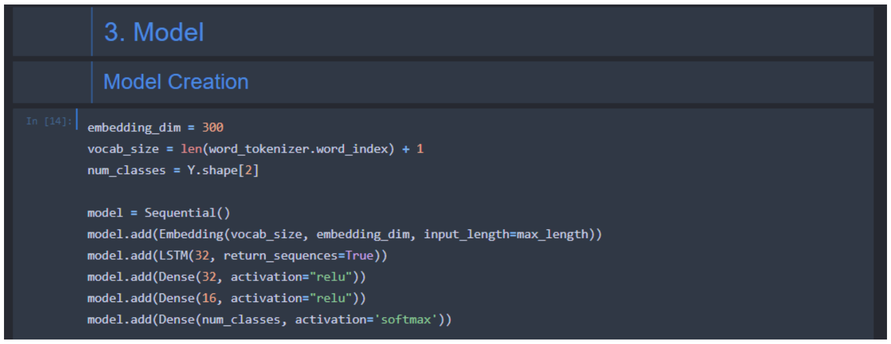
7) Splitting Data



Splitting Data into Training, Testing and Validation Set

```
In [11]: TEST_SIZE = 0.15
         X_train, X_test, Y_train, Y_test = train_test_split(X_padded, Y, test_size=TEST_SIZE, shuffle=False)

         VALID_SIZE = 0.15
         X_train, X_validation, Y_train, Y_validation = train_test_split(X_train, Y_train, test_size=VALID_SIZE, \
                                                                          shuffle=False)
```

8) *Model Creation*

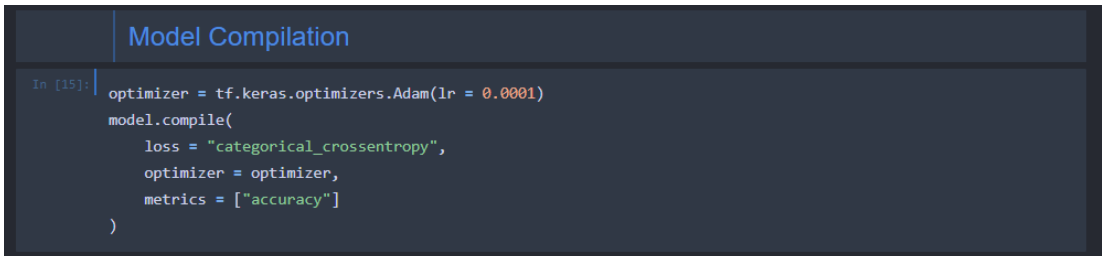


3. Model

Model Creation

In [14]:
```
embedding_dim = 300
vocab_size = len(word_tokenizer.word_index) + 1
num_classes = Y.shape[2]

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))
model.add(LSTM(32, return_sequences=True))
model.add(Dense(32, activation="relu"))
model.add(Dense(16, activation="relu"))
model.add(Dense(num_classes, activation='softmax'))
```

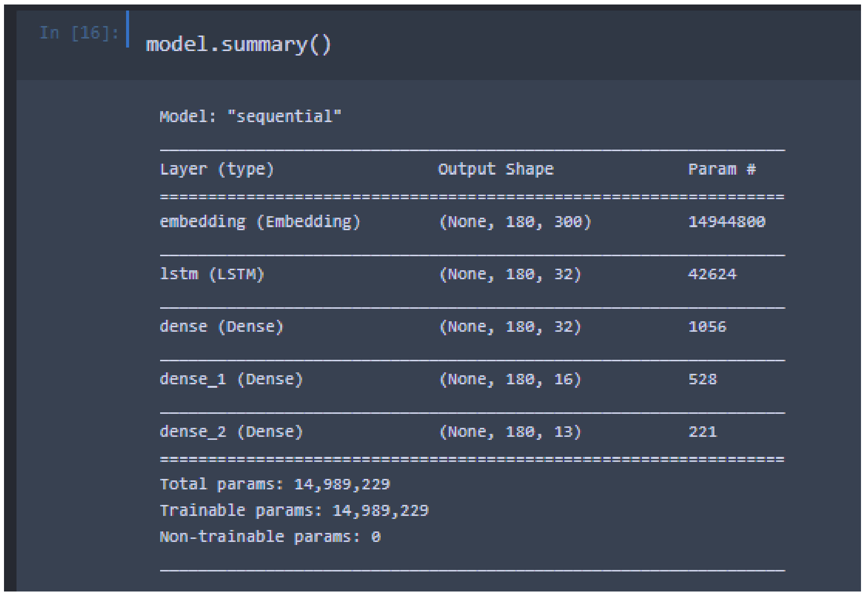
9) *Optimizer and model compilation*



Model Compilation

In [15]:
```
optimizer = tf.keras.optimizers.Adam(lr = 0.0001)
model.compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer,
    metrics = ["accuracy"]
)
```

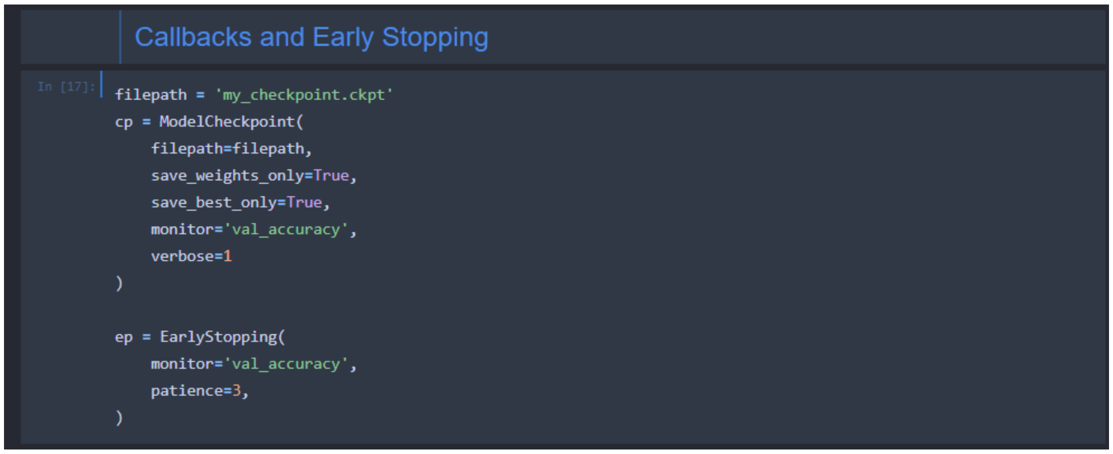
10) *Model Summary*



In [16]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 180, 300)          14944800
_________________________________________________________________
lstm (LSTM)                  (None, 180, 32)           42624
_________________________________________________________________
dense (Dense)                (None, 180, 32)           1056
_________________________________________________________________
dense_1 (Dense)              (None, 180, 16)           528
_________________________________________________________________
dense_2 (Dense)              (None, 180, 13)           221
=================================================================
Total params: 14,989,229
Trainable params: 14,989,229
Non-trainable params: 0
_________________________________________________________________
```

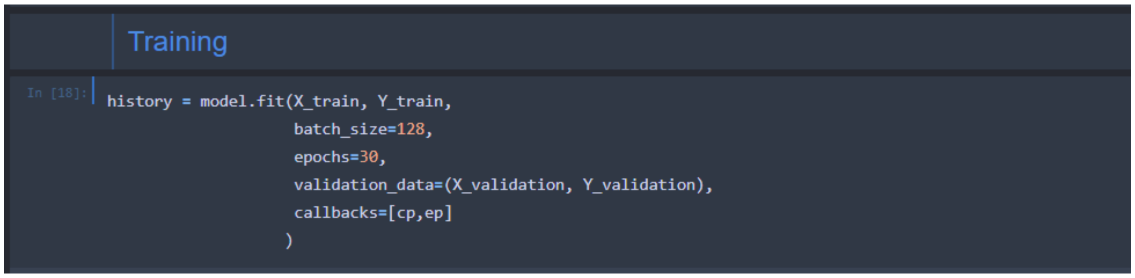
11) Setting up of Checkpoint and Early Stopping to choose best model from epochs



Callbacks and Early Stopping

```
In [17]:
filepath = 'my_checkpoint.ckpt'
cp = ModelCheckpoint(
    filepath=filepath,
    save_weights_only=True,
    save_best_only=True,
    monitor='val_accuracy',
    verbose=1
)

ep = EarlyStopping(
    monitor='val_accuracy',
    patience=3,
)
```

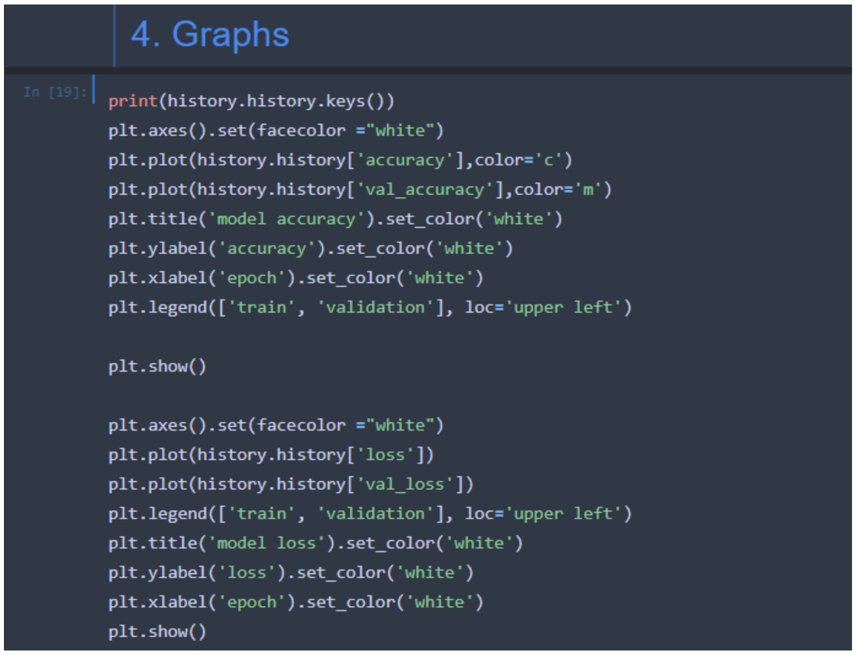
12)Training



Training

```
In [18]:
history = model.fit(X_train, Y_train,
                    batch_size=128,
                    epochs=30,
                    validation_data=(X_validation, Y_validation),
                    callbacks=[cp,ep]
                   )
```

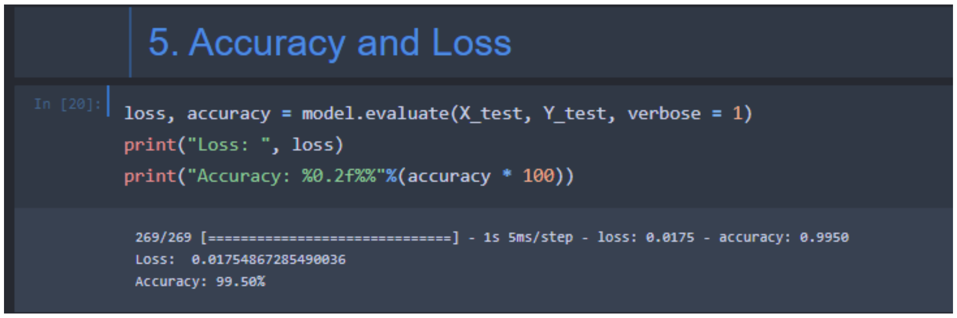
13) Graphs



4. Graphs

```
In [19]:
print(history.history.keys())
plt.axes().set(facecolor ="white")
plt.plot(history.history['accuracy'],color='c')
plt.plot(history.history['val_accuracy'],color='m')
plt.title('model accuracy').set_color('white')
plt.ylabel('accuracy').set_color('white')
plt.xlabel('epoch').set_color('white')
plt.legend(['train', 'validation'], loc='upper left')

plt.show()

plt.axes().set(facecolor ="white")
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train', 'validation'], loc='upper left')
plt.title('model loss').set_color('white')
plt.ylabel('loss').set_color('white')
plt.xlabel('epoch').set_color('white')
plt.show()
```

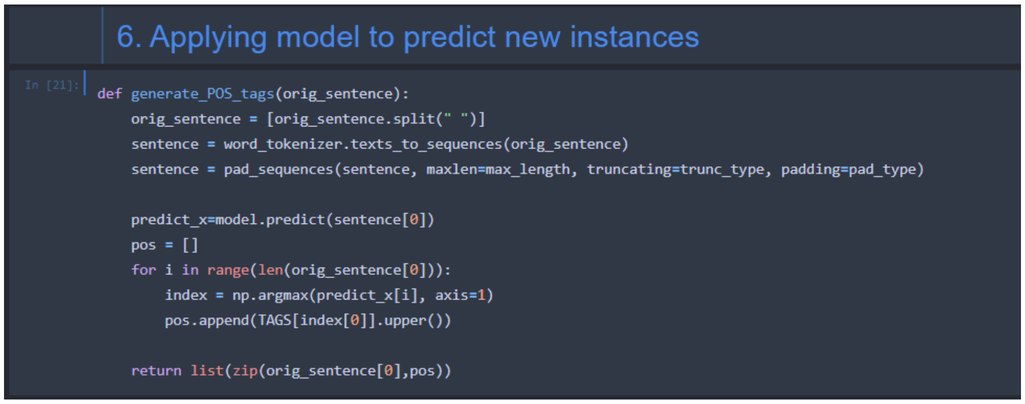
14) Model Evaluation



5. Accuracy and Loss

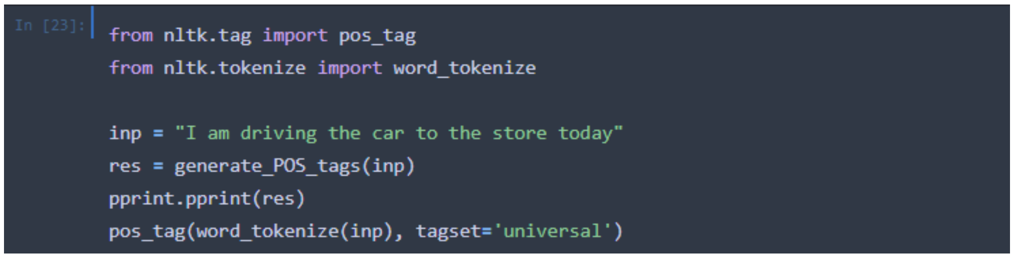
```
In [20]: loss, accuracy = model.evaluate(X_test, Y_test, verbose = 1)
         print("Loss: ", loss)
         print("Accuracy: %0.2f%%"%(accuracy * 100))

269/269 [==============================] - 1s 5ms/step - loss: 0.0175 - accuracy: 0.9950
Loss:  0.017548672854900036
Accuracy: 99.50%
```

15) POS Tagging New Data

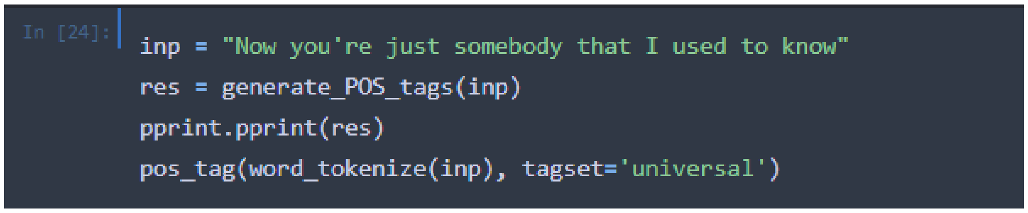


6. Applying model to predict new instances

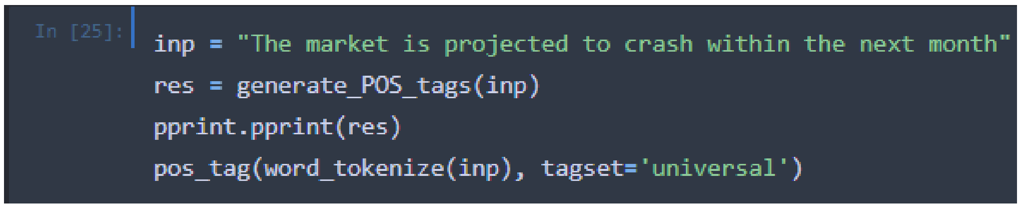
```
In [21]: def generate_POS_tags(orig_sentence):
             orig_sentence = [orig_sentence.split(" ")]
             sentence = word_tokenizer.texts_to_sequences(orig_sentence)
             sentence = pad_sequences(sentence, maxlen=max_length, truncating=trunc_type, padding=pad_type)

             predict_x=model.predict(sentence[0])
             pos = []
             for i in range(len(orig_sentence[0])):
                 index = np.argmax(predict_x[i], axis=1)
                 pos.append(TAGS[index[0]].upper())

             return list(zip(orig_sentence[0],pos))
```



```
In [23]: from nltk.tag import pos_tag
         from nltk.tokenize import word_tokenize

         inp = "I am driving the car to the store today"
         res = generate_POS_tags(inp)
         pprint.pprint(res)
         pos_tag(word_tokenize(inp), tagset='universal')
```



```
In [24]: inp = "Now you're just somebody that I used to know"
         res = generate_POS_tags(inp)
         pprint.pprint(res)
         pos_tag(word_tokenize(inp), tagset='universal')
```

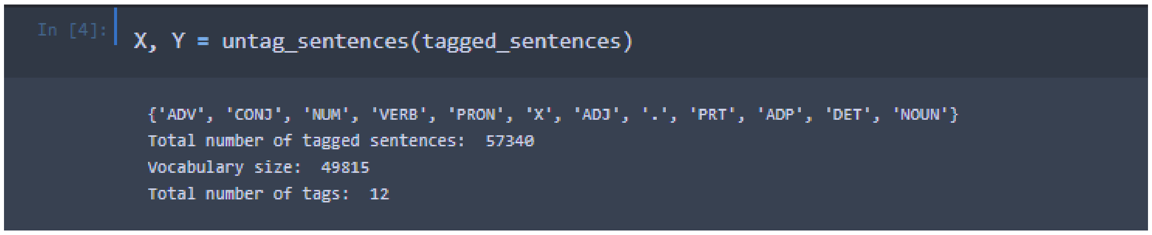


```
In [25]: inp = "The market is projected to crash within the next month"
         res = generate_POS_tags(inp)
         pprint.pprint(res)
         pos_tag(word_tokenize(inp), tagset='universal')
```

VI. RESULTS

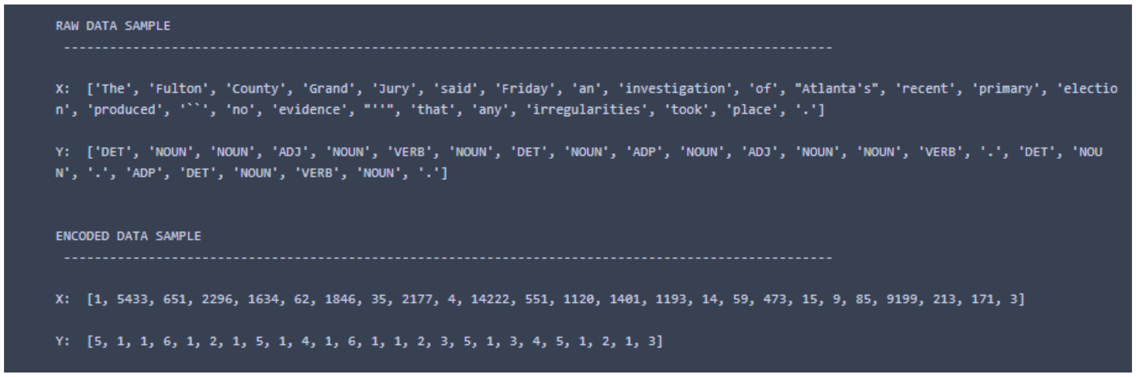
The different outputs corresponding to the driver code has been shown below:

1) *Tags and Dataset information*



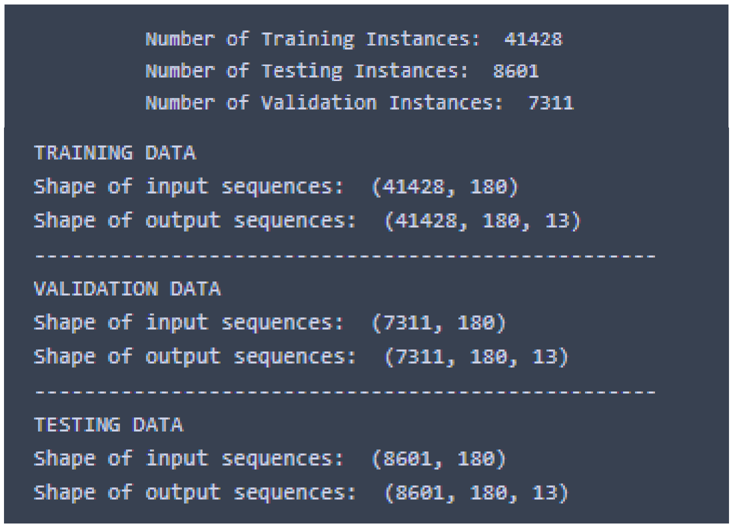
```
In [4]: X, Y = untag_sentences(tagged_sentences)

        {'ADV', 'CONJ', 'NUM', 'VERB', 'PRON', 'X', 'ADJ', '.', 'PRT', 'ADP', 'DET', 'NOUN'}
        Total number of tagged sentences:  57340
        Vocabulary size:  49815
        Total number of tags:  12
```

2) *Raw Data and Encoded Data sample*



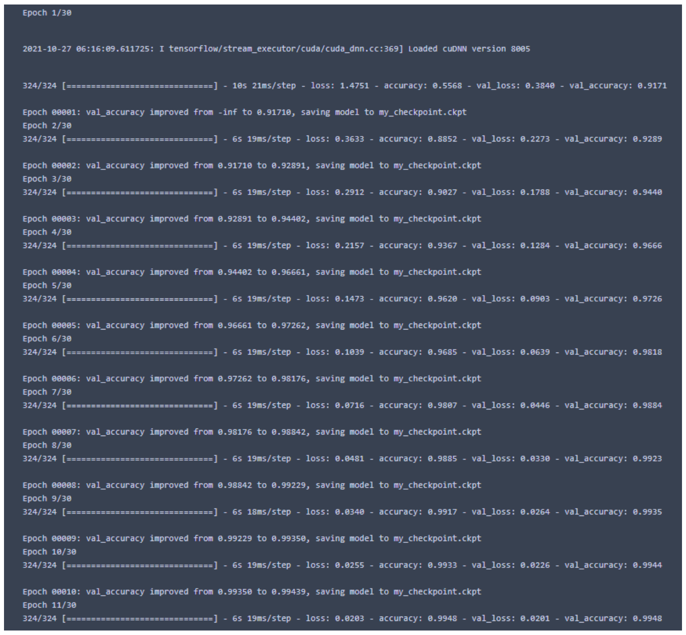
```
RAW DATA SAMPLE
 ----------------------------------------------------------------------------------------------------

X:  ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', "Atlanta's", 'recent', 'primary', 'electio
n', 'produced', '``', 'no', 'evidence', "''", 'that', 'any', 'irregularities', 'took', 'place', '.']

Y:  ['DET', 'NOUN', 'NOUN', 'ADJ', 'NOUN', 'VERB', 'NOUN', 'DET', 'NOUN', 'ADP', 'NOUN', 'ADJ', 'NOUN', 'NOUN', 'VERB', '.', 'DET', 'NOU
N', '.', 'ADP', 'DET', 'NOUN', 'VERB', 'NOUN', '.']


ENCODED DATA SAMPLE
 ----------------------------------------------------------------------------------------------------

X:  [1, 5433, 651, 2296, 1634, 62, 1846, 35, 2177, 4, 14222, 551, 1120, 1401, 1193, 14, 59, 473, 15, 9, 85, 9199, 213, 171, 3]

Y:  [5, 1, 1, 6, 1, 2, 1, 5, 1, 4, 1, 6, 1, 1, 2, 3, 5, 1, 3, 4, 5, 1, 2, 1, 3]
```

3) *Data Splitting Details*

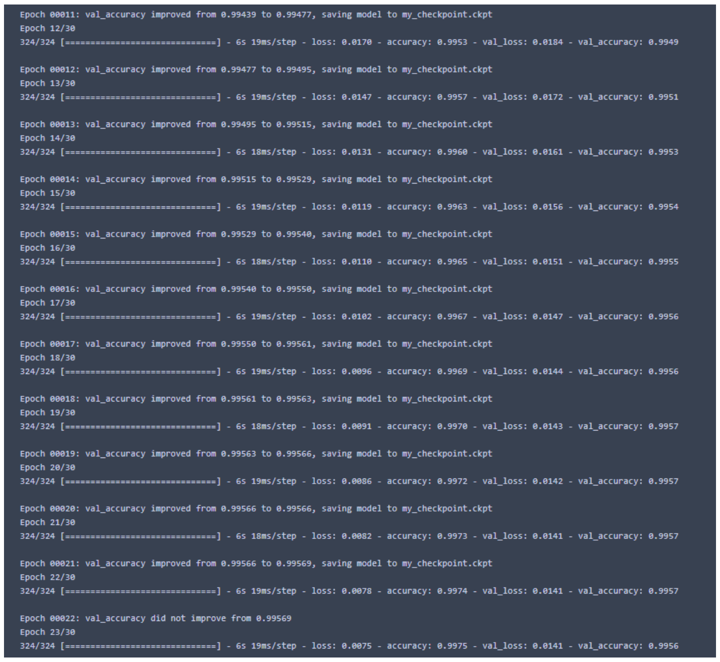


```
          Number of Training Instances:  41428
          Number of Testing Instances:  8601
          Number of Validation Instances:  7311

TRAINING DATA
Shape of input sequences:  (41428, 180)
Shape of output sequences:  (41428, 180, 13)
--------------------------------------------------
VALIDATION DATA
Shape of input sequences:  (7311, 180)
Shape of output sequences:  (7311, 180, 13)
--------------------------------------------------
TESTING DATA
Shape of input sequences:  (8601, 180)
Shape of output sequences:  (8601, 180, 13)
```

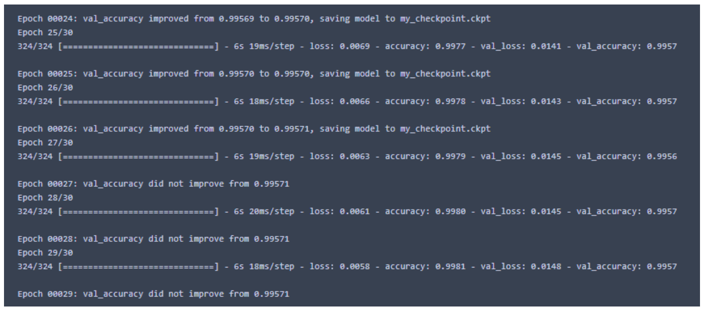
4) Training



```
Epoch 1/30


2021-10-27 06:16:09.611725: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005


324/324 [==============================] - 10s 21ms/step - loss: 1.4751 - accuracy: 0.5568 - val_loss: 0.3840 - val_accuracy: 0.9171

Epoch 00001: val_accuracy improved from -inf to 0.91710, saving model to my_checkpoint.ckpt
Epoch 2/30
324/324 [==============================] - 6s 19ms/step - loss: 0.3633 - accuracy: 0.8852 - val_loss: 0.2273 - val_accuracy: 0.9289

Epoch 00002: val_accuracy improved from 0.91710 to 0.92891, saving model to my_checkpoint.ckpt
Epoch 3/30
324/324 [==============================] - 6s 19ms/step - loss: 0.2912 - accuracy: 0.9027 - val_loss: 0.1788 - val_accuracy: 0.9440

Epoch 00003: val_accuracy improved from 0.92891 to 0.94402, saving model to my_checkpoint.ckpt
Epoch 4/30
324/324 [==============================] - 6s 19ms/step - loss: 0.2157 - accuracy: 0.9367 - val_loss: 0.1284 - val_accuracy: 0.9666

Epoch 00004: val_accuracy improved from 0.94402 to 0.96661, saving model to my_checkpoint.ckpt
Epoch 5/30
324/324 [==============================] - 6s 19ms/step - loss: 0.1473 - accuracy: 0.9620 - val_loss: 0.0903 - val_accuracy: 0.9726

Epoch 00005: val_accuracy improved from 0.96661 to 0.97262, saving model to my_checkpoint.ckpt
Epoch 6/30
324/324 [==============================] - 6s 19ms/step - loss: 0.1039 - accuracy: 0.9685 - val_loss: 0.0639 - val_accuracy: 0.9818

Epoch 00006: val_accuracy improved from 0.97262 to 0.98176, saving model to my_checkpoint.ckpt
Epoch 7/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0716 - accuracy: 0.9807 - val_loss: 0.0446 - val_accuracy: 0.9884

Epoch 00007: val_accuracy improved from 0.98176 to 0.98842, saving model to my_checkpoint.ckpt
Epoch 8/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0481 - accuracy: 0.9885 - val_loss: 0.0330 - val_accuracy: 0.9923

Epoch 00008: val_accuracy improved from 0.98842 to 0.99229, saving model to my_checkpoint.ckpt
Epoch 9/30
324/324 [==============================] - 6s 18ms/step - loss: 0.0340 - accuracy: 0.9917 - val_loss: 0.0264 - val_accuracy: 0.9935

Epoch 00009: val_accuracy improved from 0.99229 to 0.99350, saving model to my_checkpoint.ckpt
Epoch 10/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0255 - accuracy: 0.9933 - val_loss: 0.0226 - val_accuracy: 0.9944

Epoch 00010: val_accuracy improved from 0.99350 to 0.99439, saving model to my_checkpoint.ckpt
Epoch 11/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0203 - accuracy: 0.9948 - val_loss: 0.0201 - val_accuracy: 0.9948
```

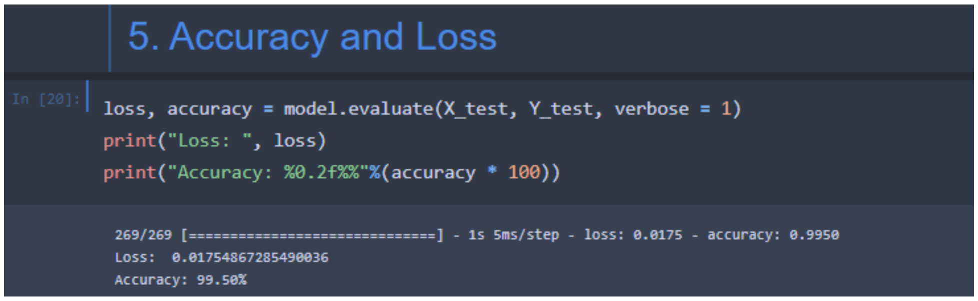


```
Epoch 00011: val_accuracy improved from 0.99439 to 0.99477, saving model to my_checkpoint.ckpt
Epoch 12/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0170 - accuracy: 0.9953 - val_loss: 0.0184 - val_accuracy: 0.9949

Epoch 00012: val_accuracy improved from 0.99477 to 0.99495, saving model to my_checkpoint.ckpt
Epoch 13/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0147 - accuracy: 0.9957 - val_loss: 0.0172 - val_accuracy: 0.9951

Epoch 00013: val_accuracy improved from 0.99495 to 0.99515, saving model to my_checkpoint.ckpt
Epoch 14/30
324/324 [==============================] - 6s 18ms/step - loss: 0.0131 - accuracy: 0.9960 - val_loss: 0.0161 - val_accuracy: 0.9953

Epoch 00014: val_accuracy improved from 0.99515 to 0.99529, saving model to my_checkpoint.ckpt
Epoch 15/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0119 - accuracy: 0.9963 - val_loss: 0.0156 - val_accuracy: 0.9954

Epoch 00015: val_accuracy improved from 0.99529 to 0.99540, saving model to my_checkpoint.ckpt
Epoch 16/30
324/324 [==============================] - 6s 18ms/step - loss: 0.0110 - accuracy: 0.9965 - val_loss: 0.0151 - val_accuracy: 0.9955

Epoch 00016: val_accuracy improved from 0.99540 to 0.99550, saving model to my_checkpoint.ckpt
Epoch 17/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0102 - accuracy: 0.9967 - val_loss: 0.0147 - val_accuracy: 0.9956

Epoch 00017: val_accuracy improved from 0.99550 to 0.99561, saving model to my_checkpoint.ckpt
Epoch 18/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0096 - accuracy: 0.9969 - val_loss: 0.0144 - val_accuracy: 0.9956

Epoch 00018: val_accuracy improved from 0.99561 to 0.99563, saving model to my_checkpoint.ckpt
Epoch 19/30
324/324 [==============================] - 6s 18ms/step - loss: 0.0091 - accuracy: 0.9970 - val_loss: 0.0143 - val_accuracy: 0.9957

Epoch 00019: val_accuracy improved from 0.99563 to 0.99566, saving model to my_checkpoint.ckpt
Epoch 20/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0086 - accuracy: 0.9972 - val_loss: 0.0142 - val_accuracy: 0.9957

Epoch 00020: val_accuracy improved from 0.99566 to 0.99566, saving model to my_checkpoint.ckpt
Epoch 21/30
324/324 [==============================] - 6s 18ms/step - loss: 0.0082 - accuracy: 0.9973 - val_loss: 0.0141 - val_accuracy: 0.9957

Epoch 00021: val_accuracy improved from 0.99566 to 0.99569, saving model to my_checkpoint.ckpt
Epoch 22/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0078 - accuracy: 0.9974 - val_loss: 0.0141 - val_accuracy: 0.9957

Epoch 00022: val_accuracy did not improve from 0.99569
Epoch 23/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0075 - accuracy: 0.9975 - val_loss: 0.0141 - val_accuracy: 0.9956
```



```
Epoch 00024: val_accuracy improved from 0.99569 to 0.99570, saving model to my_checkpoint.ckpt
Epoch 25/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0069 - accuracy: 0.9977 - val_loss: 0.0141 - val_accuracy: 0.9957

Epoch 00025: val_accuracy improved from 0.99570 to 0.99570, saving model to my_checkpoint.ckpt
Epoch 26/30
324/324 [==============================] - 6s 18ms/step - loss: 0.0066 - accuracy: 0.9978 - val_loss: 0.0143 - val_accuracy: 0.9957

Epoch 00026: val_accuracy improved from 0.99570 to 0.99571, saving model to my_checkpoint.ckpt
Epoch 27/30
324/324 [==============================] - 6s 19ms/step - loss: 0.0063 - accuracy: 0.9979 - val_loss: 0.0145 - val_accuracy: 0.9956

Epoch 00027: val_accuracy did not improve from 0.99571
Epoch 28/30
324/324 [==============================] - 6s 20ms/step - loss: 0.0061 - accuracy: 0.9980 - val_loss: 0.0145 - val_accuracy: 0.9957

Epoch 00028: val_accuracy did not improve from 0.99571
Epoch 29/30
324/324 [==============================] - 6s 18ms/step - loss: 0.0058 - accuracy: 0.9981 - val_loss: 0.0148 - val_accuracy: 0.9957

Epoch 00029: val_accuracy did not improve from 0.99571
```

5) *Accuracy and evaluation*

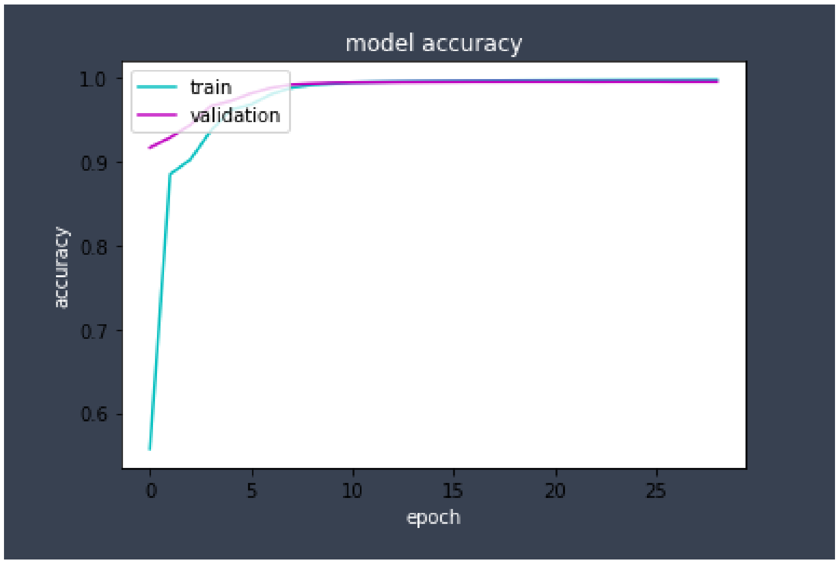


5. Accuracy and Loss

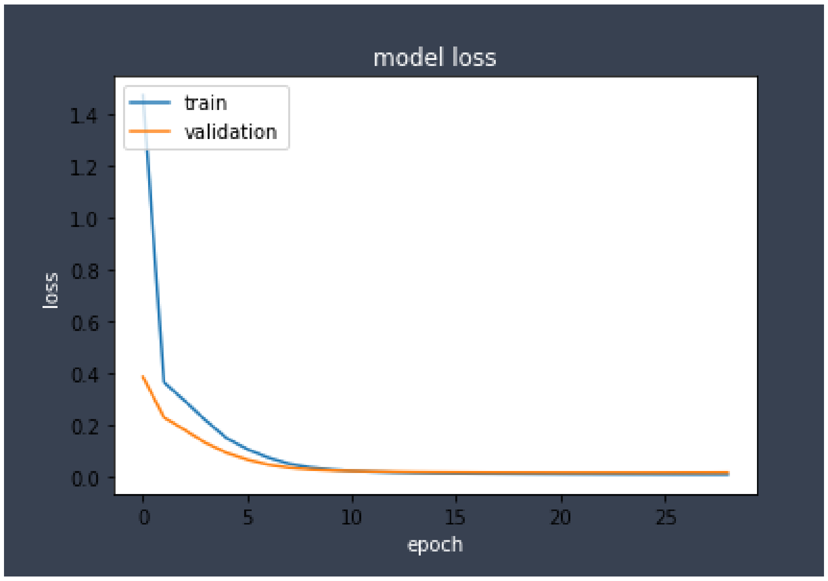
```
In [20]:
loss, accuracy = model.evaluate(X_test, Y_test, verbose = 1)
print("Loss: ", loss)
print("Accuracy: %0.2f%%"%(accuracy * 100))
```

```
269/269 [==============================] - 1s 5ms/step - loss: 0.0175 - accuracy: 0.9950
Loss:  0.01754867285490036
Accuracy: 99.50%
```

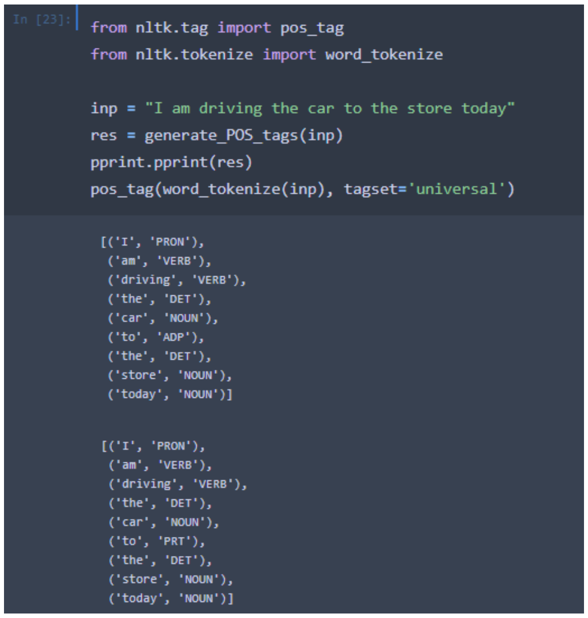
6) *Model Accuracy Graph*



7) *Model Loss Graph*

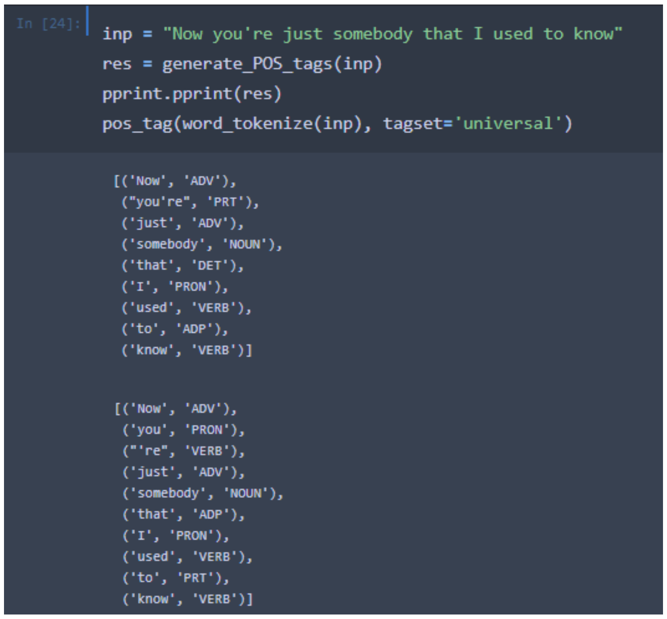


8) *New Sentence POS Tagging (compared with results from NLTK's POS Tagger)*



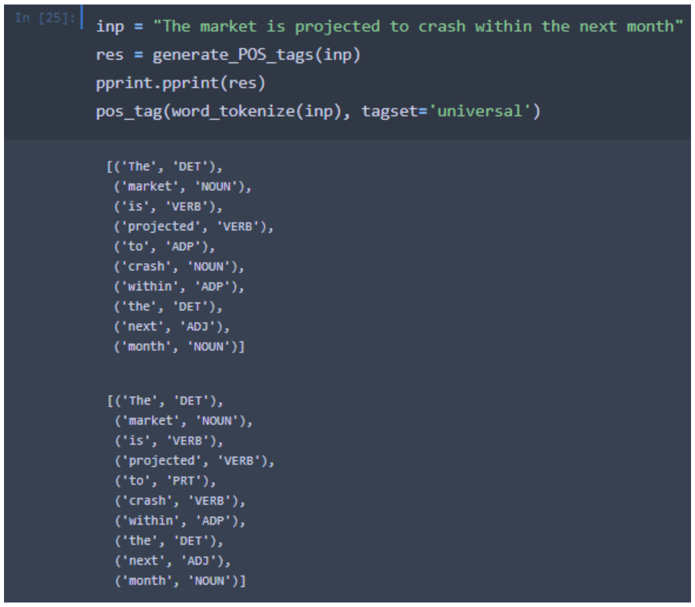
```
In [23]: from nltk.tag import pos_tag
         from nltk.tokenize import word_tokenize

         inp = "I am driving the car to the store today"
         res = generate_POS_tags(inp)
         pprint.pprint(res)
         pos_tag(word_tokenize(inp), tagset='universal')
```

```
[('I', 'PRON'),
 ('am', 'VERB'),
 ('driving', 'VERB'),
 ('the', 'DET'),
 ('car', 'NOUN'),
 ('to', 'ADP'),
 ('the', 'DET'),
 ('store', 'NOUN'),
 ('today', 'NOUN')]


[('I', 'PRON'),
 ('am', 'VERB'),
 ('driving', 'VERB'),
 ('the', 'DET'),
 ('car', 'NOUN'),
 ('to', 'PRT'),
 ('the', 'DET'),
 ('store', 'NOUN'),
 ('today', 'NOUN')]
```



```
In [24]: inp = "Now you're just somebody that I used to know"
         res = generate_POS_tags(inp)
         pprint.pprint(res)
         pos_tag(word_tokenize(inp), tagset='universal')
```

```
[('Now', 'ADV'),
 ("you're", 'PRT'),
 ('just', 'ADV'),
 ('somebody', 'NOUN'),
 ('that', 'DET'),
 ('I', 'PRON'),
 ('used', 'VERB'),
 ('to', 'ADP'),
 ('know', 'VERB')]


[('Now', 'ADV'),
 ('you', 'PRON'),
 ("'re", 'VERB'),
 ('just', 'ADV'),
 ('somebody', 'NOUN'),
 ('that', 'ADP'),
 ('I', 'PRON'),
 ('used', 'VERB'),
 ('to', 'PRT'),
 ('know', 'VERB')]
```



```
In [25]: inp = "The market is projected to crash within the next month"
         res = generate_POS_tags(inp)
         pprint.pprint(res)
         pos_tag(word_tokenize(inp), tagset='universal')
```

```
[('The', 'DET'),
 ('market', 'NOUN'),
 ('is', 'VERB'),
 ('projected', 'VERB'),
 ('to', 'ADP'),
 ('crash', 'NOUN'),
 ('within', 'ADP'),
 ('the', 'DET'),
 ('next', 'ADJ'),
 ('month', 'NOUN')]


[('The', 'DET'),
 ('market', 'NOUN'),
 ('is', 'VERB'),
 ('projected', 'VERB'),
 ('to', 'PRT'),
 ('crash', 'VERB'),
 ('within', 'ADP'),
 ('the', 'DET'),
 ('next', 'ADJ'),
 ('month', 'NOUN')]
```

VII. CONCLUSIONS

As seen from the output screenshots, the model performs well with the dataset. It managed to score a Training Accuracy of 99.81% in the 29th epoch before the early stopping was activated. The validation accuracy appeared to drop from Epoch 27. Therefore, early stopping came into play and the weights with Validation Accuracy of 99.571% were taken. The Validation Loss was observed to be 0.0143. This however was not the lowest value since that was observed at epoch 25 with a value 0.0141. The callback function monitored the validation accuracy and the model at the 27th epoch was saved as it has the highest validation accuracy (of 99.571%). On testing, a Testing Accuracy of 99.50% was noted.

As a result, a POS Tagger was developed using the Brown Corpus for assigning the Universal POS Tagset.

VIII. REFERENCES

[1] https://www.nltk.org/book/ch02.html

[2] Marneffe, M.-C & Dozat, T. & Silveira, N. & Haverinen, K. & Ginter, F. & Nivre, Joakim & Manning, C.D.. (2014). Universal Stanford Dependencies: A cross-linguistic typology. Proceedings of the 9Th International Conference on Language Resources and Evaluation (LREC). 4585-4592.

[3] Petrov, Slav & Das, Dipanjan & McDonald, Ryan. (2011). A Universal Part-of-Speech Tagset. Computing Research Repository - CORR.

[4] Zeman, Daniel. "Reusable Tagset Conversion Using Tagset Drivers." In LREC, vol. 2008, pp. 28-30. 2008.

[5] https://machinelearningmastery.com/

[6] https://machinelearningknowledge.ai/