Neural Part-of-Speech Tagger (POS Tagger)

The goal of this project is to build a neural parts-of-speech tagger.

The data is in JSON format and the key abbreviations are listed below:

- word: word in the particular sentence
- upos: Universal part-of-speech tag
- xpos: Language-specific part-of-speech tag

Metrics: We are going to evaulate our model with accuracy as it is a standard metric for most deep learning models.

author: Pratyush Mohit

In [497…

```
import numpy as np
import json
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import tensorflow as tf
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Input, Embedding, SimpleRNN, LSTM, Dense, TimeDistri
from tensorflow.keras.models import Model

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
with open('telugu_pos (1).json', 'r') as f:
    data = json.load(f)
```

In [3]:

```
data[0:3]
```

Out[3]:

```
[[{'word': 'మరో', 'upos': 'avy', 'xpos': 'QT_QTF'},
  {'word': 'సంగతి', 'upos': 'n', 'xpos': 'N_NN'},
  {'word': 'మీకు', 'upos': 'pn', 'xpos': 'PR_PRP'},
  {'word': 'తెలుసా', 'upos': 'avy', 'xpos': 'V_VM'},
  {'word': '?', 'upos': 'punc', 'xpos': 'RD_PUNC'}],
 [{'word': 'అందరి', 'upos': 'pn', 'xpos': 'PR_PRP'},
  {'word': 'ముందూ', 'upos': 'n', 'xpos': 'N_NST'},
  {'word': 'నా', 'upos': 'pn', 'xpos': 'PR_PRP'},
  {'word': 'తెల్లబట్ట', 'upos': 'n', 'xpos': 'N_NN'},
  {'word': 'బాధ', 'upos': 'n', 'xpos': 'N_NN'},
  {'word': 'ఎలా', 'upos': 'avy', 'xpos': 'PR_PRQ'},
  {'word': 'చెప్పుకోనేది', 'upos': 'unk', 'xpos': 'V_VM'},
  {'word': '?', 'upos': 'punc', 'xpos': 'RD_PUNC'}],
 [{'word': 'ఇట్లా', 'upos': 'avy', 'xpos': 'RB'},
  {'word': 'ఎందుకు', 'upos': 'avy', 'xpos': 'PR_PRQ'},
  {'word': 'జరుగుతోంది', 'upos': 'v', 'xpos': 'V_VM'},
  {'word': '?', 'upos': 'punc', 'xpos': 'RD_PUNC'}]]
```

In [4]:

```
data[0]
```

Out[4]:

```
[{'word': 'మరో', 'upos': 'avy', 'xpos': 'QT_QTF'},
 {'word': 'సంగతి', 'upos': 'n', 'xpos': 'N_NN'},
 {'word': 'మీకు', 'upos': 'pn', 'xpos': 'PR_PRP'},
 {'word': 'తెలుసా', 'upos': 'avy', 'xpos': 'V_VM'},
 {'word': '?', 'upos': 'punc', 'xpos': 'RD_PUNC'}]
```

In [5]:

```
#we will create two datasets. One for upos tags and the other for xpos tags
```

In [6]:

```
all_sentences = []
all_upos = []
all_xpos = []
for sentence in tqdm(data):
    current_sentence = []
    current_upos = []
    current_xpos = []
    for word in sentence:
        current_sentence.append(word['word'])
        current_upos.append(word['upos'])
        current_xpos.append(word['xpos'])
    all_sentences.append(current_sentence)
    all_upos.append(current_upos)
    all_xpos.append(current_xpos)
```

```
100%|██████████████████████████████████████████████████████████████████████████
██████████████| 3185/3185 [00:00<00:00, 398153.86it/s]
```

In [7]:

```
print(len(all_sentences))
print(len(all_upos))
print(len(all_xpos))
```

```
3185
3185
3185
```

We are now going to build a model for predicting upos

In [209…

```
train_sentences, test_sentences, train_upos, test_upos = train_test_split(all_
```

In [449…

```
print(len(train_sentences))
print(len(test_sentences))
print(len(train_upos))
print(len(test_upos))
```

```
2548
637
2548
637
```

In [450…

```
train_sentences[0:5]
```

Out[450…

```
[['చదువు', 'తెలివిని', 'పెంచుతుంది', '.'],
 ['పోలీసుల', 'కంట', 'పడింది', '.'],
 ['రెండో', 'ఏడు', 'నిండేలోపల', 'మెదడు', 'బాగా', 'పెరుగుతుంది', '.'],
 ['పాత', 'రకం', 'విత్తనాలు', 'ఈ', 'వ్యాధులను', 'ఋూLL', '.'],
 ['కడుపులో', 'తిరుగుతూ', '.']]
```

In [451…

```
train_upos[0:5]
```

Out[451…

```
[['n', 'n', 'v', 'punc'],
 ['n', 'unk', 'unk', 'punc'],
 ['adj', 'n', 'v', 'n', 'avy', 'v', 'punc'],
 ['adj', 'n', 'n', 'avy', 'n', 'unk', 'punc'],
 ['n', 'v', 'punc']]
```

In [452…

```
words, upos = set([]), set([])

for sentence in train_sentences:
    for word in sentence:
        words.add(word.lower())

for tag in train_upos:
    for t in tag:
        upos.add(t)

word2index = {w: i + 2 for i, w in enumerate(list(words))}
word2index['-PAD-'] = 0  # The special value used for padding
word2index['-OOV-'] = 1  # The special value used for OOVs

upos2index = {t: i + 1 for i, t in enumerate(list(upos))}
upos2index['-PAD-'] = 0
```

In [453…

```
len(words)
```

Out[453…

```
4970
```

In [454…

```
len(upos)
```

Out[454…

```
24
```

In [455…

```
train_sentences_x, test_sentences_x, train_upos_y, test_upos_y = [], [], [],

for sentence in train_sentences:
    sentence_int = []
    for word in sentence:
        try:
            sentence_int.append(word2index[word.lower()])
        except KeyError:
            sentence_int.append(word2index['-OOV-'])
    train_sentences_x.append(sentence_int)

for sentence in test_sentences:
    sentence_int = []
    for word in sentence:
        try:
            sentence_int.append(word2index[word.lower()])
        except KeyError:
            sentence_int.append(word2index['-OOV-'])
    test_sentences_x.append(sentence_int)

for s in train_upos:
    train_upos_y.append([upos2index[t] for t in s])

for s in test_upos:
    test_upos_y.append([upos2index[t] for t in s])
```

In [456…

```
print(train_sentences_x[0])
print(test_sentences_x[0])
print(train_upos_y[0])
print(test_upos_y[0])
```

```
[390, 3852, 2949, 1705]
[1, 1, 647, 1, 1, 3449, 1, 1437, 1705]
[6, 6, 24, 7]
[12, 6, 7, 6, 6, 12, 6, 24, 7]
```

In [457…

```
MAX_LENGTH = len(max(train_sentences_x, key=len))
print(MAX_LENGTH)
```

```
26
```

In [219…

```
train_sentences_x = pad_sequences(train_sentences_x, maxlen=MAX_LENGTH, paddi
test_sentences_x = pad_sequences(test_sentences_x, maxlen=MAX_LENGTH, padding=
train_upos_y = pad_sequences(train_upos_y, maxlen=MAX_LENGTH, padding='post')
test_upos_y = pad_sequences(test_upos_y, maxlen=MAX_LENGTH, padding='post')
```

In [220…

```
print(train_sentences_x[0])
print(test_sentences_x[0])
print(train_upos_y[0])
print(test_upos_y[0])
```

```
[ 390 3852 2949 1705    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0]
[    1    1  647    1    1 3449    1 1437 1705    0    0    0    0    0
```

```
     0     0     0     0     0     0     0     0     0     0     0     0]
[ 6  6 24  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0]
[12  6  7  6  6 12  6 24  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

In [221…

```
def to_categorical(sequences, categories):
    cat_sequences = []
    for s in sequences:
        cats = []
        for item in s:
            cats.append(np.zeros(categories))
            cats[-1][item] = 1.0
        cat_sequences.append(cats)
    return np.array(cat_sequences)
```

In [222…

```
cat_train_upos_y = to_categorical(train_upos_y, len(upos2index))
cat_test_upos_y = to_categorical(test_upos_y, len(upos2index))
```

Model 1 - Vanilla Recurrent Neural Network

In [312…

```
tf.keras.backend.clear_session()

input_layer_1 = Input(shape=(MAX_LENGTH,))
embedding_1 = Embedding(input_dim=len(word2index), output_dim=100)(input_laye
rnn = SimpleRNN(100, return_sequences=True)(embedding_1)
output_1 = TimeDistributed(Dense(len(upos2index)))(rnn)
activation_1 = Activation('softmax')(output_1)
```

In [313…

```
model1 = Model(inputs=[input_layer_1], outputs=[activation_1])
```

In [314…

```
model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac
```

In [315…

```
model1.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 26)]              0         
_________________________________________________________________
embedding (Embedding)        (None, 26, 100)           497200    
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 26, 100)           20100     
_________________________________________________________________
time_distributed (TimeDistri (None, 26, 25)            2525      
_________________________________________________________________
activation (Activation)      (None, 26, 25)            0         
=================================================================
Total params: 519,825
Trainable params: 519,825
Non-trainable params: 0
_________________________________________________________________
```

In [316…

```
history1 = model1.fit(train_sentences_x, cat_train_upos_y, batch_size=128, epo
```

```
Epoch 1/40
20/20 [==============================] - 2s 63ms/step - loss: 2.3126 - accurac
y: 0.5056 - val_loss: 0.7733 - val_accuracy: 0.7907
Epoch 2/40
20/20 [==============================] - 1s 48ms/step - loss: 0.7488 - accurac
y: 0.7941 - val_loss: 0.6890 - val_accuracy: 0.8032
Epoch 3/40
20/20 [==============================] - 1s 47ms/step - loss: 0.6500 - accurac
y: 0.8223 - val_loss: 0.6240 - val_accuracy: 0.8272
Epoch 4/40
20/20 [==============================] - 1s 49ms/step - loss: 0.5696 - accurac
y: 0.8454 - val_loss: 0.5445 - val_accuracy: 0.8806
Epoch 5/40
20/20 [==============================] - 1s 47ms/step - loss: 0.4825 - accurac
y: 0.9141 - val_loss: 0.4810 - val_accuracy: 0.9009
Epoch 6/40
20/20 [==============================] - 1s 47ms/step - loss: 0.3896 - accurac
y: 0.9363 - val_loss: 0.4237 - val_accuracy: 0.9120
Epoch 7/40
20/20 [==============================] - 1s 50ms/step - loss: 0.3247 - accurac
y: 0.9393 - val_loss: 0.3876 - val_accuracy: 0.9139
Epoch 8/40
20/20 [==============================] - 1s 48ms/step - loss: 0.2766 - accurac
y: 0.9416 - val_loss: 0.3480 - val_accuracy: 0.9213
Epoch 9/40
20/20 [==============================] - 1s 47ms/step - loss: 0.2281 - accurac
y: 0.9490 - val_loss: 0.3161 - val_accuracy: 0.9303
Epoch 10/40
20/20 [==============================] - 1s 47ms/step - loss: 0.1985 - accurac
y: 0.9556 - val_loss: 0.2973 - val_accuracy: 0.9337
Epoch 11/40
20/20 [==============================] - 1s 47ms/step - loss: 0.1694 - accurac
y: 0.9630 - val_loss: 0.2665 - val_accuracy: 0.9396
Epoch 12/40
20/20 [==============================] - 1s 48ms/step - loss: 0.1410 - accurac
y: 0.9699 - val_loss: 0.2385 - val_accuracy: 0.9451
Epoch 13/40
20/20 [==============================] - 1s 47ms/step - loss: 0.1179 - accurac
y: 0.9760 - val_loss: 0.2211 - val_accuracy: 0.9478
Epoch 14/40
20/20 [==============================] - 1s 46ms/step - loss: 0.0962 - accurac
y: 0.9822 - val_loss: 0.2039 - val_accuracy: 0.9525
Epoch 15/40
20/20 [==============================] - 1s 46ms/step - loss: 0.0801 - accurac
y: 0.9859 - val_loss: 0.1946 - val_accuracy: 0.9542
Epoch 16/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0686 - accurac
y: 0.9881 - val_loss: 0.1906 - val_accuracy: 0.9551
Epoch 17/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0544 - accurac
y: 0.9907 - val_loss: 0.1847 - val_accuracy: 0.9566
Epoch 18/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0490 - accurac
y: 0.9917 - val_loss: 0.1794 - val_accuracy: 0.9582
Epoch 19/40
20/20 [==============================] - 1s 46ms/step - loss: 0.0428 - accurac
y: 0.9928 - val_loss: 0.1745 - val_accuracy: 0.9588
Epoch 20/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0351 - accurac
y: 0.9942 - val_loss: 0.1744 - val_accuracy: 0.9586
Epoch 21/40
```

```
20/20 [==============================] - 1s 48ms/step - loss: 0.0341 - accurac
y: 0.9940 - val_loss: 0.1716 - val_accuracy: 0.9594
Epoch 22/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0286 - accurac
y: 0.9951 - val_loss: 0.1719 - val_accuracy: 0.9592
Epoch 23/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0259 - accurac
y: 0.9956 - val_loss: 0.1708 - val_accuracy: 0.9595
Epoch 24/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0245 - accurac
y: 0.9957 - val_loss: 0.1690 - val_accuracy: 0.9601
Epoch 25/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0210 - accurac
y: 0.9963 - val_loss: 0.1682 - val_accuracy: 0.9601
Epoch 26/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0211 - accurac
y: 0.9964 - val_loss: 0.1692 - val_accuracy: 0.9599
Epoch 27/40
20/20 [==============================] - 1s 46ms/step - loss: 0.0172 - accurac
y: 0.9970 - val_loss: 0.1690 - val_accuracy: 0.9601
Epoch 28/40
20/20 [==============================] - 1s 45ms/step - loss: 0.0174 - accurac
y: 0.9969 - val_loss: 0.1683 - val_accuracy: 0.9606
Epoch 29/40
20/20 [==============================] - 1s 46ms/step - loss: 0.0154 - accurac
y: 0.9974 - val_loss: 0.1683 - val_accuracy: 0.9602
Epoch 30/40
20/20 [==============================] - 1s 49ms/step - loss: 0.0154 - accurac
y: 0.9975 - val_loss: 0.1681 - val_accuracy: 0.9605
Epoch 31/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0128 - accurac
y: 0.9981 - val_loss: 0.1661 - val_accuracy: 0.9611
Epoch 32/40
20/20 [==============================] - 1s 49ms/step - loss: 0.0128 - accurac
y: 0.9978 - val_loss: 0.1657 - val_accuracy: 0.9612
Epoch 33/40
20/20 [==============================] - 1s 49ms/step - loss: 0.0121 - accurac
y: 0.9978 - val_loss: 0.1674 - val_accuracy: 0.9608
Epoch 34/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0112 - accurac
y: 0.9981 - val_loss: 0.1661 - val_accuracy: 0.9612
Epoch 35/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0101 - accurac
y: 0.9983 - val_loss: 0.1679 - val_accuracy: 0.9610
Epoch 36/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0099 - accurac
y: 0.9982 - val_loss: 0.1667 - val_accuracy: 0.9610
Epoch 37/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0088 - accurac
y: 0.9985 - val_loss: 0.1665 - val_accuracy: 0.9611
Epoch 38/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0085 - accurac
y: 0.9984 - val_loss: 0.1669 - val_accuracy: 0.9609
Epoch 39/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0077 - accurac
y: 0.9986 - val_loss: 0.1669 - val_accuracy: 0.9607
Epoch 40/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0075 - accurac
```

In [317…

```
scores = model1.evaluate(test_sentences_x, cat_test_upos_y)
print(f"{model1.metrics_names[1]}: {scores[1] * 100}")
```

```
20/20 [==============================] - 0s 7ms/step - loss: 0.1675 - accurac
y: 0.9605
```

```
accuracy: 96.05120420455933
```

Model 2 - Long Short Term Memory

In [378…

```
tf.keras.backend.clear_session()

input_layer_2 = Input(shape=(MAX_LENGTH,))
embedding_2 = Embedding(input_dim=len(word2index), output_dim=128)(input_laye
lstm = LSTM(256, return_sequences=True)(embedding_2)
output_2 = TimeDistributed(Dense(len(upos2index)))(lstm)
activation_2 = Activation('softmax')(output_2)
```

In [379…

```
model2 = Model(inputs=[input_layer_2], outputs=[activation_2])
```

In [380…

```
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac
```

In [381…

```
model2.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 26)]              0         
_________________________________________________________________
embedding (Embedding)        (None, 26, 128)           636416    
_________________________________________________________________
lstm (LSTM)                  (None, 26, 256)           394240    
_________________________________________________________________
time_distributed (TimeDistri (None, 26, 25)            6425      
_________________________________________________________________
activation (Activation)      (None, 26, 25)            0         
=================================================================
Total params: 1,037,081
Trainable params: 1,037,081
Non-trainable params: 0
_________________________________________________________________
```

In [382…

```
history2 = model2.fit(train_sentences_x, cat_train_upos_y, batch_size=128, epo
```

```
Epoch 1/40
20/20 [==============================] - 3s 47ms/step - loss: 2.3616 - accurac
y: 0.6554 - val_loss: 0.8219 - val_accuracy: 0.7895
Epoch 2/40
20/20 [==============================] - 0s 18ms/step - loss: 0.7883 - accurac
y: 0.7927 - val_loss: 0.6988 - val_accuracy: 0.8026
Epoch 3/40
20/20 [==============================] - 0s 18ms/step - loss: 0.6698 - accurac
y: 0.7962 - val_loss: 0.6309 - val_accuracy: 0.8007
Epoch 4/40
20/20 [==============================] - 0s 18ms/step - loss: 0.5735 - accurac
y: 0.7995 - val_loss: 0.5817 - val_accuracy: 0.8297
Epoch 5/40
20/20 [==============================] - 0s 18ms/step - loss: 0.5231 - accurac
y: 0.8265 - val_loss: 0.5350 - val_accuracy: 0.8589
Epoch 6/40
20/20 [==============================] - 0s 17ms/step - loss: 0.4628 - accurac
```

```
y: 0.8576 - val_loss: 0.4953 - val_accuracy: 0.8832
Epoch 7/40
20/20 [==============================] - 0s 17ms/step - loss: 0.4215 - accurac
y: 0.9012 - val_loss: 0.4528 - val_accuracy: 0.8931
Epoch 8/40
20/20 [==============================] - 0s 18ms/step - loss: 0.3690 - accurac
y: 0.9114 - val_loss: 0.4109 - val_accuracy: 0.8963
Epoch 9/40
20/20 [==============================] - 0s 18ms/step - loss: 0.3317 - accurac
y: 0.9153 - val_loss: 0.3776 - val_accuracy: 0.9018
Epoch 10/40
20/20 [==============================] - 0s 18ms/step - loss: 0.3045 - accurac
y: 0.9171 - val_loss: 0.3523 - val_accuracy: 0.9040
Epoch 11/40
20/20 [==============================] - 0s 18ms/step - loss: 0.2796 - accurac
y: 0.9202 - val_loss: 0.3356 - val_accuracy: 0.9055
Epoch 12/40
20/20 [==============================] - 0s 17ms/step - loss: 0.2596 - accurac
y: 0.9244 - val_loss: 0.3198 - val_accuracy: 0.9103
Epoch 13/40
20/20 [==============================] - 0s 17ms/step - loss: 0.2341 - accurac
y: 0.9324 - val_loss: 0.3004 - val_accuracy: 0.9168
Epoch 14/40
20/20 [==============================] - 0s 17ms/step - loss: 0.2049 - accurac
y: 0.9407 - val_loss: 0.2820 - val_accuracy: 0.9228
Epoch 15/40
20/20 [==============================] - 0s 17ms/step - loss: 0.1901 - accurac
y: 0.9428 - val_loss: 0.2601 - val_accuracy: 0.9269
Epoch 16/40
20/20 [==============================] - 0s 17ms/step - loss: 0.1607 - accurac
y: 0.9489 - val_loss: 0.2472 - val_accuracy: 0.9298
Epoch 17/40
20/20 [==============================] - 0s 17ms/step - loss: 0.1423 - accurac
y: 0.9531 - val_loss: 0.2272 - val_accuracy: 0.9352
Epoch 18/40
20/20 [==============================] - 0s 17ms/step - loss: 0.1230 - accurac
y: 0.9588 - val_loss: 0.2202 - val_accuracy: 0.9384
Epoch 19/40
20/20 [==============================] - 0s 17ms/step - loss: 0.1070 - accurac
y: 0.9647 - val_loss: 0.2086 - val_accuracy: 0.9444
Epoch 20/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0901 - accurac
y: 0.9725 - val_loss: 0.1939 - val_accuracy: 0.9516
Epoch 21/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0784 - accurac
y: 0.9794 - val_loss: 0.1814 - val_accuracy: 0.9580
Epoch 22/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0639 - accurac
y: 0.9868 - val_loss: 0.1736 - val_accuracy: 0.9603
Epoch 23/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0538 - accurac
y: 0.9889 - val_loss: 0.1678 - val_accuracy: 0.9608
Epoch 24/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0455 - accurac
y: 0.9910 - val_loss: 0.1615 - val_accuracy: 0.9627
Epoch 25/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0393 - accurac
y: 0.9918 - val_loss: 0.1530 - val_accuracy: 0.9634
Epoch 26/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0335 - accurac
y: 0.9933 - val_loss: 0.1533 - val_accuracy: 0.9632
Epoch 27/40
20/20 [==============================] - 0s 18ms/step - loss: 0.0308 - accurac
y: 0.9937 - val_loss: 0.1512 - val_accuracy: 0.9631
Epoch 28/40
```

```
20/20 [==============================] - 0s 17ms/step - loss: 0.0288 - accurac
y: 0.9936 - val_loss: 0.1481 - val_accuracy: 0.9644
Epoch 29/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0244 - accurac
y: 0.9946 - val_loss: 0.1501 - val_accuracy: 0.9633
Epoch 30/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0221 - accurac
y: 0.9954 - val_loss: 0.1487 - val_accuracy: 0.9641
Epoch 31/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0205 - accurac
y: 0.9954 - val_loss: 0.1495 - val_accuracy: 0.9635
Epoch 32/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0187 - accurac
y: 0.9957 - val_loss: 0.1503 - val_accuracy: 0.9638
Epoch 33/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0175 - accurac
y: 0.9960 - val_loss: 0.1485 - val_accuracy: 0.9645
Epoch 34/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0172 - accurac
y: 0.9961 - val_loss: 0.1479 - val_accuracy: 0.9645
Epoch 35/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0165 - accurac
y: 0.9958 - val_loss: 0.1461 - val_accuracy: 0.9644
Epoch 36/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0152 - accurac
y: 0.9967 - val_loss: 0.1446 - val_accuracy: 0.9650
Epoch 37/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0139 - accurac
y: 0.9967 - val_loss: 0.1448 - val_accuracy: 0.9651
Epoch 38/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0130 - accurac
y: 0.9969 - val_loss: 0.1463 - val_accuracy: 0.9649
Epoch 39/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0128 - accurac
y: 0.9970 - val_loss: 0.1481 - val_accuracy: 0.9642
Epoch 40/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0116 - accurac
```

In [383…

```
scores = model2.evaluate(test_sentences_x, cat_test_upos_y)
print(f"{model2.metrics_names[1]}: {scores[1] * 100}")
```

```
20/20 [==============================] - 0s 5ms/step - loss: 0.1455 - accurac
y: 0.9656
accuracy: 96.56442403793335
```

In [498…

```
model2.save('pos_tagger.h5')
```

Observations:

We have built two models to predict the upos (Universal Parts Of Speech) tag for each word in a sentence from the dataset. The first model is built with a simple/vanilla RNN. After training, we see that the accuracy received is 96.0512%. The second model is built with a Bidirectional LSTM and we have received an accuracy of 96.5644% on validation data. Both the models have a time distributed layer as we need temporal slicing.

We see that model 2 is performing better than model 1 i.e, the Bidirectional LSTM model performs better than a simple RNN model. LSTM is able to capture long term dependencies. This means if a particular sentence is long, LSTM is able to capture the essence better than an simple RNN model. If the dataset is larger and more complex with many long sentences, RNN will eventually fail to perform well but LSTM can be tuned as it is flexible and able to capture more complex patterns in the data.

Now we will alter the sentences a bit and evaluate the model performance

In [196…

```
#we will remove all words with length less than or equal to two
```

In [424…

```
train_sentences_augmented = []
all_indices = []

for sentence in train_sentences:
    current_sentence = []
    current_index = []
    for index, word in enumerate(sentence):
        if len(word) > 3:
            current_sentence.append(word)
            current_index.append(index)
    train_sentences_augmented.append(current_sentence)
    all_indices.append(current_index)
```

In [458…

```
train_sentences_augmented[0:5]
```

Out[458…

```
[['చదువు', 'తెలివిని', 'పెంచుతుంది'],
 ['పోలీసుల', 'పడిండు'],
 ['రెండో', 'నిండేలోపల', 'మెదడు', 'బాగా', 'పెరుగుతుంది'],
 ['విత్తనాలు', 'వ్యాధులను', 'ఝాLL'],
 ['కడుపులో', 'తిరుగుతూ']]
```

In [459…

```
train_upos_augmented = []
for indices, upos in zip(all_indices, train_upos):
    train_upos_augmented.append(list(np.array(upos)[indices]))
```

In [460…

```
train_upos_augmented[0:5]
```

Out[460…

```
[['n', 'n', 'v'],
 ['n', 'unk'],
 ['adj', 'v', 'n', 'avy', 'v'],
 ['n', 'n', 'unk'],
 ['n', 'v']]
```

In [469…

```
words_aug, upos_aug = set([]), set([])

for sentence in train_sentences_augmented:
    for word in sentence:
        words_aug.add(word.lower())

for tag in train_upos_augmented:
    for t in tag:
        upos_aug.add(t)

word2index_aug = {w: i + 2 for i, w in enumerate(list(words_aug))}
word2index_aug['-PAD-'] = 0  # The special value used for padding
word2index_aug['-OOV-'] = 1  # The special value used for OOVs

upos2index_aug = {t: i + 1 for i, t in enumerate(list(upos_aug))}
upos2index_aug['-PAD-'] = 0
upos2index_aug['-OOV-'] = 1
```

In [462…

```
len(words_aug)
```

Out[462…

```
4733
```

In [463…

```
len(upos_aug)
```

Out[463…

```
21
```

In [470…

```
train_sentences_x, test_sentences_x, train_upos_y, test_upos_y = [], [], [],

for sentence in train_sentences:
    sentence_int = []
    for word in sentence:
        try:
            sentence_int.append(word2index_aug[word.lower()])
        except KeyError:
            sentence_int.append(word2index_aug['-OOV-'])
    train_sentences_x.append(sentence_int)

for sentence in test_sentences:
    sentence_int = []
    for word in sentence:
        try:
            sentence_int.append(word2index_aug[word.lower()])
        except KeyError:
            sentence_int.append(word2index_aug['-OOV-'])
    test_sentences_x.append(sentence_int)

for sentence in train_upos:
    sentence_int = []
    for word in sentence:
        try:
            sentence_int.append(upos2index_aug[word.lower()])
        except KeyError:
            sentence_int.append(upos2index_aug['-OOV-'])
    train_upos_y.append(sentence_int)

for sentence in test_upos:
    sentence_int = []
    for word in sentence:
        try:
            sentence_int.append(upos2index_aug[word.lower()])
        except KeyError:
            sentence_int.append(upos2index_aug['-OOV-'])
    test_upos_y.append(sentence_int)
```

In [471…

```
MAX_LENGTH = len(max(train_sentences_x, key=len))
print(MAX_LENGTH)
```

```
26
```

In [472…

```
train_sentences_x = pad_sequences(train_sentences_x, maxlen=MAX_LENGTH, paddi
test_sentences_x = pad_sequences(test_sentences_x, maxlen=MAX_LENGTH, padding=
train_upos_y = pad_sequences(train_upos_y, maxlen=MAX_LENGTH, padding='post')
test_upos_y = pad_sequences(test_upos_y, maxlen=MAX_LENGTH, padding='post')
```

In [473…

```
cat_train_upos_y = to_categorical(train_upos_y, len(upos2index))
cat_test_upos_y = to_categorical(test_upos_y, len(upos2index))
```

Model 1 - Recurrent Neural Network

In [480…

```
tf.keras.backend.clear_session()

input_layer_1_aug = Input(shape=(MAX_LENGTH,))
embedding_1_aug = Embedding(input_dim=len(word2index), output_dim=100)(input_
rnn_aug = SimpleRNN(100, return_sequences=True)(embedding_1_aug)
output_1_aug = TimeDistributed(Dense(len(upos2index)))(rnn_aug)
activation_1_aug = Activation('softmax')(output_1_aug)
```

In [481…

```
model1_aug = Model(inputs=[input_layer_1_aug], outputs=[activation_1_aug])
```

In [482…

```
model1_aug.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
```

In [483…

```
model1_aug.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 26)]              0         
_________________________________________________________________
embedding (Embedding)        (None, 26, 100)           497200    
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 26, 100)           20100     
_________________________________________________________________
time_distributed (TimeDistri (None, 26, 25)            2525      
_________________________________________________________________
activation (Activation)      (None, 26, 25)            0         
=================================================================
Total params: 519,825
Trainable params: 519,825
Non-trainable params: 0
_________________________________________________________________
```

In [484…

```
history1_aug = model1_aug.fit(train_sentences_x, cat_train_upos_y, batch_size=
```

```
Epoch 1/40
20/20 [==============================] - 2s 64ms/step - loss: 2.0684 - accurac
y: 0.6089 - val_loss: 0.8437 - val_accuracy: 0.7900
Epoch 2/40
20/20 [==============================] - 1s 47ms/step - loss: 0.7723 - accurac
y: 0.7938 - val_loss: 0.6986 - val_accuracy: 0.8007
Epoch 3/40
20/20 [==============================] - 1s 48ms/step - loss: 0.6708 - accurac
y: 0.8027 - val_loss: 0.6425 - val_accuracy: 0.8000
Epoch 4/40
20/20 [==============================] - 1s 49ms/step - loss: 0.6012 - accurac
y: 0.8075 - val_loss: 0.5812 - val_accuracy: 0.8230
Epoch 5/40
20/20 [==============================] - 1s 47ms/step - loss: 0.5384 - accurac
y: 0.8332 - val_loss: 0.5263 - val_accuracy: 0.8494
Epoch 6/40
20/20 [==============================] - 1s 50ms/step - loss: 0.4750 - accurac
y: 0.8673 - val_loss: 0.4886 - val_accuracy: 0.8707
Epoch 7/40
20/20 [==============================] - 1s 47ms/step - loss: 0.4234 - accurac
y: 0.9121 - val_loss: 0.4497 - val_accuracy: 0.8835
Epoch 8/40
```

```
20/20 [==============================] - 1s 48ms/step - loss: 0.3748 - accurac
y: 0.9285 - val_loss: 0.4137 - val_accuracy: 0.8869
Epoch 9/40
20/20 [==============================] - 1s 47ms/step - loss: 0.3214 - accurac
y: 0.9380 - val_loss: 0.3944 - val_accuracy: 0.8913
Epoch 10/40
20/20 [==============================] - 1s 48ms/step - loss: 0.2835 - accurac
y: 0.9419 - val_loss: 0.3672 - val_accuracy: 0.8993
Epoch 11/40
20/20 [==============================] - 1s 48ms/step - loss: 0.2420 - accurac
y: 0.9485 - val_loss: 0.3476 - val_accuracy: 0.9037
Epoch 12/40
20/20 [==============================] - 1s 46ms/step - loss: 0.2096 - accurac
y: 0.9528 - val_loss: 0.3358 - val_accuracy: 0.9055
Epoch 13/40
20/20 [==============================] - 1s 48ms/step - loss: 0.1867 - accurac
y: 0.9547 - val_loss: 0.3267 - val_accuracy: 0.9070
Epoch 14/40
20/20 [==============================] - 1s 47ms/step - loss: 0.1628 - accurac
y: 0.9600 - val_loss: 0.3174 - val_accuracy: 0.9125
Epoch 15/40
20/20 [==============================] - 1s 47ms/step - loss: 0.1457 - accurac
y: 0.9656 - val_loss: 0.3128 - val_accuracy: 0.9142
Epoch 16/40
20/20 [==============================] - 1s 48ms/step - loss: 0.1318 - accurac
y: 0.9688 - val_loss: 0.3024 - val_accuracy: 0.9164
Epoch 17/40
20/20 [==============================] - 1s 48ms/step - loss: 0.1185 - accurac
y: 0.9735 - val_loss: 0.3074 - val_accuracy: 0.9166
Epoch 18/40
20/20 [==============================] - 1s 46ms/step - loss: 0.1054 - accurac
y: 0.9765 - val_loss: 0.2925 - val_accuracy: 0.9199
Epoch 19/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0965 - accurac
y: 0.9789 - val_loss: 0.2892 - val_accuracy: 0.9208
Epoch 20/40
20/20 [==============================] - 1s 49ms/step - loss: 0.0883 - accurac
y: 0.9811 - val_loss: 0.2895 - val_accuracy: 0.9210
Epoch 21/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0797 - accurac
y: 0.9827 - val_loss: 0.2907 - val_accuracy: 0.9210
Epoch 22/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0737 - accurac
y: 0.9838 - val_loss: 0.2895 - val_accuracy: 0.9219
Epoch 23/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0705 - accurac
y: 0.9834 - val_loss: 0.2867 - val_accuracy: 0.9227
Epoch 24/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0648 - accurac
y: 0.9851 - val_loss: 0.2904 - val_accuracy: 0.9227
Epoch 25/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0592 - accurac
y: 0.9866 - val_loss: 0.2866 - val_accuracy: 0.9235
Epoch 26/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0566 - accurac
y: 0.9863 - val_loss: 0.2876 - val_accuracy: 0.9233
Epoch 27/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0518 - accurac
y: 0.9875 - val_loss: 0.2829 - val_accuracy: 0.9243
Epoch 28/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0497 - accurac
y: 0.9874 - val_loss: 0.2865 - val_accuracy: 0.9233
Epoch 29/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0454 - accurac
y: 0.9890 - val_loss: 0.2847 - val_accuracy: 0.9238
```

```
Epoch 30/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0457 - accurac
y: 0.9882 - val_loss: 0.2849 - val_accuracy: 0.9239
Epoch 31/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0411 - accurac
y: 0.9897 - val_loss: 0.2868 - val_accuracy: 0.9239
Epoch 32/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0417 - accurac
y: 0.9889 - val_loss: 0.2831 - val_accuracy: 0.9250
Epoch 33/40
20/20 [==============================] - 1s 49ms/step - loss: 0.0388 - accurac
y: 0.9900 - val_loss: 0.2852 - val_accuracy: 0.9252
Epoch 34/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0374 - accurac
y: 0.9904 - val_loss: 0.2844 - val_accuracy: 0.9256
Epoch 35/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0368 - accurac
y: 0.9903 - val_loss: 0.2811 - val_accuracy: 0.9263
Epoch 36/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0334 - accurac
y: 0.9910 - val_loss: 0.2837 - val_accuracy: 0.9258
Epoch 37/40
20/20 [==============================] - 1s 47ms/step - loss: 0.0331 - accurac
y: 0.9910 - val_loss: 0.2833 - val_accuracy: 0.9256
Epoch 38/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0326 - accurac
y: 0.9909 - val_loss: 0.2826 - val_accuracy: 0.9258
Epoch 39/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0311 - accurac
y: 0.9913 - val_loss: 0.2830 - val_accuracy: 0.9265
Epoch 40/40
20/20 [==============================] - 1s 48ms/step - loss: 0.0303 - accurac
y: 0.9916 - val_loss: 0.2841 - val_accuracy: 0.9262
```

In [485…

```
scores = model1_aug.evaluate(test_sentences_x, cat_test_upos_y)
print(f"{model1_aug.metrics_names[1]}: {scores[1] * 100}")
```

```
20/20 [==============================] - 0s 7ms/step - loss: 0.2841 - accurac
y: 0.9262
accuracy: 92.61562824249268
```

Model 2 - Long Short Term Memory

In [491…

```
tf.keras.backend.clear_session()

input_layer_2_aug = Input(shape=(MAX_LENGTH,))
embedding_2_aug = Embedding(input_dim=len(word2index), output_dim=128)(input_
lstm_aug = LSTM(256, return_sequences=True)(embedding_2_aug)
output_2_aug = TimeDistributed(Dense(len(upos2index)))(lstm_aug)
activation_2_aug = Activation('softmax')(output_2_aug)
```

In [492…

```
model2_aug = Model(inputs=[input_layer_2_aug], outputs=[activation_2_aug])
```

In [493…

```
model2_aug.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
```

In [494…

```
model2_aug.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 26)]              0         
_________________________________________________________________
embedding (Embedding)        (None, 26, 128)           636416    
_________________________________________________________________
lstm (LSTM)                  (None, 26, 256)           394240    
_________________________________________________________________
time_distributed (TimeDistri (None, 26, 25)            6425      
_________________________________________________________________
activation (Activation)      (None, 26, 25)            0         
=================================================================
Total params: 1,037,081
Trainable params: 1,037,081
Non-trainable params: 0
_________________________________________________________________
```

In [495…

```
history2_aug = model2_aug.fit(train_sentences_x, cat_train_upos_y, batch_size=
```

```
Epoch 1/40
20/20 [==============================] - 3s 47ms/step - loss: 2.3059 - accurac
y: 0.6651 - val_loss: 0.9246 - val_accuracy: 0.7895
Epoch 2/40
20/20 [==============================] - 0s 18ms/step - loss: 0.7673 - accurac
y: 0.7939 - val_loss: 0.6716 - val_accuracy: 0.7910
Epoch 3/40
20/20 [==============================] - 0s 18ms/step - loss: 0.6419 - accurac
y: 0.7978 - val_loss: 0.6159 - val_accuracy: 0.7932
Epoch 4/40
20/20 [==============================] - 0s 18ms/step - loss: 0.5692 - accurac
y: 0.8042 - val_loss: 0.5701 - val_accuracy: 0.8221
Epoch 5/40
20/20 [==============================] - 0s 18ms/step - loss: 0.5267 - accurac
y: 0.8331 - val_loss: 0.5314 - val_accuracy: 0.8454
Epoch 6/40
20/20 [==============================] - 0s 17ms/step - loss: 0.4687 - accurac
y: 0.8615 - val_loss: 0.4986 - val_accuracy: 0.8680
Epoch 7/40
20/20 [==============================] - 0s 18ms/step - loss: 0.4239 - accurac
y: 0.8931 - val_loss: 0.4601 - val_accuracy: 0.8767
Epoch 8/40
20/20 [==============================] - 0s 18ms/step - loss: 0.3753 - accurac
y: 0.9062 - val_loss: 0.4611 - val_accuracy: 0.8800
Epoch 9/40
20/20 [==============================] - 0s 18ms/step - loss: 0.3351 - accurac
y: 0.9147 - val_loss: 0.4258 - val_accuracy: 0.8843
Epoch 10/40
20/20 [==============================] - 0s 18ms/step - loss: 0.2965 - accurac
y: 0.9202 - val_loss: 0.4066 - val_accuracy: 0.8885
Epoch 11/40
20/20 [==============================] - 0s 19ms/step - loss: 0.2680 - accurac
y: 0.9244 - val_loss: 0.4068 - val_accuracy: 0.8902
Epoch 12/40
20/20 [==============================] - 0s 18ms/step - loss: 0.2373 - accurac
y: 0.9313 - val_loss: 0.4119 - val_accuracy: 0.8975
Epoch 13/40
20/20 [==============================] - 0s 18ms/step - loss: 0.2182 - accurac
y: 0.9365 - val_loss: 0.4021 - val_accuracy: 0.9010
Epoch 14/40
20/20 [==============================] - 0s 17ms/step - loss: 0.1967 - accurac
y: 0.9411 - val_loss: 0.3980 - val_accuracy: 0.9003
Epoch 15/40
```

```
20/20 [==============================] - 0s 18ms/step - loss: 0.1718 - accurac
y: 0.9457 - val_loss: 0.4108 - val_accuracy: 0.8968
Epoch 16/40
20/20 [==============================] - 0s 17ms/step - loss: 0.1565 - accurac
y: 0.9500 - val_loss: 0.4058 - val_accuracy: 0.8984
Epoch 17/40
20/20 [==============================] - 0s 18ms/step - loss: 0.1387 - accurac
y: 0.9546 - val_loss: 0.4230 - val_accuracy: 0.8983
Epoch 18/40
20/20 [==============================] - 0s 18ms/step - loss: 0.1284 - accurac
y: 0.9571 - val_loss: 0.4114 - val_accuracy: 0.9004
Epoch 19/40
20/20 [==============================] - 0s 18ms/step - loss: 0.1134 - accurac
y: 0.9623 - val_loss: 0.4125 - val_accuracy: 0.9042
Epoch 20/40
20/20 [==============================] - 0s 18ms/step - loss: 0.1061 - accurac
y: 0.9652 - val_loss: 0.4089 - val_accuracy: 0.9083
Epoch 21/40
20/20 [==============================] - 0s 17ms/step - loss: 0.1006 - accurac
y: 0.9682 - val_loss: 0.4171 - val_accuracy: 0.9103
Epoch 22/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0923 - accurac
y: 0.9716 - val_loss: 0.4032 - val_accuracy: 0.9148
Epoch 23/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0874 - accurac
y: 0.9737 - val_loss: 0.4100 - val_accuracy: 0.9163
Epoch 24/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0825 - accurac
y: 0.9755 - val_loss: 0.4011 - val_accuracy: 0.9183
Epoch 25/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0769 - accurac
y: 0.9769 - val_loss: 0.3969 - val_accuracy: 0.9176
Epoch 26/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0711 - accurac
y: 0.9782 - val_loss: 0.3927 - val_accuracy: 0.9229
Epoch 27/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0692 - accurac
y: 0.9789 - val_loss: 0.3855 - val_accuracy: 0.9252
Epoch 28/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0677 - accurac
y: 0.9791 - val_loss: 0.3806 - val_accuracy: 0.9262
Epoch 29/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0621 - accurac
y: 0.9818 - val_loss: 0.3742 - val_accuracy: 0.9259
Epoch 30/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0600 - accurac
y: 0.9814 - val_loss: 0.3685 - val_accuracy: 0.9284
Epoch 31/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0580 - accurac
y: 0.9824 - val_loss: 0.3646 - val_accuracy: 0.9284
Epoch 32/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0553 - accurac
y: 0.9830 - val_loss: 0.3550 - val_accuracy: 0.9281
Epoch 33/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0537 - accurac
y: 0.9829 - val_loss: 0.3592 - val_accuracy: 0.9287
Epoch 34/40
20/20 [==============================] - 0s 18ms/step - loss: 0.0499 - accurac
y: 0.9844 - val_loss: 0.3435 - val_accuracy: 0.9295
Epoch 35/40
20/20 [==============================] - 0s 18ms/step - loss: 0.0475 - accurac
y: 0.9846 - val_loss: 0.3343 - val_accuracy: 0.9318
Epoch 36/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0468 - accurac
y: 0.9853 - val_loss: 0.3344 - val_accuracy: 0.9308
```

```
Epoch 37/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0432 - accurac
y: 0.9862 - val_loss: 0.3341 - val_accuracy: 0.9323
Epoch 38/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0408 - accurac
y: 0.9868 - val_loss: 0.3376 - val_accuracy: 0.9312
Epoch 39/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0417 - accurac
y: 0.9874 - val_loss: 0.3337 - val_accuracy: 0.9313
Epoch 40/40
20/20 [==============================] - 0s 17ms/step - loss: 0.0394 - accurac
```

In [496…

```
scores = model2_aug.evaluate(test_sentences_x, cat_test_upos_y)
print(f"{model2_aug.metrics_names[1]}: {scores[1] * 100}")
```

```
20/20 [==============================] - 0s 5ms/step - loss: 0.3360 - accurac
y: 0.9323
accuracy: 93.2314932346344
```

Observations:

As a part of data augmentation, we have removed words with length less than or equal to two. We have then performed similar preproccessing steps for the augmented data and have used the same models.

We quickly notice that, the validation accuracy is reduced by a considerable margin when compared to model with full data. By comparing the RNN and LSTM models built for augmented data, we see that LSTM again is performing better than simple RNN. The validation accuracy of the simple RNN model is 92.6156% while the validation accuracy for the bidirectional LSTM model is 93.2314%. This shows that LSTM is a better model than a simple RNN.

The same preprocessing techniques and model architecture can be used for predicting xpos tags as well.