Training Project Report

Multi-Level Predictive Language Model

Name: Sahejpal Singh Arneja

Neptun : TJ73O8

Table of Contents

Introduction

Simply speaking, the goal of grammar learning is to learn the right order of words in a sentence. This assumption is the fundamental statement of modern, predictive language models, e.g., USE, BERT, GPT-3. Such models are typically trained by involving transfer learning and also provide high quality language models.

Aim

Most models involve word-based representation. However, some approaches are based on characters and hyphens. The goal of this project is to broaden this scope and involve different modalities, e.g., characters, hyphens, words, compound words, sentences, POS tags, etc. The goal is to learn how to combine the mentioned modalities.

Objective

What we wanted to do was to combine various uni-models and make one mega model which would nlp us to get a more accurate prediction of the next word.

The plan was to

1.Make a Natural Language model that would predict the next character the user will enter

2.Make a Natural Language model that would predict the next word the user will enter

3.Make a POS(Part-of-Speech) tagger model that would tell us what type of word the next word can be (for example, a noun, adverb, conjunction, adjective etc)

Theory

I had chosen LSTMs as the technology I wanted to work on to implement the models

LSTMs

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems.

This is a behaviour required in complex problem domains like machine translation, speech recognition, and more.

LSTMs are a complex area of deep learning. It can be hard to get your hands around what LSTMs are, and how terms like bidirectional and sequence-to-sequence relate to the field.

Advantages of LSTM

- One of the biggest problems of recurrent neural networks is the vanishing gradient problem.
- It happens when the gradient shrinks during backpropagation.
- If it becomes very small, the network stops learning. This mostly happen when long sentences are present.
- LSTM networks address this problem by having an inner memory cell to remember important information or forget others.
- LSTM has a similar flow as a RNN, it processes data and passes information as it propagates forward.
- The difference is in the operations within the cells.

Application of LSTMs

Apart from the answers listed previously, LSTM works well for **time series prediction**. If your data is in a **sequential format** (Time, Sentence, etc...,)

Types of LSTM models based on input and output

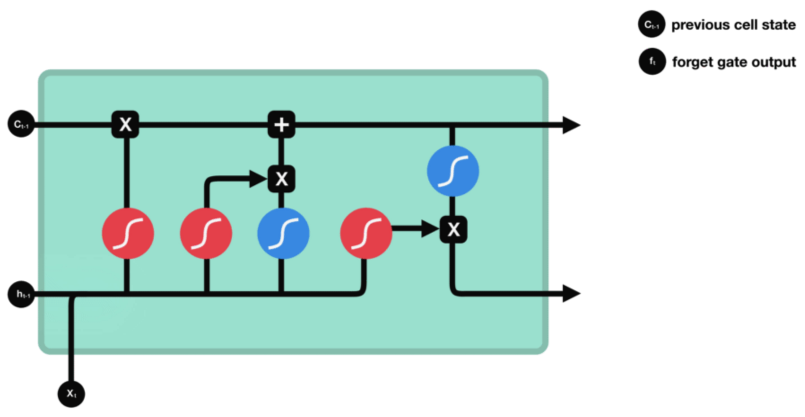
1. **One input to One output** -: Giving labels to image
2. **One input to many outputs**-: Giving description/caption to image
   (description will have sequence of words - many output)
3. **Many inputs to one output** -: Predicting the next word in given incomplete statement
4. **Many inputs to Many outputs**-: Stock market prediction for following days based on past data

Structure of LSTM

LSTM consists of

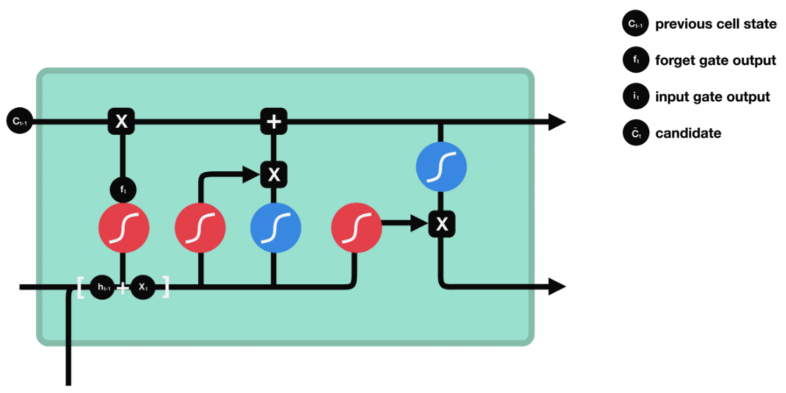
Forget Gate

- Decides what information should be kept or thrown away
- Information from the previous hidden state and from the current input



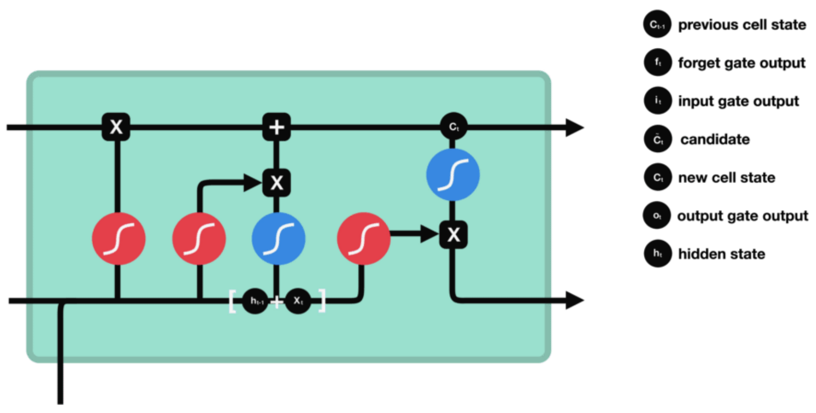
Input Gate

- Decides what information is relevant to add from the current state



Output Gate

- Determines what the next hidden state should be



Frameworks used to create the models

Word Prediction model

- For the Word Prediction model, I used Keras to create the different layers of the model.
- I also used the Spacy tokenizer to get the tokens for the sequences
- To visualize the data pyplot from Matplotlib was used

POS Tagger model

- For the POS Tagging model I used Keras to create the different layers of the sequential model.
- The pre-processing and tokenizing were done using Keras Tokenizer
- The Glove embedding was implemented using Gensim
- Sklearn was used to split the Train and Test data sets
- Matplotlib for visualisation of the metrics

Data Sets

Word Prediction Model Data Set

I found that most of the language prediction model used English Literature books for training the model. In my model I have used 'The Adventures of Sherlock Holmes' and 'Metamorphosis' as the datasets.

POS Tagger Model Data Set

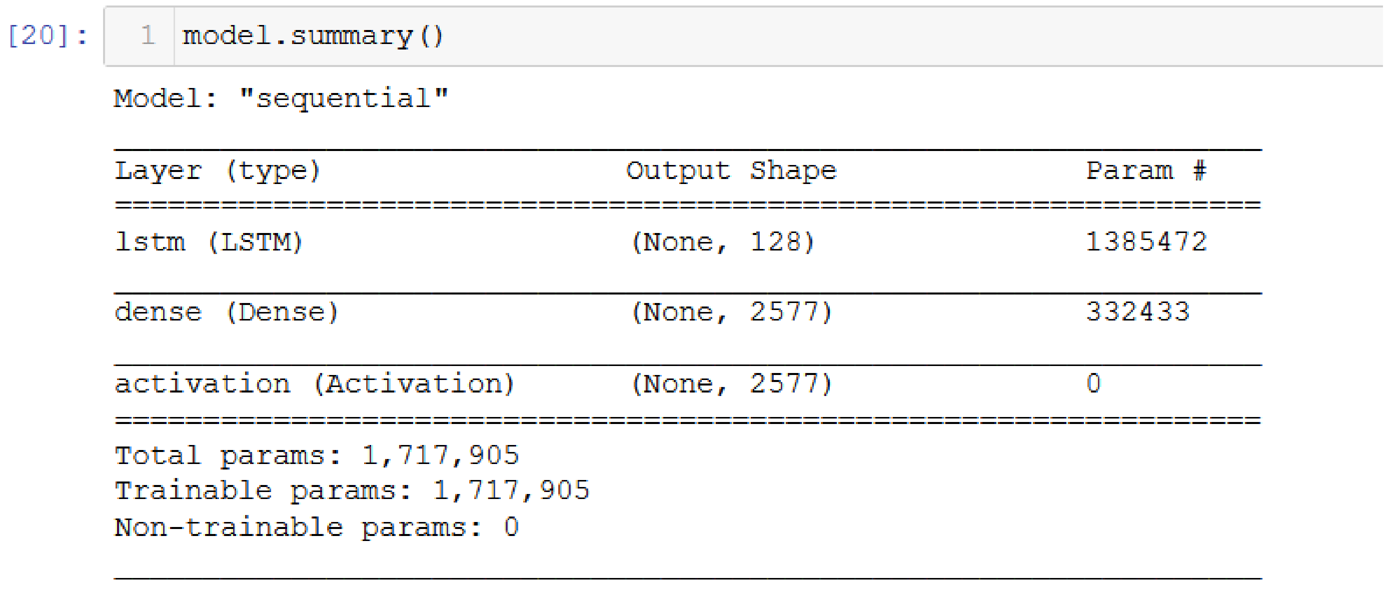
For the POS tagging model I used 3 of pre-tagged data sets for training my model

- Treebank corpus
- Brown Corpus
- Conll Corpus

All of these corpuses are available using nltk

Model Summaries

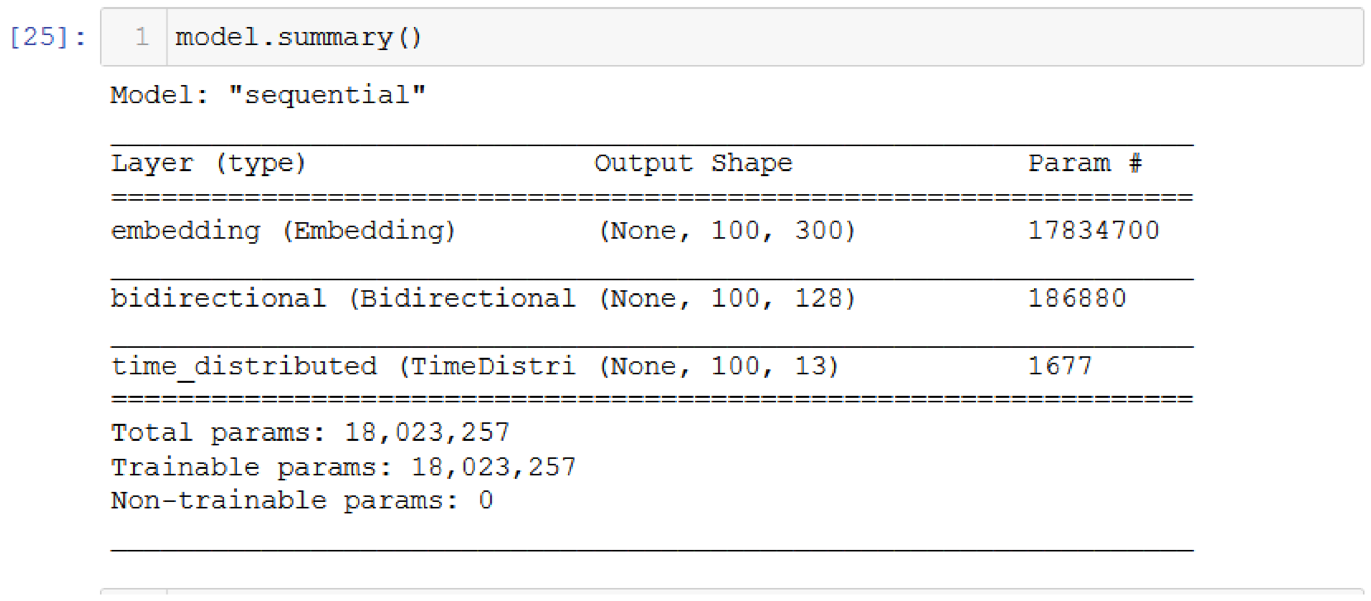
Word Prediction Model Summary



```
[20]:  model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 128)               1385472
_________________________________________________________________
dense (Dense)                (None, 2577)              332433
_________________________________________________________________
activation (Activation)      (None, 2577)              0
=================================================================
Total params: 1,717,905
Trainable params: 1,717,905
Non-trainable params: 0
_________________________________________________________________
```

POS Tagging Model Summary



```
[25]:  model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 300)          17834700
_________________________________________________________________
bidirectional (Bidirectional (None, 100, 128)          186880
_________________________________________________________________
time_distributed (TimeDistri (None, 100, 13)           1677
=================================================================
Total params: 18,023,257
Trainable params: 18,023,257
Non-trainable params: 0
_________________________________________________________________
```

Code Snippets for Word Prediction Model

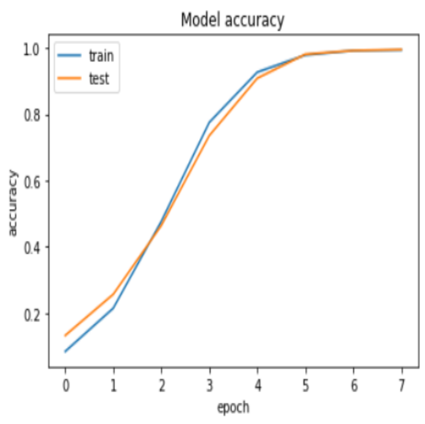
Training for Word Prediction Model

In [21]:
```
optimizer = Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
history = model.fit(X, Y, validation_split=0.05, batch_size=128, epochs=8, shuffle=True).history
```

```
Epoch 1/8
345/345 [==============================] - 37s 99ms/step - loss: 5.7320 - accuracy: 0.0837 - val_loss: 5.0266 - val_a
ccuracy: 0.1320
Epoch 2/8
345/345 [==============================] - 31s 89ms/step - loss: 4.2124 - accuracy: 0.2141 - val_loss: 3.8195 - val_a
ccuracy: 0.2561
Epoch 3/8
345/345 [==============================] - 31s 90ms/step - loss: 2.4435 - accuracy: 0.4763 - val_loss: 2.4142 - val_a
ccuracy: 0.4636
Epoch 4/8
345/345 [==============================] - 32s 91ms/step - loss: 0.9825 - accuracy: 0.7752 - val_loss: 1.1498 - val_a
ccuracy: 0.7352
Epoch 5/8
345/345 [==============================] - 31s 89ms/step - loss: 0.3480 - accuracy: 0.9269 - val_loss: 0.4154 - val_a
ccuracy: 0.9086
Epoch 6/8
345/345 [==============================] - 31s 90ms/step - loss: 0.1262 - accuracy: 0.9786 - val_loss: 0.1321 - val_a
ccuracy: 0.9810
Epoch 7/8
345/345 [==============================] - 31s 90ms/step - loss: 0.0579 - accuracy: 0.9920 - val_loss: 0.0624 - val_a
ccuracy: 0.9922
Epoch 8/8
345/345 [==============================] - 31s 89ms/step - loss: 0.0416 - accuracy: 0.9937 - val_loss: 0.0378 - val_a
ccuracy: 0.9953
```

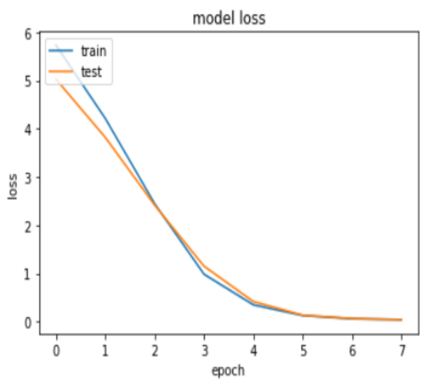
Accuracy Plot

```
Out[22]: <matplotlib.legend.Legend at 0x20cef48cd90>
```



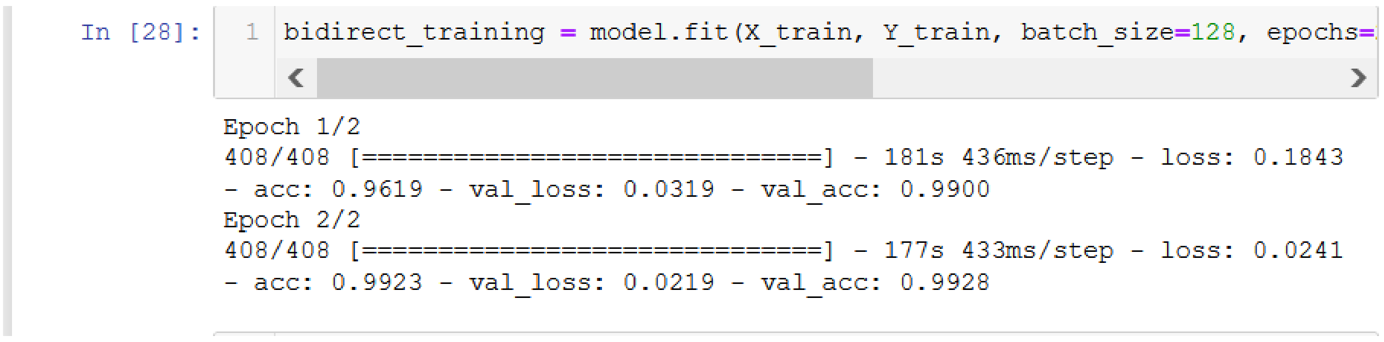
Loss Plot

```
Out[24]: <matplotlib.legend.Legend at 0x20cef4d1220>
```



Code Snippets for POS Tagging Model

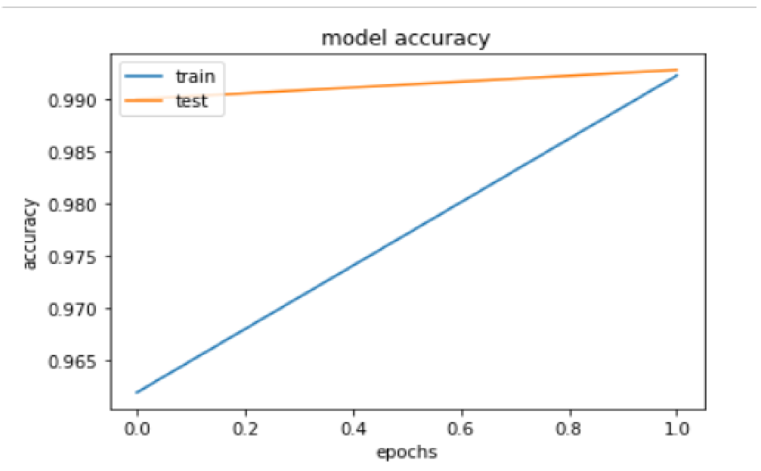
Training for POs Tagging Model



```
In [28]: bidirect_training = model.fit(X_train, Y_train, batch_size=128, epochs=
```

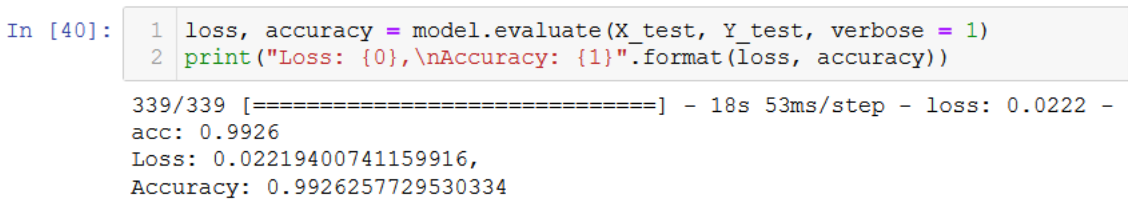
```
Epoch 1/2
408/408 [==============================] - 181s 436ms/step - loss: 0.1843
- acc: 0.9619 - val_loss: 0.0319 - val_acc: 0.9900
Epoch 2/2
408/408 [==============================] - 177s 433ms/step - loss: 0.0241
- acc: 0.9923 - val_loss: 0.0219 - val_acc: 0.9928
```

Accuracy Plot



Evaluation for POS tagger

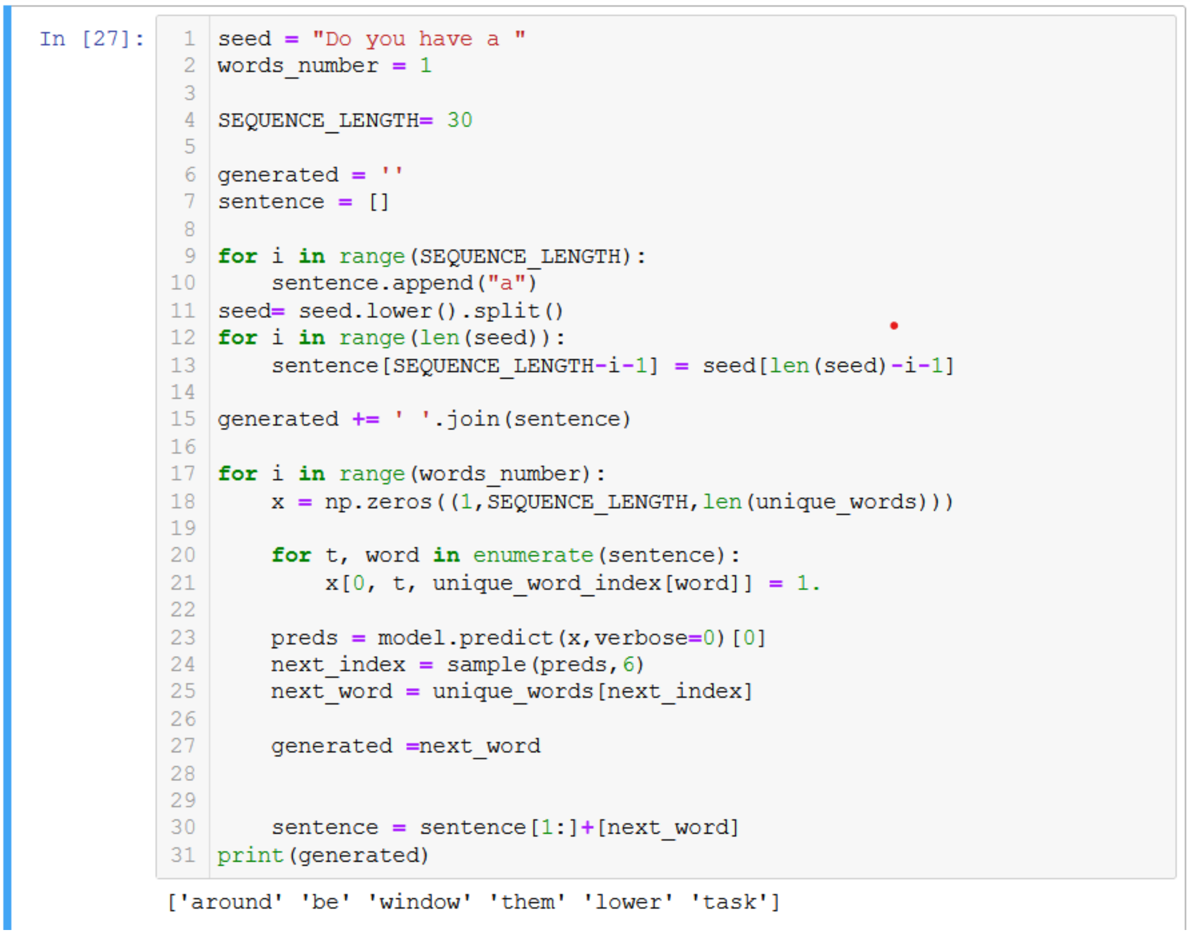
epocns



```
In [40]: loss, accuracy = model.evaluate(X_test, Y_test, verbose = 1)
         print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

```
339/339 [==============================] - 18s 53ms/step - loss: 0.0222 -
acc: 0.9926
Loss: 0.022194007411599 16,
Accuracy: 0.9926257729530334
```

Testing the Text Generation



```
In [27]:
seed = "Do you have a "
words_number = 1

SEQUENCE_LENGTH= 30

generated = ''
sentence = []

for i in range(SEQUENCE_LENGTH):
    sentence.append("a")
seed= seed.lower().split()
for i in range(len(seed)):
    sentence[SEQUENCE_LENGTH-i-1] = seed[len(seed)-i-1]

generated += ' '.join(sentence)

for i in range(words_number):
    x = np.zeros((1,SEQUENCE_LENGTH,len(unique_words)))

    for t, word in enumerate(sentence):
        x[0, t, unique_word_index[word]] = 1.

    preds = model.predict(x,verbose=0)[0]
    next_index = sample(preds,6)
    next_word = unique_words[next_index]

    generated =next_word


    sentence = sentence[1:]+[next_word]
print(generated)
```

```
['around' 'be' 'window' 'them' 'lower' 'task']
```

We can see that upon adding a seed sentence the model predicts 6 possible words that can be the next one.

Result

We can see that the models can work together an generate the next word. I would further like to increase the scope of the topic an understand the basics of all the technologies used and other applications of multi-level models

References

- https://arxiv.org/abs/1803.11175
- https://arxiv.org/abs/1705.02364
- https://keras.io/guides/making_new_layers_and_models_via_subclassing/
- https://arxiv.org/abs/1409.0473