Harsh Shinde	17u008	432014
Shardul Nazirkar	17u024	432013
Prathamesh Mundada	17u390	432015

AML Mini Project

NEXT WORD PREDICTION USING LSTM

Aim:

To analyze the given test data set and predict the next word considering the last word of a particular sentence.

Objective:

To build a next word predictor using lstm algorithm.

Purpose:

The objective of this project is to predict what could be the next word that you are planning to type. Next word prediction can be used to give a word suggestion based on last typed word in order to save the time consumed because of typing. It is similar to the predictive keypads of what's app, google searches, Instagram, emails and much more.

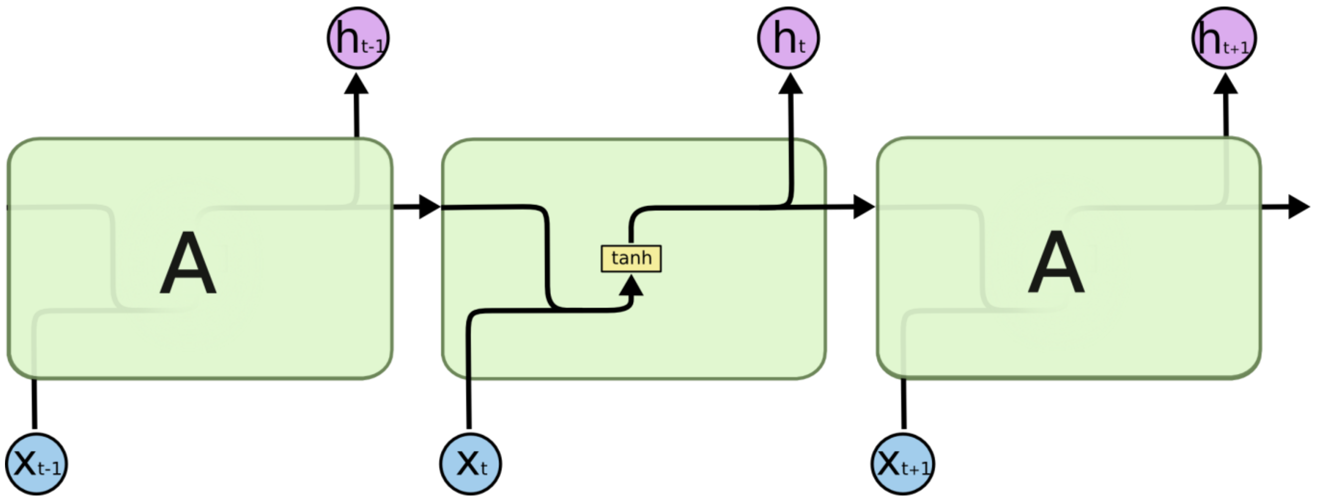
Theory:

LSTM:

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by **Hochreiter & Schmidhuber (1997),** and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

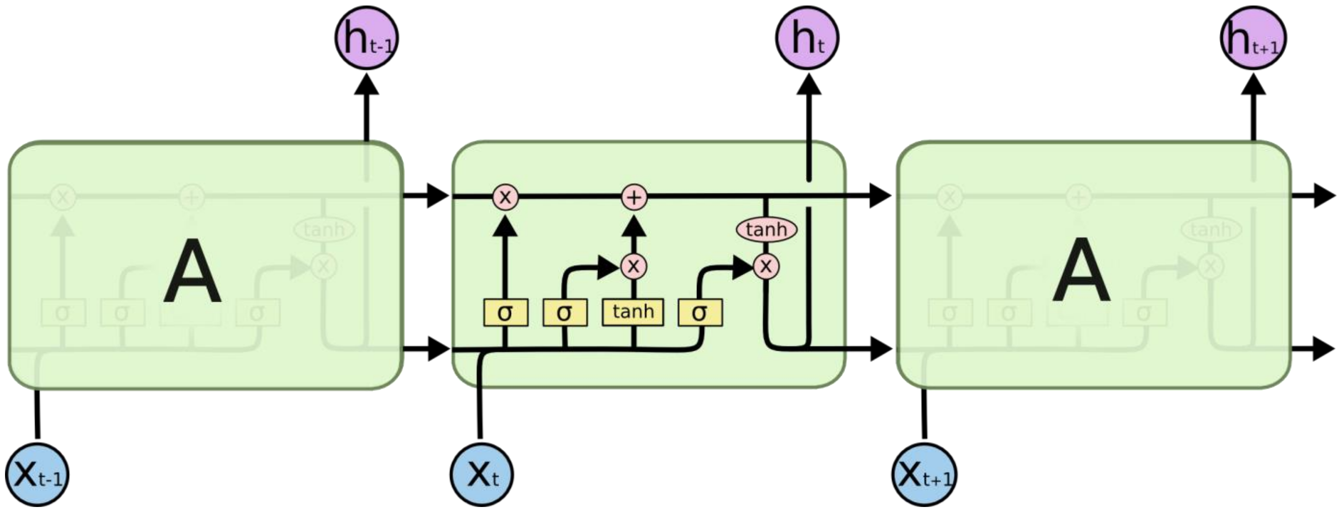
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

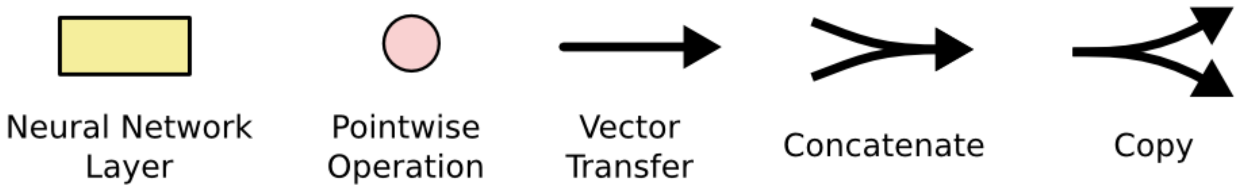


The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

Input & output parameters involved:

Input parameters involve creating an embedding layer to specify input and Output dimensions. Input length will be one as prediction will be made on exactly one word.

Output parametres has the predicted word there by decreasing the loss function through 150 epochs.

Steps involved :

1. Pre-processing the Dataset.
2. Creating the Model.
3. Compile and fit the model.
4. Prediction.

Program:

1. **Importing libraries:**

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
import pickle
import numpy as np
import os
```

2. **Reading dataset:**

```
file = open("metamorphosis_clean.txt", "r", encoding = "utf8")
lines = []

for i in file:
    lines.append(i)

print("The First Line: ", lines[0])
print("The Last Line: ", lines[-1])
```

3. Cleaning the data

```
data = ""

for i in lines:

    data = ' '. join(lines)

data = data.replace('\n', '').replace('\r', '').replace('\ufeff', '')

data[:360]

import string


translator = str.maketrans(string.punctuation, ' '*len(string.punctuation)) #map punc
tuation to space

new_data = data.translate(translator)

new_data[:500]

z = []

for i in data.split():

    if i not in z:

        z.append(i)

data = ' '.join(z)

data[:500]
```

4 . **Tokenization:**

```
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts([data])

    # saving the tokenizer for predict function.
    pickle.dump(tokenizer, open('tokenizer1.pkl', 'wb'))

    sequence_data = tokenizer.texts_to_sequences([data])[0]
    sequence_data[:10]

    vocab_size = len(tokenizer.word_index) + 1
    print(vocab_size)

      sequences = []
  for i in range(1, len(sequence_data)):
  words = sequence_data[i-1:i+1]
  sequences.append(words)

print("The Length of sequences are: ", len(sequences))
sequences = np.array(sequences)
sequences[:10]

X = []
y = []

for i in sequences:
  X.append(i[0])
  y.append(i[1])

X = np.array(X)
y = np.array(y)

print("The Data is: ", X[:5])
print("The responses are: ", y[:5])
```

```
y = to_categorical(y, num_classes=vocab_size)
y[:5]
```

5. Creating Model:

```
model = Sequential()
model.add(Embedding(vocab_size, 10, input_length=1))
model.add(LSTM(1000, return_sequences=True))
model.add(LSTM(1000))
model.add(Dense(1000, activation="relu"))
model.add(Dense(vocab_size, activation="softmax"))
model.summary()
```

6. Callbacks:

```
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import TensorBoard

checkpoint = ModelCheckpoint("nextword1.h5", monitor='loss', verbose=1,
    save_best_only=True, mode='auto')

reduce = ReduceLROnPlateau(monitor='loss', factor=0.2, patience=3, min_lr=0.0001,
verbose = 1)

logdir='logsnextword1'
tensorboard_Visualization = TensorBoard(log_dir=logdir)
```

7. Model compilation:

```
model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.001))
```

8. Fit The Model:

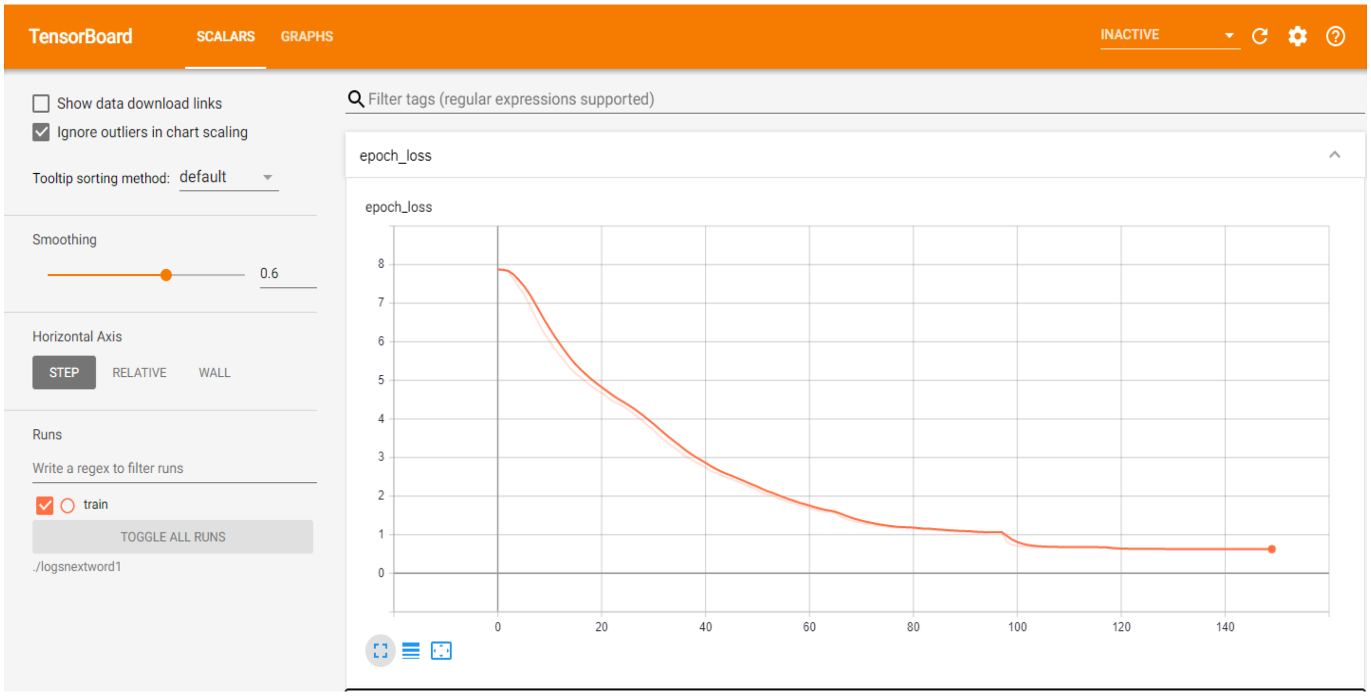
```
model.fit(X, y, epochs=150, batch_size=64, callbacks=[checkpoint, reduce,
tensorboard_Visualization])
```

9. Graph:

```
from IPython.display import Image
pil_img = Image(filename='graph1.png')
display(pil_img)
```

Output Snapshots:

Line Graph



Show data download links ☐
Ignore outliers in chart scaling ☑
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: [✓ STEP] RELATIVE WALL
Runs
Write a regex to filter runs
☑ train
./logsnextword1

LSTM MODEL CREATION :

```
Model: "sequential"
_________________________________________________________________
Layer (type)              Output Shape          Param #
=================================================================
embedding (Embedding)       (None, 1, 10)          26170
```

```
_________________________________________________________________
lstm (LSTM)              (None, 1, 1000)         4044000
_________________________________________________________________
lstm_1 (LSTM)             (None, 1000)           8004000
_________________________________________________________________
dense (Dense)            (None, 1000)            1001000
_________________________________________________________________
dense_1 (Dense)           (None, 2617)           2619617
=================================================================
Total params: 15,694,787
Trainable params: 15,694,787
Non-trainable params: 0
_________________________________________________________________
```

LSTM FITTING THE MODEL:

```
Train on 3889 samples
Epoch 1/150
3712/3889 [===========================>..] - ETA: 0s - loss: 7.8752
Epoch 00001: loss improved from inf to 7.87560, saving model to nextword1.h5
3889/3889 [==============================] - 5s 1ms/sample - loss: 7.8756
Epoch 2/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.8587
Epoch 00002: loss improved from 7.87560 to 7.86009, saving model to nextword1.h5
3889/3889 [==============================] - 1s 331us/sample - loss: 7.8601
Epoch 3/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.8187
Epoch 00003: loss improved from 7.86009 to 7.81623, saving model to nextword1.h5
3889/3889 [==============================] - 1s 372us/sample - loss: 7.8162
Epoch 4/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.6399
Epoch 00004: loss improved from 7.81623 to 7.63961, saving model to nextword1.h5
3889/3889 [==============================] - 1s 327us/sample - loss: 7.6396
Epoch 5/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.4280
Epoch 00005: loss improved from 7.63961 to 7.42898, saving model to nextword1.h5
3889/3889 [==============================] - 1s 363us/sample - loss: 7.4290
Epoch 6/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.2234
Epoch 00006: loss improved from 7.42898 to 7.23395, saving model to nextword1.h5
3889/3889 [==============================] - 1s 335us/sample - loss: 7.2339
Epoch 7/150
3648/3889 [===========================>..] - ETA: 0s - loss: 6.9898
Epoch 00007: loss improved from 7.23395 to 6.99421, saving model to nextword1.h5
3889/3889 [==============================] - 1s 369us/sample - loss: 6.9942
Epoch 8/150
```

Total training is for 150 Epochs to reduce the loss.

OUTPUTS FOR WORD PREDICTION:

```
Enter your line: dull
Weather

Enter your line: textile
samples
```

Conclusion:

We are able to develop a high-quality next word prediction for the metamorphosis dataset. We are able to reduce the loss significantly in about 150 epochs.