Long Short Term Memory Using Tensorflow and Keras Framework

MRS. S. MAHALAKSHMI[#1] , HARSHITHA BG[*2] , V SHARANYA[*3]
#Assistant Professor, *Student
Dept. of ISE, BMSIT, Bangalore
maha.shanmugam@bmsit.in
harshithabg97@gmail.com sharanyadevi1234@gmail.com

Abstract— **Long Short-Term Memory(LSTM) network are a type of recurrent neural network which are capable of learning order dependence in sequence prediction problems. The two technical problems overcome by LSTMs are vanishing gradients and exploding gradients, both related to how the network is trained and specific internal structure of the units used in the model. A recurrent neural network is a neural network that attempts to model time or sequence dependent behaviour – such as language, stock prices, electricity demand and so on. This is performed by feeding back the output of a neural network layer at time *t* to the input of the same network layer at time *t+1*. Tensorflow and Keras is the framework that supports LSTMs model. Here, we illustrates the implementation of LSTM using both Tensorflow and Keras.**

Keywords- LSTM, Tensorflow, Keras,Neural networks, Softmax.

I. INTRODUCTION

An LSTM network is a recurrent neural network that has LSTM cell blocks in place of our standard neural network layers. These cells have various components called the input gate, the forget gate and the output gate – these will be explained more fully later. LSTMs helps to diminish the vanishing (and exploding) gradient issue, and in this manner permit further systems and recurrent neural systems to perform well in handy settings, there should be an approach to decrease the augmentation of inclinations which are under zero.
The manner in which it does as such is by making an inner memory state which is basically added to the processed input, which enormously decreases the multiplicative impact of small gradient . Two other gates, the input gate and output gate, are also featured in LSTM cells.
To exhibit how LSTM Networks work, we will utilize disentangled procedure. Tensorflow is used for computational reasons. We will push groupings of three symbols as sources of input and one yield. By rehashing this procedure, the system will figure out how to foresee next word dependent on three past ones. Keras is also used to implement the LSTM

II. MODEL DESCRIPTION

A. THE STRUCTURE OF AN LSTM CELL

The structure of a typical LSTM cell is shown in the diagram below:

The data flow is from left-to-right in the diagram above, with the current input Xt and the previous cell output h_{t-1} concatenated together and entering the top “data rail”.

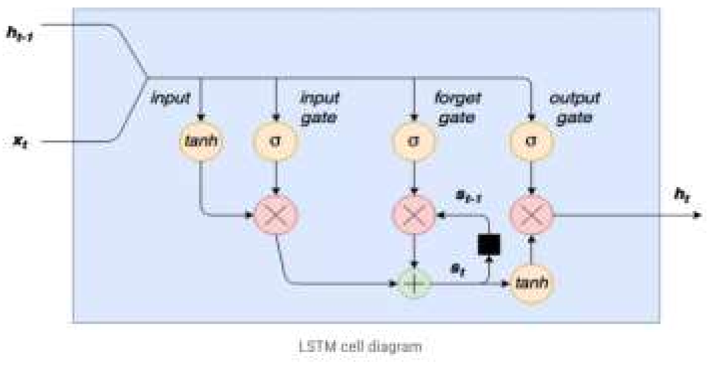


Fig 1: LSTM Cell Diagram

B. COMPONENTS OF LSTM MODEL.

An LSTM module (or cell) has 5 fundamental segments which enables it to model both long-term and short-term data.

1. **Cell state (ct) -** This represents the internal memory of the cell which stores both short term memory and long-term memories.
2. **Input gate (it) -** Chooses how much data from current input streams flows to the cell state. To begin with, the input is squashed between - 1 and 1 utilizing a tanh actuation function. This can be computed by:

   $g = \tanh(Bg+xtUg+ht-1Vg)$

   Where Ug and Vg are the weights for the input and previous cell output, respectively,and Bg is the input bias.

   The expression for the input gate is:

   $i=\sigma(bi+xtUi+ht-1Vi)$

   The output of the input stage of the LSTM cell can be expressed below, where the $\circ$ operator expresses element-wise multiplication:

   $g\circ i$

3. **Forget gate (ft) -** Decides how much information from the current input and the previous cell state flows into the current cell state.
   Two things to notice – first, there is a forget gate here – this forget gate is again a sigmoid activated set of nodes which is element-wise multiplied by $st-1$ to determine which previous states should be remembered (i.e. forget gate output close to 1) and which should be forgotten (i.e. forget gate output close to 0). This allows the LSTM cell to learn appropriate context. The forget gate can facilitate such operations and is expressed as:

   $f = \sigma(Bf+xtUf+ht-1Vf)$

   The output from this stage, st is expressed by:
   $st = st-1\circ f+g\circ i$

4. **Hidden state (ht) -** This is output state information calculated w.r.t. current input, previous hidden state and current cell input which you eventually use to predict the future stock market prices. Additionally, the hidden state can decide to only retrive the short or long-term or both types of memory stored in the cell state to make the next prediction.
5. **Output gate (ot) -** Decides how much information from the current cell state flows into the hidden state, so that if needed LSTM can only pick the long-term memories or short-term memories and long-term memories.The output gate is expressed as:

   $o=\sigma(bo+xtUo+ht-1Vo)$

   So the final output of the cell can be expressed as:

   $ht = \tanh(st)\circ o$

C. LSTM NETWORK ARCHITECTURE

In the below figure, the input shape of the text data is ordered as follows : (batch size, number of time steps, hidden size). In other words, for each batch sample and each word in the number of time steps, there is a 500 length embedding word vector to represent the input word. These embedding vectors will be learnt as part of the overall model learning. The input data is then fed into two “stacked” layers of LSTM cells (of 500 length hidden size) – in the diagram above, the LSTM network is shown as unrolled over all the time steps. The output from these unrolled cells is still (batch size, number of time steps, hidden size).

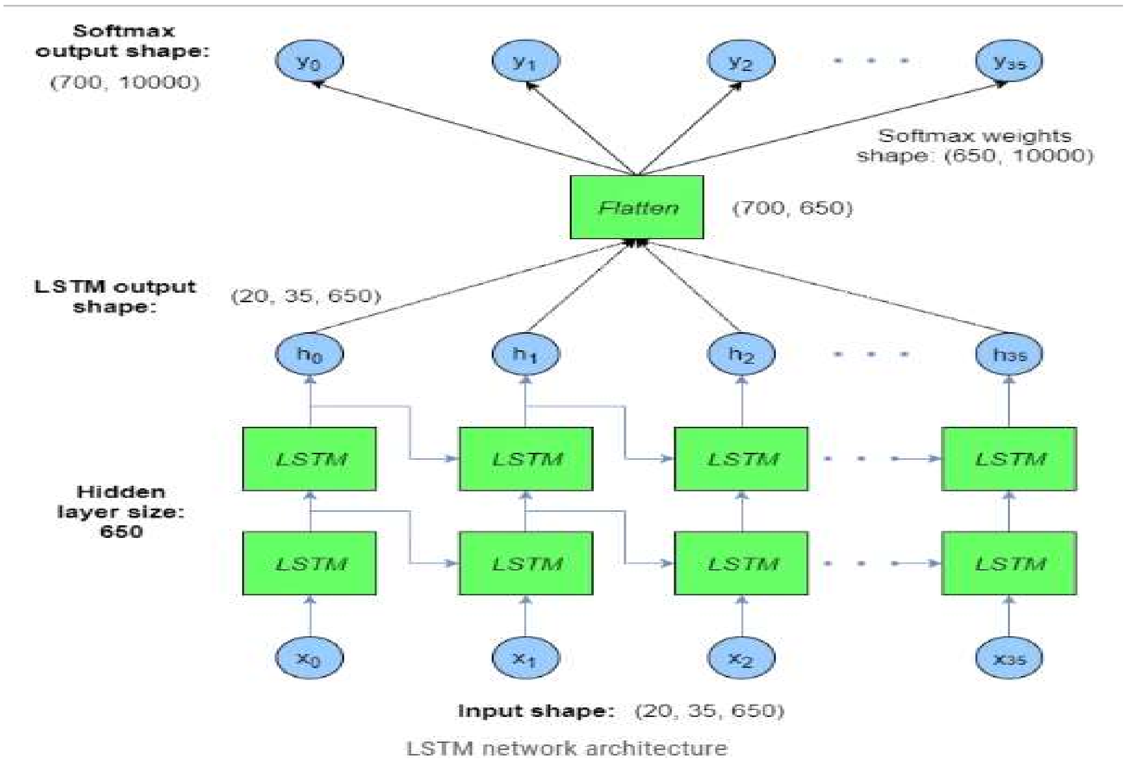


Fig 2: LSTM Network Architecture

This output data is then passed to a Keras layer called TimeDistributed, which will be explained more fully below. Finally, the output layer has a *softmax* activation applied to it. This output is compared to the training *y* data for each batch, and the error and gradient back propagation is performed from there in Keras. The training *y* data in this case is the input *x* words advanced one time step – in other words, at each time step the model is trying to predict the very next word in the sequence. However, it does this at *every* time step – hence the output layer has the same number of time steps as the input layer.

As can be seen in the design above, it is conceivable to stack layers of LSTM cells over one another – this expands the model multifaceted nature and prescient power however to the detriment of preparing times and troubles. The design appeared above is the thing that we will actualize in TensorFlow in the following segment. The small batch size is to allow a more stochastic gradient descent which will avoid settling in local minima during many training iterations.

III RESULTS AND DISCUSSION

Suppose that we need to prepare one LSTM to foresee the following word utilizing (next word in the text) an example content. Straightforward content in our model will be

" *In a sense , people are our proper occupation . Our job is to do them good and put up with them . But when they obstruct our proper tasks , they become irrelevant to us—like sun , wind, animals . Our actions may be impeded by them , but there can be no impeding our intentions or our dispositions . Because we can accommodate and adapt . The mind adapts and converts to its own purposes the obstacle to our acting . The impediment to action advances action . What stands in the way becomes the way ."*

Note that this content is little modified. Each punctuation mark is encompassed by space characters, hence every word and each punctuation mark are considered as an symbol. To exhibit how LSTM Networks work, we will utilize disentangled procedure. We will push groupings of three symbols as sources of input and one yield. By rehashing this procedure, the system will figure out how to foresee next word dependent on three past ones.

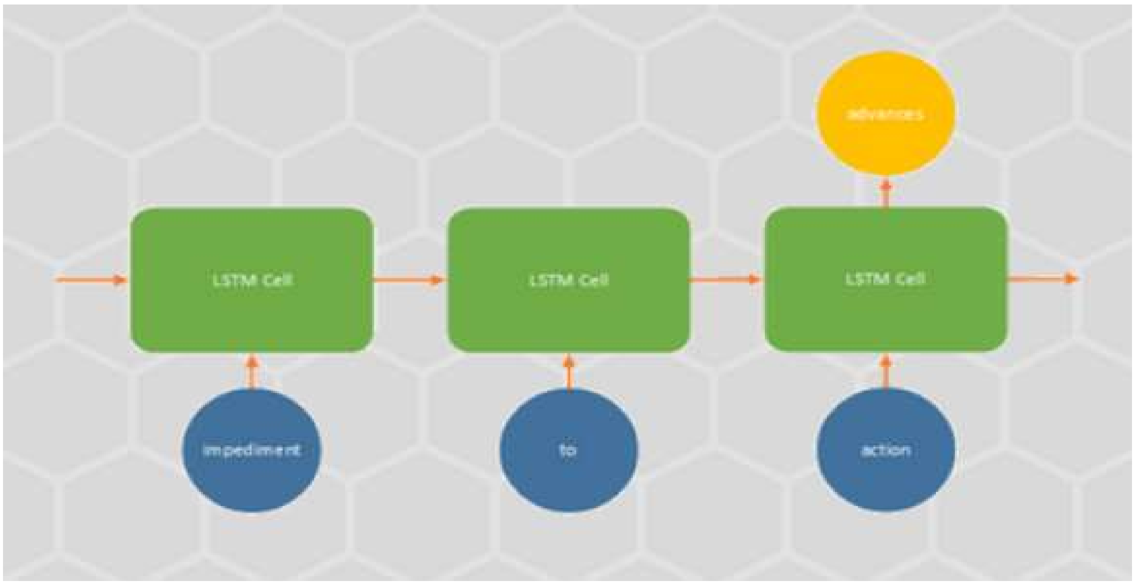
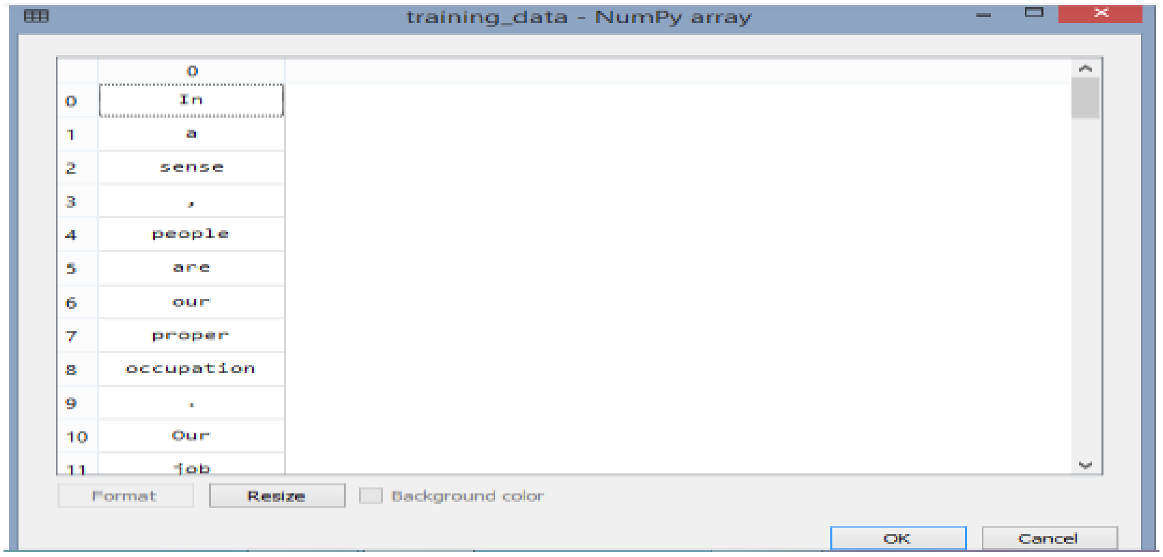


Fig 3: Training LSTM Network

A. TENSORFLOW IMPLEMENTATION

Provide a succession of three images and one yield(output) to the LSTM Network and learn it to foresee that yield. Be that as it may, since we are utilizing numerical models first thing we have to do is to set up this information (content) for any sort of activity. These networks will understand the numbers, not strings as we have it in our content, so we have to change over these symbols into numbers. We will do this by doling out an extraordinary whole number to every symbols put together the with respect to the recurrence of event.For this reason, we will utilize **DataHandler class**. This class has two purposes, to stack the information from the document, and to allocate a number to every symbol.

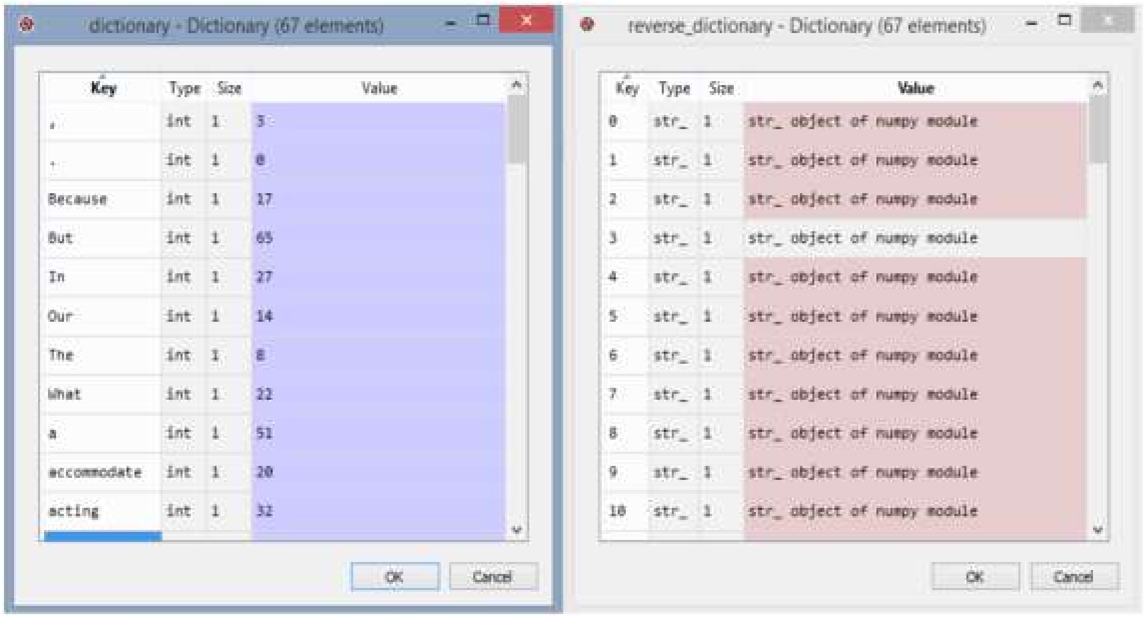
The first main technique for this class is the method read_data which is utilized to read content from the characterized document and create an array of symbols. Here is what that looks like once approached the example content:


training_data - NumPy array

	0
0	In
1	a
2	sense
3	,
4	people
5	are
6	our
7	proper
8	occupation
9	.
10	Our
11	job

Fig 4 : Array of symbols

The second strategy is build_datasets method which is utilized for making two word references (dictionaries). The main lexicon named as just word reference contains symbols as keys and their corresponding number as an value(esteem). Second lexicon reverse_dictionary contains a similar data, just keys are numbers and values are simply the symbols This is how they will look like created using the sample text we are using in this example:


dictionary - Dictionary (67 elements)
reverse_dictionary - Dictionary (67 elements)
OK
Cancel
OK
Cancel

Key	Type	Size	Value
,	int	1	5
.	int	1	0
Because	int	1	17
But	int	1	65
In	int	1	27
Our	int	1	14
The	int	1	8
What	int	1	22
a	int	1	51
accommodate	int	1	20
acting	int	1	32

Key	Type	Size	Value
0	str_	1	str_ object of numpy module
1	str_	1	str_ object of numpy module
2	str_	1	str_ object of numpy module
3	str_	1	str_ object of numpy module
4	str_	1	str_ object of numpy module
5	str_	1	str_ object of numpy module
6	str_	1	str_ object of numpy module
7	str_	1	str_ object of numpy module
8	str_	1	str_ object of numpy module
9	str_	1	str_ object of numpy module
10	str_	1	str_ object of numpy module

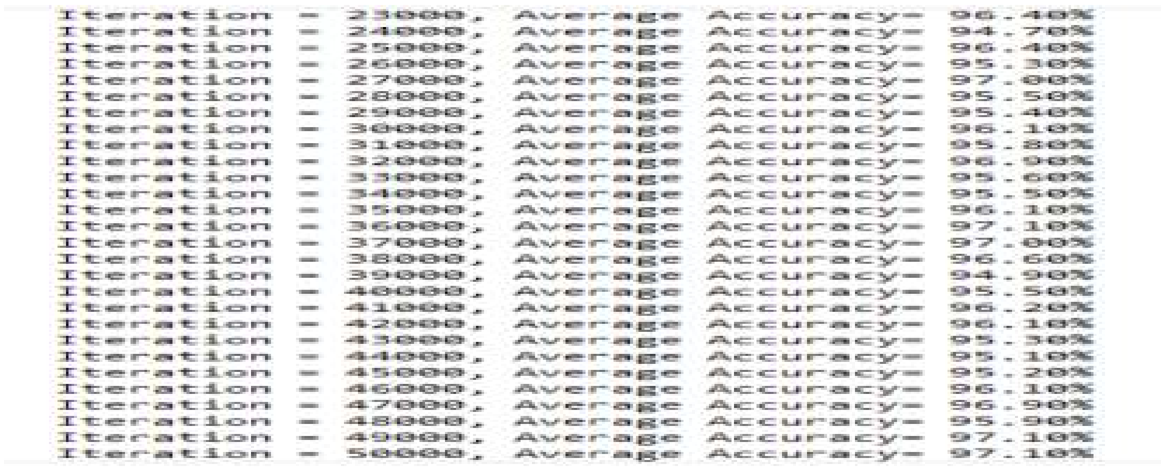
Fig 5: Reverse directory

Once the data is ready for use ,we come to the formation of the model. For this reason, we will make another class that will almost certainly produce LSTM network dependent on the passed parameters. We imported some significant classes there: TensorFlow itself and rnn class from tensorflow.contrib. Since our LSTM Network is a subtype of RNNs we will utilize this to make our model.

Initially, we reshaped our input and after that split it into successions of three symbols. At that point we made the model itself. We made two LSTM layers utilizing BasicLSTMCell technique. Each of these layers has various units characterized by the parameter num_units. Aside from that, we use MultiRNNCell to consolidate these two layers in a single system. At that point we utilized static_rnn strategy to build the system and produce the predictations.

At last, we will utilize SessionRunner class. This class contains condition in which our model will be run and assessed.50000 cycles are being run utilizing our model. We infused model, optimizer, loss function, etc. in the constructor, so the class has this data accessible. Obviously, the primary thing we have to split up the datain the given lexicon, and make encoded yields.Additionally, we are pushing irregular groupings in the model so we abstain from overfitting. This is taken care of by counterbalance(offset) variable. At long last, we will run the session and get accuracy.

Once we run this model we get the accuracy of 97.10%.



```
Iteration - 23000, Average Accuracy= 96.40%
Iteration - 24000, Average Accuracy= 94.70%
Iteration - 25000, Average Accuracy= 96.40%
Iteration - 26000, Average Accuracy= 95.30%
Iteration - 27000, Average Accuracy= 97.00%
Iteration - 28000, Average Accuracy= 95.50%
Iteration - 29000, Average Accuracy= 95.40%
Iteration - 30000, Average Accuracy= 96.10%
Iteration - 31000, Average Accuracy= 95.80%
Iteration - 32000, Average Accuracy= 96.90%
Iteration - 33000, Average Accuracy= 95.60%
Iteration - 34000, Average Accuracy= 95.50%
Iteration - 35000, Average Accuracy= 96.10%
Iteration - 36000, Average Accuracy= 97.10%
Iteration - 37000, Average Accuracy= 97.00%
Iteration - 38000, Average Accuracy= 96.60%
Iteration - 39000, Average Accuracy= 94.90%
Iteration - 40000, Average Accuracy= 95.50%
Iteration - 41000, Average Accuracy= 96.20%
Iteration - 42000, Average Accuracy= 96.10%
Iteration - 43000, Average Accuracy= 95.30%
Iteration - 44000, Average Accuracy= 95.10%
Iteration - 45000, Average Accuracy= 95.20%
Iteration - 46000, Average Accuracy= 96.10%
Iteration - 47000, Average Accuracy= 96.90%
Iteration - 48000, Average Accuracy= 95.90%
Iteration - 49000, Average Accuracy= 97.10%
Iteration - 50000, Average Accuracy= 97.10%
```

Fig 6 : Resulting Accuracy using Tensorflow Platform

B. KERAS IMPLEMENTATION

The energy of Keras is that it abstracts a lot of things we had to take care while we had been the use of TensorFlow. However, it is giving us a much less flexibility. Of course, everything is a trade-off. When training neural networks, we generally feed information into them in small batches, referred to as mini-batches or simply "batches" (for extra information on mini-batch gradient descent, see my tutorial here). Keras has two some accessible features which can extract training facts automatically from a pre-supplied Python iterator/generator object and enter it to the model. One of these Keras features is referred to as fit_generator. The first argument to fit_generator is the Python iterator function that we will create, and it will be used to extract batches of records at some point of the coaching process. This feature in Keras will handle all of the facts extraction, input into the model, executing gradient steps, logging metrics such as accuracy and executing callbacks .

The first step includes developing a Keras mannequin with the Sequential() constructor. The first layer in the network, as per the architecture graph proven previously, is a phrase embedding layer. This will convert our words(referenced by using integers in the data) into meaningful embedding vectors. This Embedding() layer takes the size of the vocabulary as its first argument, then the measurement of the resultant embedding vector that you prefer as the subsequent argument. Finally, because this layer is the first layer in the network, we should specify the "length" of the enter i.e. the number of steps/words in every sample.

In this case, the enter to the embedding layer is (batch_size, num_steps) and the output is (batch_size, num_steps, hidden_size). Note that Keras, in the Sequential model, constantly keeps the batch size as the first dimension. It receives the batch measurement from the Keras becoming function.

The next layer is the first of our two LSTM layers. To specify an LSTM layer, first you have to furnish the variety of nodes in the hidden layers inside the LSTM cell, e.g. the variety of cells in the overlook gate layer, the tanh squashing enter layer and so on. The subsequent argument that is unique in the code above is the return_sequences=True argument. What this does is make certainthat the LSTM mobile returns all of the outputs from the unrolled LSTM mobilethrough time. If this argument is left out, the LSTM mobile will truly grant the output of the LSTM mobilephone from the closing time step.

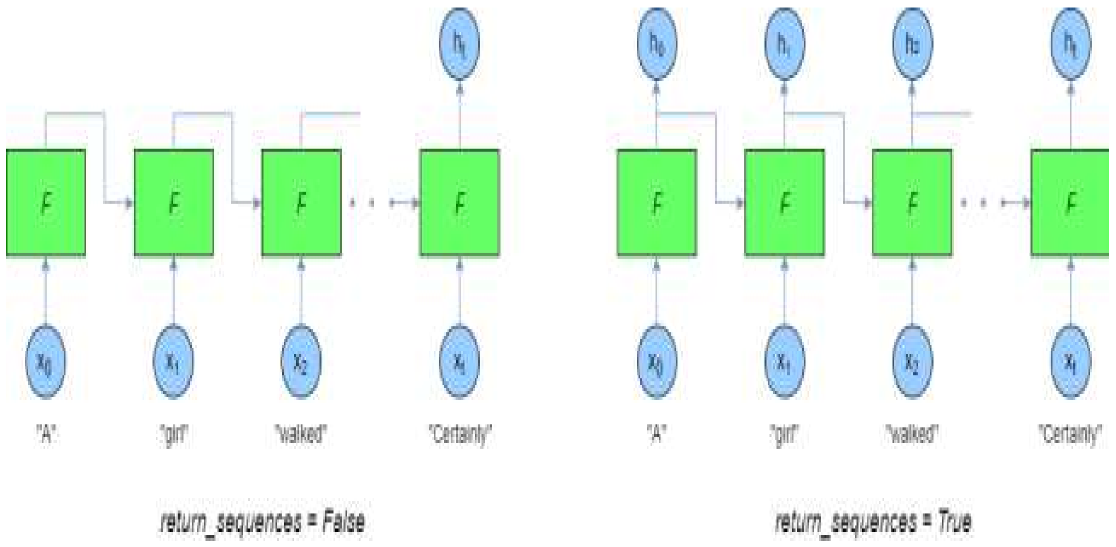
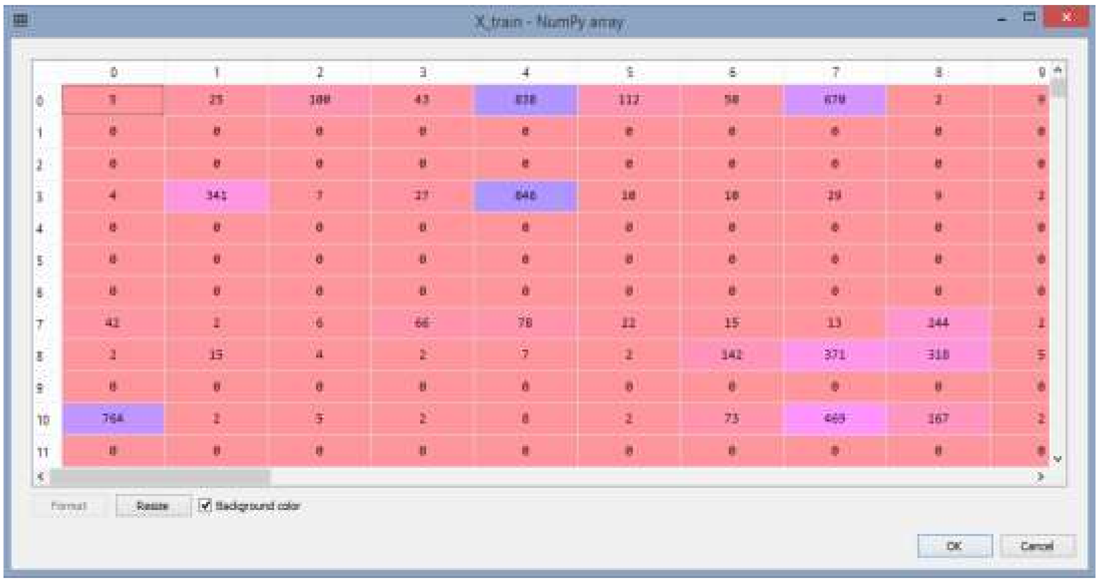


Fig 5: Keras Network model

As can be found in the format above, there is solely one output when return sequences=False – $h t$. However, when return sequences=True all of the unrolled outputs from the LSTM cells are back h0…hth0…ht. In this case, we desire the latter arrangement. Why? Well, in this instance we are attempting to predict the very subsequent word in the sequence. However, if we are attempting to educate the model, it is best to be able to evaluatethe LSTM cell output at every time step with the very next word in the sequence – in this way

we get num steps sources to right mistakes in the model (via back-propagation) instead than simply one for every sample.
Therefore, for both stacked LSTM layers, we prefer to return all the sequences. The output form of every LSTM layer is (batch size, num steps, hidden size).
The subsequent layer in our Keras LSTM network is a dropout layer to forestall overfitting. After that, there is a exclusive Keras layer for use in recurrent neural networks known as Time Distributed. This function adds an impartial layer for each time step in the recurrent model. So, for instance, if we have 10 time steps in a model, a Time Distributed layer working on a Dense layer would produce 10 unbiased Dense layers, one for each time step. The activation for these dense layers is set to be softmax in the closinglayer of our Keras LSTM model.

We import Sequential, Dense and Dropout. Still.. We used Embedding as properly as LSTM from the keras.layers. As you can think about LSTM is used for growing LSTM layers in the networks. Embedding, on the other hand, is used to furnish a dense representation of words.
We are loading dataset of pinnacle one thousand words. After this, we want to divide this dataset and create and pad sequences. This is completed by way of the use of sequence from keras.preprocessing. In the padding we used quantity 200, meaning that our sequences will be 200 phrases long. Here is how training input data is looking like after this:


X_train - NumPy array

	0	1	2	3	4	5	6	7	8	9
0	5	25	100	43	838	112	50	679	2	9
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	4	341	7	27	846	18	18	29	9	2
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	42	2	6	66	78	22	15	13	344	1
8	2	15	4	2	7	2	142	371	318	5
9	0	0	0	0	0	0	0	0	0	0
10	764	2	5	2	0	2	73	469	167	2
11	0	0	0	0	0	0	0	0	0	0

Fig 6: X_train dataset

Now we can define, bring together and match LSTM model. Sequential is used for model composition. The first layer that is added to it is Embedding . After the word embedding is accomplished we brought one LSTM layer. In the end, seeing that this is a classification problem, we are attempting to parent out was the overview precise or bad, Dense layer with sigmoid characteristic is added.
Finally, the model is compiled and binary_crossentorpy and Adam optimizer are used. This is how our model is looking like:

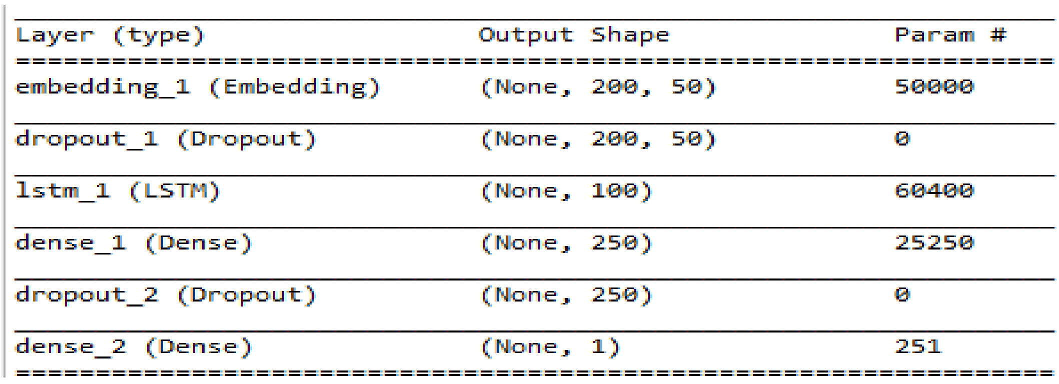
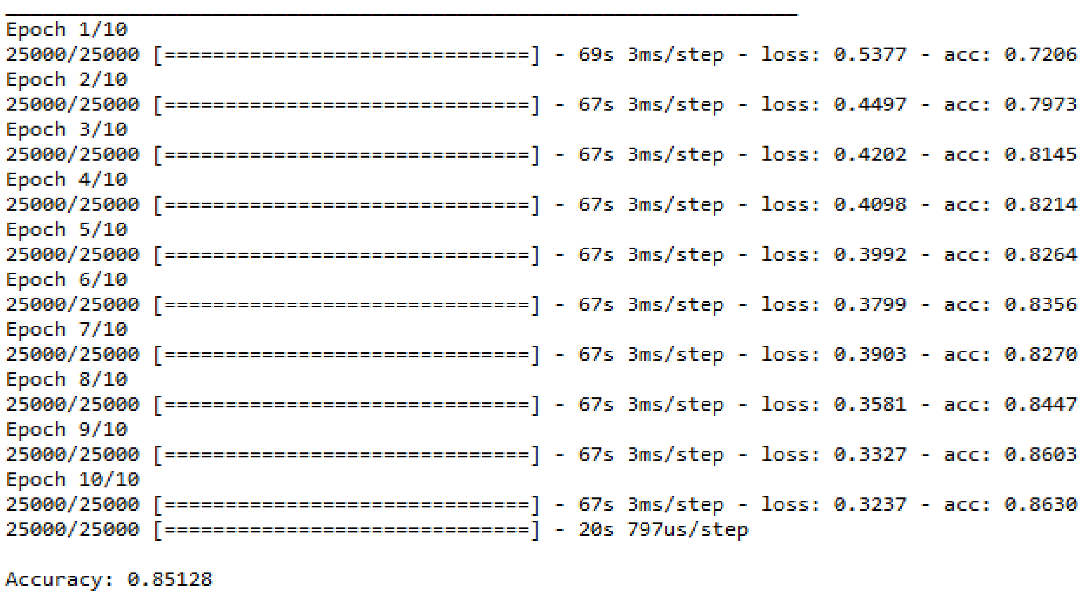

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 200, 50)           50000
_________________________________________________________________
dropout_1 (Dropout)          (None, 200, 50)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               60400
_________________________________________________________________
dense_1 (Dense)              (None, 250)               25250
_________________________________________________________________
dropout_2 (Dropout)          (None, 250)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
=================================================================
```

Fig 7: Model Overview

Once, the data is fit we get an accuracy of 85.12%.



```
Epoch 1/10
25000/25000 [==============================] - 69s 3ms/step - loss: 0.5377 - acc: 0.7206
Epoch 2/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.4497 - acc: 0.7973
Epoch 3/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.4202 - acc: 0.8145
Epoch 4/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.4098 - acc: 0.8214
Epoch 5/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.3992 - acc: 0.8264
Epoch 6/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.3799 - acc: 0.8356
Epoch 7/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.3903 - acc: 0.8270
Epoch 8/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.3581 - acc: 0.8447
Epoch 9/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.3327 - acc: 0.8603
Epoch 10/10
25000/25000 [==============================] - 67s 3ms/step - loss: 0.3237 - acc: 0.8630
25000/25000 [==============================] - 20s 797us/step

Accuracy: 0.85128
```

Fig 8: Resulting accuracy using Keras platform

IV. CONCLUSION

We saw two methodologies while making LSTM systems. The two methodologies were managing basic issues and every wa utilizing an alternate API. Along these lines one could see that TensorFlow is progressively itemized and adaptable, in any case, you need to take care of lot more stuff than when you are using Keras. Keras is easier and increasingly direct however it doesn't give us the adaptability and conceivable outcomes we have when we are utilizing pure TensorFlow.Overall, we obtain an accuracy of 97.10% using Tensorflow and 85.12% using Keras.

REFERENCES

[1] YAZAN and M. F. Talu, "Comparison of the stochastic gradient descent based optimization techniques," 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, 2017, pp. 1-5.doi: 10.1109/IDAP.2017.8090299

[2] R. Gylberth, R. Adnan, S. Yazid and T. Basaruddin, "Differentially private optimization algorithms for deep neural networks," 2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS), Bali, 2017, pp. 387-394.doi: 10.1109/ICACSIS.2017.8355063

[3] Z. Hong, "A preliminary study on artificial neural network," 2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference, Chongqing, 2011, pp. 336-338.doi: 10.1109/ITAIC.2011.6030344

[4] John Persano , Said M. Mikki , And Yahia M. M. Antar ," Gradient Population Optimization: A Tensorflow-Based Heterogeneous Von-Neumann Paradigm for Large-Scale Search", IEEE Access, VOLUME 6, December 2018.

[5] R. Kozma, M. Kitamura, A. Malinowski and J. M. Zurada, "On performance measures of artificial neural networks trained by structural learning algorithms," Proceedings 1995 Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, Dunedin, New Zealand, 1995, pp. 22-25.

[6] N. S. Lari and M. S. Abadeh, "Training artificial neural network by krill-herd algorithm," 2014 IEEE 7th Joint International Information Technology and Artificial Intelligence Conference,2014.

[7] Fatih Etam, Galip Aydm , "Data Classification with Deep Learning using Tensorflow", International Conference on Computer Science and Engineering,2017.

[8] John Persano , Said M. Mikki , And Yahia M. M. Antar ," Gradient Population Optimization: A Tensorflow-Based Heterogeneous Von-Neumann Paradigm for Large-Scale Search", IEEE Access, VOLUME 6, December 2018.