Experiment No. 7
Design and implement LSTM model for handwriting recognition
Date of Performance: 25/09/23
Date of Submission: 09/10/23

Aim: Design and implement LSTM model for handwriting recognition.

Objective: Ability to design a LSTM network to solve the given problem.

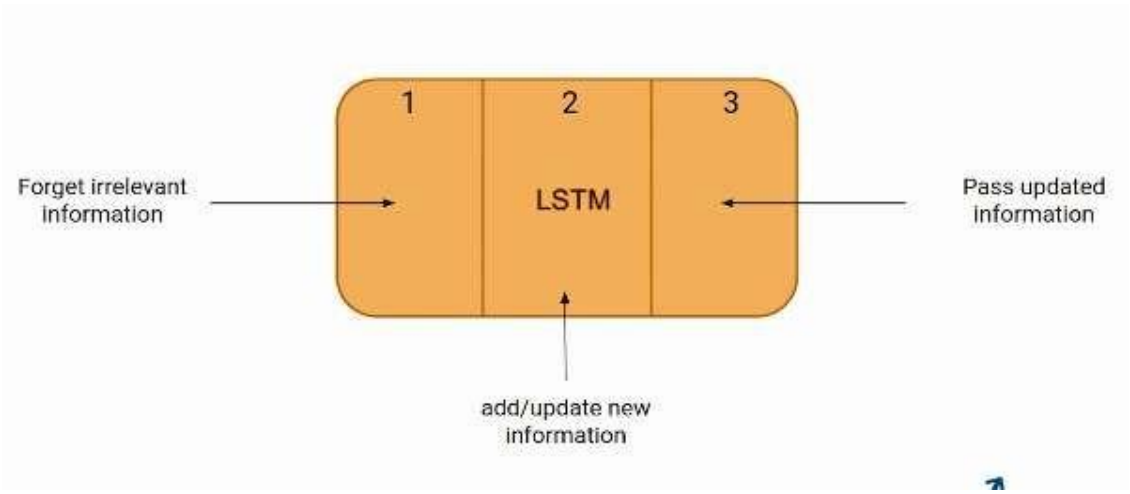
Theory:

LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) architecture widely used in Deep Learning. It excels at capturing long-term dependencies, making it ideal for sequence prediction tasks.

Unlike traditional neural networks, LSTM incorporates feedback connections, allowing it to process entire sequences of data, not just individual data points. This makes it highly effective in understanding and predicting patterns in sequential data like time series, text, and speech.

LSTM Architecture

In the introduction to long short-term memory, we learned that it resolves the vanishing gradient problem faced by RNN, so now, in this section, we will see how it resolves this problem by learning the architecture of the LSTM. At a high level, LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. The LSTM network architecture consists of three parts, as shown in the image below, and each part performs an individual function.



The Logic Behind LSTM

The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp. This one cycle of LSTM is considered a single-time step.

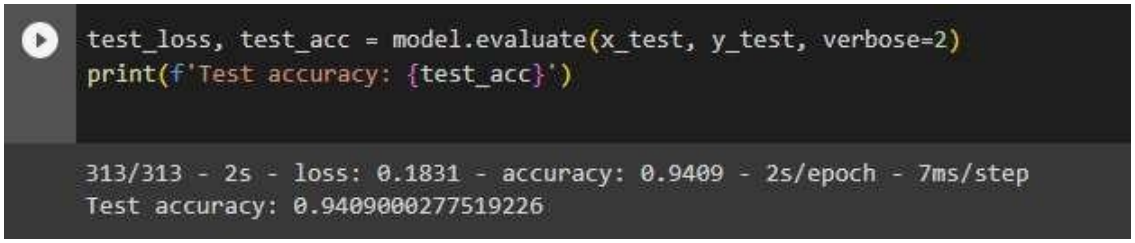
These three parts of an LSTM unit are known as gates. They control the flow of information in and out of the memory cell or lstm cell. The first gate is called Forget gate, the second gate is known as the Input gate, and the last one is the Output gate. An LSTM unit that consists of these three gates and a memory cell or lstm cell can be considered as a layer of neurons in traditional feedforward neural network, with each neuron having a hidden layer and a current state.

Code:

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.layers import LSTM, Dense, Input
```

```
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

input_shape = (28, 28)  # Input image size for MNIST

model = keras.Sequential([

    Input(shape=input_shape),

    LSTM(128),

    Dense(10, activation='softmax')])

model.compile(

    optimizer='adam',

    loss='sparse_categorical_crossentropy',

    metrics=['accuracy']

)

model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, batch_size=64)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print(f'Test accuracy: {test_acc}')
```

Output:



Conclusion:

Designing and implementing an LSTM (Long Short-Term Memory) model for handwriting recognition is a complex task that involves several steps. Here, I'll provide you with a high-level overview of the process and some Python code snippets to help you get started. You'll need a dataset of handwritten characters or words for training and testing your model. The MNIST dataset can be a good starting point, although you may need to preprocess the data to match your specific task.