

**Aim –**

Design RNN or its variant including LSTM or GRU

- A) Select a suitable time series dataset. E.g - Predict sentiments based on product reviews.
- B) Apply for prediction

**Objective –**

Implement a Recurrent Neural Network (RNN) or its variant (LSTM or GRU) on a selected time series dataset, such as predicting sentiments based on product reviews, to develop a predictive model for sentiment analysis.

**Software Requirements –**

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE or Google Colab

**Hardware Requirements –**

A machine with sufficient RAM and processing power for model training(8GB RAM recommended)

**Dataset –**

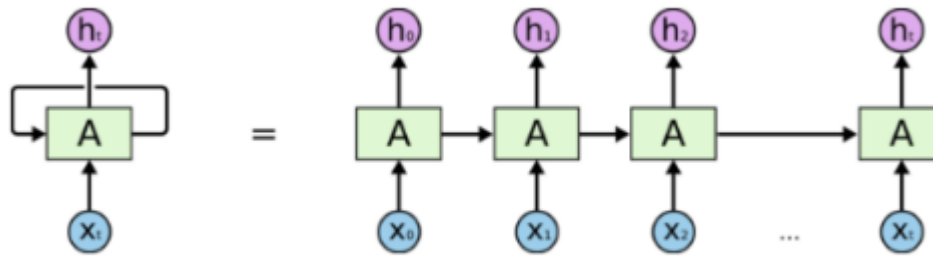
Inbuilt tensorflow-keras-imdb dataset.

**Libraries or Modules Used –**

- Keras
- Tensorflow

**Theory –**

Recurrent Neural Network (RNN) Recurrent Neural Network is a generalization of feed-forward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input. Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as un segmented, connected handwriting recognition or speech recognition. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are



**An unrolled recurrent neural network.**

First, it takes the  $x(0)$  from the sequence of input and then it outputs  $h(0)$  which together with  $x(1)$  is the input for the next step. So, the  $h(0)$  and  $x(1)$  is the input for the next step. Similarly,  $h(1)$  from the next is the input with  $x(2)$  for the next step and so on. This way, it keeps remembering the context while training. The formula for the current state is

$$h_t = f(h_{t-1}, x_t)$$

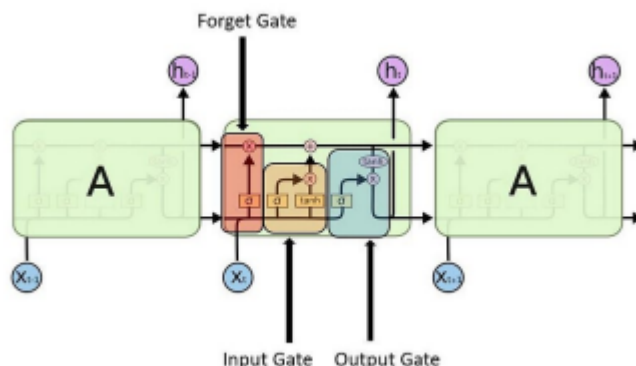
Applying Activation Function:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$W$  is weight,  $h$  is the single hidden vector,  $W_{hh}$  is the weight at previous hidden state,  $W_{xh}$  is the weight at current input state,  $\tanh$  is the Activation function, that implements a Non-linearity that squashes the activations to the range  $[-1, 1]$

$$y_t = W_{hy}h_t$$

$y_t$  is the output state. Why is the weight at the output state. Long Short Term Memory (LSTM) Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:



- 1) Input gate - discover which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from -1 to 1.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- 2) Forget gate - discover what details to be discarded from the block. It is decided by the sigmoid function. it looks at the previous state ( $h_{t-1}$ ) and the content input ( $x_t$ ) and outputs a number between 0 (omit this) and 1 (keep this) for each number in the cell state  $C_{t-1}$ .

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- 3) Output gate — the input and the memory of the block is used to decide the output. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from -1 to 1 and multiplied with output of Sigmoid.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

#### Algorithm –

- 1) Load IMDb Movie Reviews dataset (50,000 reviews)
- 2) Pre-process dataset by removing special characters, numbers, etc. from user reviews + convert sentiment labels positive & negative to numbers 1 & 0, respectively
- 3) Import GloVe Word Embedding to build Embedding Dictionary + Use this to build Embedding Matrix for our Corpus
- 4) Model Training using Deep Learning in Keras for separate: Simple Neural Net, CNN and LSTM Models and analyse model performance and results Perform Predictions
- 5) Perform predictions on real IMDb movie reviews

**Application -** 1) Product Review Sentiment Analysis: Predict sentiment (positive, negative, neutral) from user reviews for product improvement insights.

2) Customer Feedback Analysis: Analyze sentiments in customer feedback to understand overall satisfaction and identify areas for improvement.

3) Brand Monitoring: Monitor social media for product mentions and analyze sentiments to assess brand perception.

4) Market Research: Analyze sentiments in market surveys to gauge consumer opinions about specific products or features.

5) Quality Assurance in E-commerce: Automatically categorize and flag reviews with negative sentiments to improve product quality.

### Code –

```
import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

from tensorflow.keras.preprocessing.sequence import pad_sequences

# Set the parameters

max_features = 10000 # Number of words to consider as features maxlen = 100 # Cut texts after this
number of words (among top max_features most common words) batch_size = 32

# Load the IMDB dataset

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad sequences to have a consistent length for the input to the RNN

x_train = pad_sequences(x_train, maxlen=maxlen)

x_test = pad_sequences(x_test, maxlen=maxlen)

# Build the RNN model with LSTM

model = Sequential() model.add(Embedding(max_features, 128)) model.add(LSTM(64, dropout=0.2,
recurrent_dropout=0.2)) model.add(Dense(1, activation='sigmoid'))

# Compile the model

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

model.fit(x_train, y_train, batch_size=batch_size, epochs=5, validation_data=(x_test, y_test))

# Evaluate the model

score, acc = model.evaluate(x_test, y_test, batch_size=batch_size) print(f'Test score: {score}')
print(f'Test accuracy: {acc}')
```

### Conculsion-