|  |  |
| --- | --- |
| **Ex no : 5** | **Implement Recurrent neural networks to generate new text** |
| **Date :** |

**Aim**

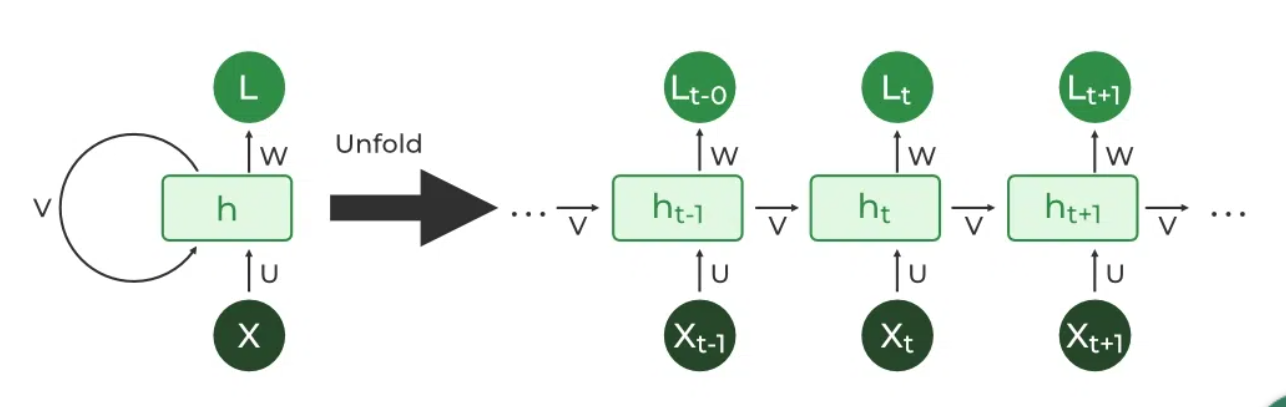
To implement recurrent neural networks to generate new text.

**Basic Theory of recurrent neural networks.**

Recurrent Neural Network (RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.

Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its Hidden state, which remembers some information about a sequence.

The state is also referred to as Memory State since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.



RNNs have the same input and output architecture as any other deep neural architecture. However, differences arise in the way information flows from input to output.

Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains the same. It calculates state hidden state Hi for every input Xi.

The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step. Each unit has an internal state which is called the hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step.

This hidden state is updated at every time step to signify the change in the knowledge of the network about the past.

**Code**

#Import Libraries

import random

import numpy as np

import tensorflow as tf

#Load the file and pre-process it

filepath = tf.keras.utils.get\_file('shakespeare.txt',

'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')

text = open(filepath, 'rb').read().decode(encoding='utf-8')

text = open(filepath, 'rb').read().decode(encoding='utf-8').lower()

text = text[300000:800000]

characters = sorted(set(text))

char\_to\_index = dict((c, i) for i, c in enumerate(characters))

index\_to\_char = dict((i, c) for i, c in enumerate(characters))

#Assigning Length and Size

SEQ\_LENGTH = 40

STEP\_SIZE = 3

sentences = []

next\_char = []

#Creating x,y set as Train and Test

for i in range(0, len(text) - SEQ\_LENGTH, STEP\_SIZE):

sentences.append(text[i: i + SEQ\_LENGTH])

next\_char.append(text[i + SEQ\_LENGTH])

x = np.zeros((len(sentences), SEQ\_LENGTH,

len(characters)), dtype=np.bool)

y = np.zeros((len(sentences),

len(characters)), dtype=np.bool)

for i, satz in enumerate(sentences):

for t, char in enumerate(satz):

x[i, t, char\_to\_index[char]] = 1

y[i, char\_to\_index[next\_char[i]]] = 1

#Model Design and training.

import random

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import RMSprop

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

model = Sequential()

model.add(LSTM(128,

input\_shape=(SEQ\_LENGTH,

len(characters))))

model.add(Dense(len(characters)))

model.add(Activation('softmax'))

model.compile(loss='categorical\_crossentropy',

optimizer=RMSprop(lr=0.01))

model.fit(x, y, batch\_size=256, epochs=4)

#Defining organizing function

def sample(preds, temperature=1.0):

preds = np.asarray(preds).astype('float64')

preds = np.log(preds) / temperature

exp\_preds = np.exp(preds)

preds = exp\_preds / np.sum(exp\_preds)

probas = np.random.multinomial(1, preds, 1)

return np.argmax(probas)

#Final text generation function

def generate\_text(length, temperature):

start\_index = random.randint(0, len(text) - SEQ\_LENGTH - 1)

generated = ''

sentence = text[start\_index: start\_index + SEQ\_LENGTH]

generated += sentence

for i in range(length):

x\_predictions = np.zeros((1, SEQ\_LENGTH, len(characters)))

for t, char in enumerate(sentence):

x\_predictions[0, t, char\_to\_index[char]] = 1

predictions = model.predict(x\_predictions, verbose=0)[0]

next\_index = sample(predictions,

temperature)

next\_character = index\_to\_char[next\_index]

generated += next\_character

sentence = sentence[1:] + next\_character

return generated

#Testing text generation function

print(generate\_text(300, 0.2))

**OUTPUT**

stretched arms,

yet parted but the shado hat the be the ward i har the hard the seat the came the with the be the beat the her will the seald the sere the sear her sear the her the beat the sout the will the the cand the ward in the sould the sear se the sear he her se the hard of the be the sear the sear the will the wall the her sear th

**RESULT**

Thus implementation of recurrent neural networks to generate new text has been carried out successfully.