**Experiment No.: 7**

**Title: Recurrent Neural Network**

**Name: Aarya Tiwari Batch: Honours - 4 Roll No.: 16010421119**

**Experiment No.: 7**

**Aim:** To implement Recurrent Neural Network

**Resources needed: Python/Matlab**

**Theory:** Recurrent neural networks (RNNs) are a class of neural networks that are naturally suited to processing time-series data and other sequential data. Recurrent neural networks are an extension to feedforward networks, in order to allow the processing of variable-length (or even infinite-length) sequences, and some of the most popular recurrent architectures in use, including long short-term memory (LSTM) and gated recurrent units (GRUs). RNN is a neural network designed for analyzing streams of data by means of hidden units. In some of the applications like text processing, speech recognition and DNA sequences, the output depends on the previous computations. Since RNNs deal with sequential data, they are well suited for the health informatics domain where enormous amounts of sequential data are available to process.

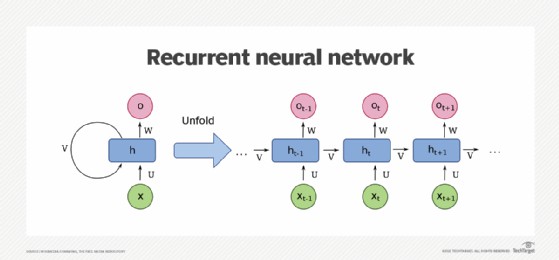


Figure 1: courtesy of Wikimedia Commons -- depicting a one-unit RNN. The bottom is the input state; middle, the hidden state; top, the output state. U, V, W are the weights of the network. Compressed version of the diagram on the left, unfold version on the right.

Looking at the visual below, the “rolled” visual of the RNN represents the whole neural network, or rather the entire predicted phrase, like “feeling under the weather.” The “unrolled” visual represents the individual layers, or time steps, of the neural network. Each layer maps to a single word in that phrase, such as “weather”. Prior inputs, such as “feeling” and “under”, would be represented as a hidden state in the third timestep to predict the output in the sequence, “the”.

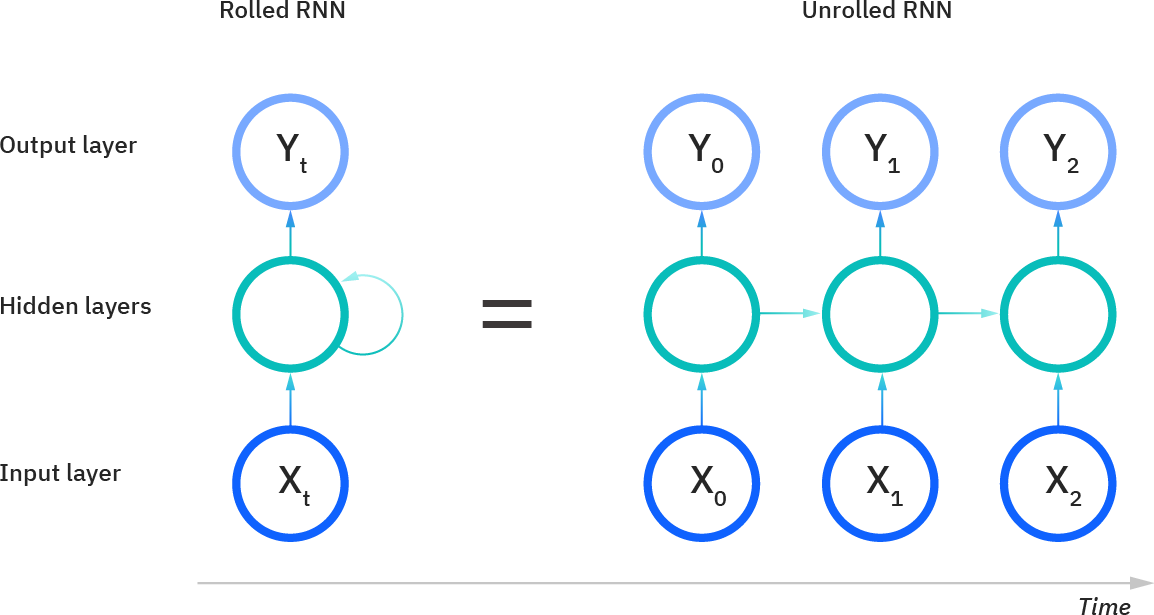


Figure 2: Rolled and Unrolled RNN

Another distinguishing characteristic of recurrent networks is that they share parameters across each layer of the network. While feedforward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network. That said, these weights are still adjusted in the through the processes of backpropagation and gradient descent to facilitate reinforcement learning. Recurrent neural networks leverage backpropagation through time (BPTT) algorithm to determine the gradients, which is slightly different from traditional backpropagation as it is specific to sequence data. The principles of BPTT are the same as traditional backpropagation, where the model trains itself by calculating errors from its output layer to its input layer. These calculations allow us to adjust and fit the parameters of the model appropriately. BPTT differs from the traditional approach in that BPTT sums errors at each time step whereas feedforward networks do not need to sum errors as they do not share parameters across each layer.

**Training through RNN:**

1. A single time step of the input is provided to the network.
2. Then calculate its current state using set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

**Activity:**

* + Import Requisite Libraries
  + Load any time series dataset.
  + Pre-process and visualize the dataset.
  + Form the Training and Testing Data.
  + Develop and train the model.
  + Plot the predictions for training and testing data.

**Program:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

# Load the dataset

data = pd.read\_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv')

# Pre-process and visualize the dataset

data['Month'] = pd.to\_datetime(data['Month'])

plt.plot(data['Month'], data['Passengers'])

plt.xlabel('Month')

plt.ylabel('Number of Passengers')

plt.title('Airline Passengers Over Time')

plt.show()

# Form the Training and Testing Data

X = data['Passengers'].values.reshape(-1, 1)

y = data['Passengers'].values.reshape(-1, 1)

# Normalize the data

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X = scaler.fit\_transform(X)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

# Reshape data for RNN input

X\_train = X\_train.reshape((X\_train.shape[0], 1, X\_train.shape[1]))

X\_test = X\_test.reshape((X\_test.shape[0], 1, X\_test.shape[1]))

# Develop and train the model

model = Sequential([

    SimpleRNN(64, input\_shape=(X\_train.shape[1], X\_train.shape[2])),

    Dense(1)

])

model.compile(optimizer='adam', loss='mean\_squared\_error')

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=8, validation\_data=(X\_test, y\_test), verbose=0)

# Plot the predictions for training and testing data

train\_predictions = model.predict(X\_train)

test\_predictions = model.predict(X\_test)

plt.plot(data['Month'][:-len(test\_predictions)], y\_train, label='Actual (Training)')

plt.plot(data['Month'][-len(test\_predictions):], y\_test, label='Actual (Testing)')

plt.plot(data['Month'][:-len(test\_predictions)], train\_predictions, label='Predicted (Training)')

plt.plot(data['Month'][-len(test\_predictions):], test\_predictions, label='Predicted (Testing)')

plt.xlabel('Month')

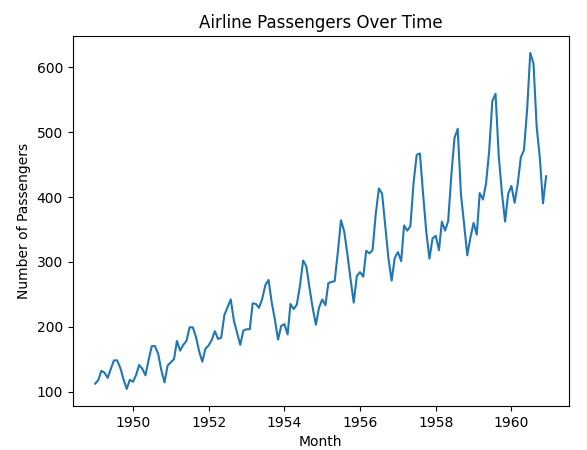
plt.ylabel('Number of Passengers')

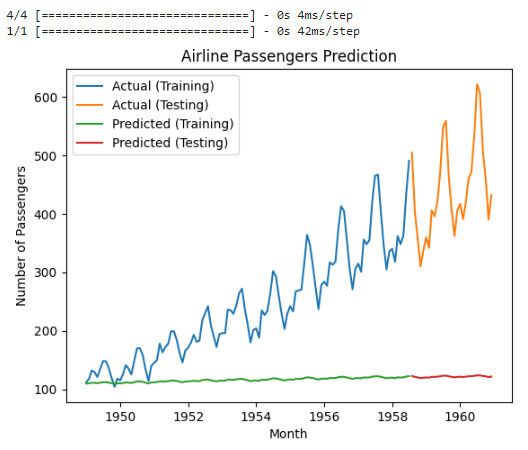
plt.title('Airline Passengers Prediction')

plt.legend()

plt.show()

**Output:**





**Post Lab Question- Answers (If Any):**

1. Differentiate between recurrent Neural Network and Feedforward Neural Network.

Recurrent Neural Networks (RNNs) and Feedforward Neural Networks (FNNs) are both types of artificial neural networks but differ significantly in their architectures and capabilities.

Architecture:

Feedforward Neural Network (FNN):

* In an FNN, the information flows in only one direction, from the input layer through one or more hidden layers to the output layer.
* There are no cycles or loops in the network structure.
* It is often represented as a series of connected layers, and each neuron in one layer is connected to every neuron in the subsequent layer.

Recurrent Neural Network (RNN):

* In an RNN, there are connections that form directed cycles, allowing information to persist.
* The output from one time step is fed back into the network as input to the next time step, creating a form of memory.
* This recurrent connection enables RNNs to handle sequential data such as time series, text, and speech effectively.

Memory:

Feedforward Neural Network (FNN):

* FNNs lack memory as they process each input independently without considering previous inputs or outputs.
* They are suitable for static inputs where the order of input data doesn't matter.

Recurrent Neural Network (RNN):

* RNNs possess memory due to their recurrent connections, allowing them to maintain information over time.
* They can effectively capture temporal dependencies in sequential data by considering the sequence context.

Applications:

Feedforward Neural Network (FNN):

* FNNs are commonly used for tasks like classification, regression, pattern recognition, and function approximation where the input and output are independent of each other.
* They are suitable for static datasets where each input is unrelated to the others.

Recurrent Neural Network (RNN):

* RNNs excel in tasks involving sequential data such as time series prediction, natural language processing (e.g., language modeling, machine translation), speech recognition, and handwriting recognition.
* They are well-suited for tasks where the order of the input data is critical, and context matters.

Training:

Feedforward Neural Network (FNN):

* FNNs are typically trained using gradient descent algorithms such as backpropagation.
* They can suffer from vanishing or exploding gradients, especially in deep networks.

Recurrent Neural Network (RNN):

* Training RNNs involves techniques like backpropagation through time (BPTT), which extends backpropagation to sequential data by unfolding the network over time.
* They can also suffer from vanishing gradients, but techniques like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells have been developed to mitigate this issue.

1. What are the problems associated with RNN.

Vanishing and Exploding Gradient Problem:

* RNNs can suffer from the vanishing or exploding gradient problem during training, especially in long sequences.
* In long sequences, gradients may become very small (vanish) or very large (explode), which makes it difficult for the network to learn long-term dependencies effectively.

Short-term Memory:

* Standard RNN architectures have difficulty in capturing long-term dependencies due to their inherent short-term memory limitations.
* They tend to forget information from earlier time steps as the sequence progresses, which limits their ability to model complex temporal patterns.

Difficulty in Capturing Long-Term Dependencies:

* RNNs struggle to capture dependencies that span a large number of time steps, particularly if these dependencies are far apart.
* This limitation can hinder performance in tasks where understanding long-range dependencies is crucial, such as language translation or speech recognition.

Training Instability:

* Training RNNs can be challenging and unstable, especially when dealing with large datasets and complex architectures.
* They are sensitive to hyperparameters and initialization, and small changes in these parameters can lead to significant differences in performance.

Computationally Intensive:

* RNNs are computationally intensive, particularly during training, due to the sequential nature of their computations.
* This can lead to longer training times, especially for deep RNN architectures or when dealing with large datasets.

Inability to Handle Variable-Length Sequences:

* Standard RNNs require fixed-length input sequences, which can be problematic when dealing with variable-length input data.
* Padding or truncation techniques are often used to address this issue, but they may introduce additional complexity and inefficiency.

Difficulty in Capturing Contextual Information:

* RNNs may struggle to capture contextual information effectively, especially in tasks involving complex linguistic or semantic relationships.
* This limitation can lead to suboptimal performance in natural language processing tasks such as sentiment analysis or question answering.

Overfitting:

* Like other neural network architectures, RNNs are susceptible to overfitting, especially when dealing with small datasets or when the model capacity is too high.
* Regularization techniques such as dropout or weight decay are commonly used to mitigate this problem.

**CO4: Understand the essentials of Recurrent and Recursive Nets.**

**Conclusion:**

We learnt about RNN and implemented it in Google Colab

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of faculty in-charge with date**

**References:**

**Books/ Journals/ Websites:**

1. Josh Patterson and Adam Gibson, “Deep Learning A Practitioner’s Approach”, O’Reilly Media 2017
2. **http**[**s://www**](http://www.ibm.com/cloud/learn/recurrent-neural-networks)**.ibm**[**.com/cloud/learn/recurrent-neural-networks**](http://www.ibm.com/cloud/learn/recurrent-neural-networks)
3. **https://searchenterpriseai.techtarget.com/definition/recurrent-neural-networks**
4. **http**[**s://www**](http://www.sciencedirect.com/science/article/pii/B9780128161760000260)**.sc**[**ience**](http://www.sciencedirect.com/science/article/pii/B9780128161760000260)**d**[**irect.com/science/article/pii/B9780128161760000260**](http://www.sciencedirect.com/science/article/pii/B9780128161760000260)