**ADTA 5560: Recurrent Neural Networks for Sequence Data**

Thuan L Nguyen, PhD

**Final Project by Yog Chaudhary**

**2. PART I: A Time-series Data Set (10 Points)**

Write a report on the dataset that includes (but not limited to) all the critical information of the dataset, e.g., name, data type, an official website where to get the data, links to download, the time-series span (how long the time span is), the data (how many data points), the data structure of the data contained in the dataset, and so on.

**Airline Baggage Complaints - Time Series Dataset**

**Name:** Airline Baggage Complaints - Time Series Dataset

**Data Type:** Time-series

**Official Website:** [**https://www.kaggle.com/datasets/gabrielsantello/airline-baggage-complaints-time-series-dataset?resource=download**](https://www.kaggle.com/datasets/gabrielsantello/airline-baggage-complaints-time-series-dataset?resource=download)

**Download Link:** Airline Baggage Complaints Dataset (4 kB)

**About the Dataset:**

Traveling by air inevitably comes with occasional problems such as flight delays, flight cancellations, and issues about baggage. Below is a dataset describing how complaints related to baggage issues were tracked over time to understand what problems occurred about baggage for three different airlines.

**Time Series Span:**

The data set ranges from the year 2004 to 2010 on a monthly basis for United Airlines, American Eagle, and Hawaiian Airlines.

**Data Points:**

This dataset contains 253 Datapoints, each containing one airline's monthly observation within the period in question. Therefore, each observation within the set contains the following variables:

1. Carrier: Airline operating the flight.

2. Date: Month and year of observation.

3. month: The month of observation.

4. Year: The year of the observation.

5. Per-baggage: The sum of passenger complaints about baggage that encompasses: theft, loss, damage, or misrouting.

6. Scheduled: Number of flights which airlines had scheduled for that month.

7. Cancelled:\_ Total number of flights cancelled by airline for that month.

8. Enplaned: The number of passengers that actually flew on an aircraft operated by them in any given month.

**Data Structure:**

The data are tabular with one record per month per airline. The variables are an airline name, date, number of baggage complaints, number of scheduled flights, number of cancelled flights, and the number of passengers enplaned.

**Conclusion:**

This data represents the trend and pattern of the baggage complaints of three major airlines over six years and hence holds immense value. Analysis of this data will, therefore, be useful to the airline companies in finding out the lacuna in their baggage-handling mechanism for improving the overall satisfaction of customers.

**3. PART II: RNN: Simple RNN with Sine Wave Data (10 Points)**

Build, train, and evaluate a simple recurrent neural network (a complete simple RNN) with two layers: a SimpleRNN and a fully connected layer in Keras. The student must determine how many neurons are used in the SimpleRNN layer.

**RNN: SIMPLE RECURRENT NEURAL NETWORK**

A simple RNN, or a basic recurrent neural network, contains one hidden vector H. Since there is only one hidden state vector represented by it, this one-layer neural network is referred to as Vector H. In this one-layer network, there is feedback. At times, it is called a vanilla neural network, or an Elman RNN-after Professor Jeffrey Elman.

**Recurrent Neural Network with Simple RNN Cell: Architecture:**

It mainly comprises two parts: one is Feedforward Neural Network and Simple RNN, also known as Vanilla RNN. In the feedforward neural network, there is one fully connected layer.

Now, we will build, train and test the Recurrent neural network. The basic library we are going to use is the fact that we are going to make some plotting and data visualization, also deal with vendors and numbers. We have to use Keras to import some models. We have to use a simple RNN layer, the name is DANCE. Dance is only the fully connected layer which is going to be used to build the second network.

A diagram of a network

Description automatically generated

Next, we could work on creating the Time series Generator. We are going to build a sine wave that will wrap the batteries into the model or generate them. Keras would be ideal for the creation, testing, and training.

We will generate sine waves with the help of the Minmax scaler. After generation, the data needs some preprocessing, for which we will normalize the data within the range.

We need to import basic libraries:

import pandas as pd

import NumPy as np

%Matplotlib inline

import matplotlib.pyplot as plt



**How to generate sine waves:**

First, we use a function called linspace in numpy. So, we generate 500 values or data points. We request this function to divide 500 into data points using a range from 0 to 50. We have 500 values for X lying between 0 and 50. The value will be decided by the trigonometric function.

So, it is an array, and so is y. It constructs a graph using a sine value from -1 to 1. Now, to fill up the dataframe with all data, df, we will have inside it x and y. The index is X here, and the real data is Y. Now, the aim would be to print five values: index in the first column and actual data or a variable in the second column. This index is not considered as a real attribute.

**Spilt data -Train/test:**

20% will be reserved for the project. There are 500 rows in the dataset. This will split the data frame into two data frames.

The range of the starting index is from 0 to 50. For the data frame, we will provide 500 data points for testing and one for training; this is for data cleaning. Then there is a class scaler with n=minmax. These are used for data frames in order for them to fit into an already trained scale, and we have to train this scale. Let's split a range from 0 to 50 into 500 data points to create a time series generator. Number of time series that are used to predict the future divided by length of the input sequence We are going to use 50 previous points to predict future points.

A screenshot of a computer

Description automatically generated

The next one from these three historical data, we predict from the time series input sequence, the time series generator and length=3 for the sequence. To specify the batch size, that is to say, when training or testing, we need to fit the sample into the modal. A batch means several samples. We'll predict the next one using 50 historic data points. For this project, we consider only one value as the batch size = 1. To create this instance, we need to consider batch size, input datasets, and output datasets. The normalized train dataset will be for input dataset and the output datasets will have a length value of 50.

A screenshot of a computer

Description automatically generated

**Build and train:**

So, first of all, we have to create the Recurrent modal. After mentioning the features, there is only one real attribute in this data frame; therefore, we have only one feature. Besides using sequential API, we're going to use keras because we want to use a variety of APIs. At the very beginning we called it as modal, then create a modal in keras, it is empty so far; therefore, we need to add the layer. The second subnetwork is feed-forward of our single layer Vanilla RNN. Here, we will add a simple onion layer to the modal. Here we are using 100 neurons in order to build a Basic RNN. In this project, we are confident about using 50 history points plus the input sequence. Here, since it's created, we have to shape the layer. The input shape is 2D, and the first dimensionality represents the length of the input sequence, depending on the second dimensionality. Coming next, there is a fully connected layer comprising one neuron. Optimization should be imposed at the time of concatenating the modals. Is specification the main priority for any modal process.

We divide the cost into 500 data points ranging from 0 to 50. The values are continuous data points; hence they are floating-point values. MSE is the best cost function for continuous data. Algorithms are another alternative. Close to 10,000 parameters are generated for these layers. One is used in the second layer.

A screenshot of a computer

Description automatically generated

**Train and Fit:**

For a modal, there is a need for training in a method known as a fit generator. In Keras, the fit method is provided. In a modal, input in batches is to be fitted or injected. We shall make use of a time series generator. We will make use of parameters like train, tsgenerator, and epoch. If we train a modal, the training dataset is divided into small patches.

The next processes are sample setups and then modal trainings in multiple iterations. We will inject one batch of samples into the modality. Injecting means we finish one epoch of tuning data.

A screenshot of a computer

Description automatically generated

The first type is where we have completed the training of the modal that belongs to the training dataset. In this project, we are using 5 epochs to train, meaning the system was trained with a modal 5 type or patches made from the initial dataset. Doing 50 iterations while running 5 epochs is how we finish the entire training. The loss we have is 0.0973 in 1 period. This work defines five clear-cut epochs ranging from zero to four and the ending of the last epoch abruptly. Each input sample that each individual gets contains 50 historic data, and during the evaluation against the test data, the input service time for the model is set to 50. A collection of Python arrays will be used to get the last 50 data, and the indexing applied here is negative. The last point, for counting the order, we have gone back 50 times in the count; it is a 2D array, one column and fifty rows, one attribute as well. The modal contains an RNN layer in it. Which is best, three dimensions, not two.

A screen shot of a graph

Description automatically generated

A screen shot of a graph

Description automatically generated

**Evaluate Test the model on test data:**

The sample size is only one. The sample is composed of 50 data points. and those are one feature's values. They are three dimensions. So, in order to predict the value, first we need to define an empty list. Then we will get the first batch of input sequence. The first batch needs to be the last 50 points. Each batch only includes one sample. The length of test data is 100, 50 data points. Wherein we predict on the current batch using the last 50 previously seen data points for every iteration, and this returns the first element of the array, which has the index 0. We create a new data frame taking a list of predicted values and name it prediction. First column is sine value and second is the forecast.

In this graph, the actual value is depicted by a blue line; the expected value is in yellow.

**4. PART III: RNN: LSTM Neural Network (20 Points)**

**Question 3.1:**

--) Explain the vanishing gradient problem

--) Explain the exploding gradient problem

In the mid-1990s, German academia proposed another solution developed by Sepp Hochreiter and Juergen Schmidhuber: the Long ShortTerm Memory neural network. Fault does not affect an LSTM network this way and thus propagates more consistently through the layers and over time. This neural network can keep learning over many time steps.

Diagram

Description automatically generated

Vanishing Gradient Problem: It refers to how the gradients of a network's output about its parameters approach zero during backpropagation, flowing backwards into a network, especially through many layers. In the case of RNNs, this is generally due to the repeated multiplication of gradients through each timestep in backpropagation. Hence, the gradients can be reduced to tiny sizes, actually to the exponential decrease of gradients. This, in turn, makes it nearly impossible for the network to learn long-range dependencies, since the weights associated with the earlier inputs are hardly updated, thus delaying the process of training.

Exploding Gradient Problem: This problem arises when large gradients are accumulated, and huge updates take place on the weights of a neural network while training the network. This could also result in an unstable network where, for small changes in input, huge changes in output might result. This instability is often manifested in the form of oscillations in performance during training, where the accuracy of the model may be very different across training epochs.

**Question 3.2:**

--) Discuss the limitations of the SimpleRNN neural network

This is particularly true for SimpleRNNs, owing to their structure chaining together the same weights and activation function, which are most susceptible to the problem of vanishing and exploding gradients, particularly with long sequences of input.

**Difficulty with Long-range Dependencies:**

Owing to the problem with vanishing gradients, SimpleRNNs cannot capture long-range dependencies within the input data. It gets tough for a SimpleRNN to remember the earlier inputs as the length of the input sequence grows, and, therefore, for the network to do well on tasks requiring an understanding of the whole sequence.

**Question 3.3:**

--) Explain how the LSTM neural network can provide powerful solutions to both gradient problems (Vanishing and Exploding) and address the limitations of the SimpleRNN neural network.

**Architecture of LSTM:**

Such problems are overcome with LSTM networks, which come out naturally by the simplest architecture of SimpleRNNs. The LSTM unit consists of three different gates: input, forget, and output gates controlling the flow of information. Each gate of an LSTM cell votes on whether to let the information pass through or discard the same and, therefore, lets the network maintain longer memories, and hence avoids the vanishing gradient problem.

Input Gate: This decides the value of the input for updating the memory state.

Forget Gate permits the cell either to forget or carry forward the existing memory.

Output Gate: This decides the final value for the next hidden state.

**Solution to Gradient Problems:**

Vanishing Gradients: The structure of the LSTM model enables it to regulate the flow of gradients during the process of backpropagation using its gates. Its forget gate can keep the state over time without having gradients that are exponentially small, something so important while learning long-range dependencies.

Exploding Gradient: LSTMs handle exploding gradients by mainly using gradient clipping, which scales down the magnitude of the gradient if it surpasses a threshold, preventing too extreme weight updating that may cause unstable training behavior.

**Basic knowledge of the cell state and gates in the LSTM architecture.**

* Cell state and gates are two main ideas that this neural network uses in the LSTM architecture.
* It can also make up for certain inefficiencies in Simple RNN due to the problem caused by the vanishing gradient of the short-term memory.

Diagram

Description automatically generated

* In addition to H, or Hidden state, an LSTM cell also keeps track of a C, or Cell state.
* Because the Cell state has no weight that can be carried on for many layers or time steps, then every other subsequent time step or layer may make use of the information brought in from the previous Cell state with no decline if it's still fantastic-that is, relevant and meaningful.

**It has the appearance of a channel of relaying information are flows.**

* Capable of capturing information from the earliest to the last time step along a chain of events.
* Information-theoretically, it could carry useful information during the actual processing of the sequence.
* May be able to help with short-term memory issues.
* To be part of the neural network "memory" in concert with the hidden (H) state.

Graphical user interface

Description automatically generated with medium confidence

**A CHANNEL'S FLOW:**

The information is allowed to flow from beginning to the end without any blockage in the channel.

**A SNAPSHOT OF A CELL'S STATUS EXERCISES**

The Cell State C can "remember" information in a cell at every time step, independent of how early in the sequence it is. The whole plot and record of weights and biases-all or part of their local minimums. In the shortest period of time, we will have enough sets of neural networks, each one more accurate than others. We may want to ensemble these varied networks, aggregate them, or average them before drawing some more general conclusion, we can go further down that graph, continue to take snap shots or pictures of the weights and biases at each of these local minimums. And by the end of this walk, what we have is an ensemble of neural networks. The cost function is a measure that is defined to represent the performance concerning the training data and the projected results; it is a measure showing the accuracy of your model. It has a huge array of characteristics that consist of weights and biases our network learns, and the fewer mistakes the model at any instant of time does-the least the better, it is. Thus, in the presence of more than one minimum, the network could use various weights and biases to minimize mistakes. Any minimum can then be considered as fragile but a possible solution of a more significant problem.

This flow of information in an LSTM cell is shown in four steps:

**Step 1: Forget about the gate.**

Consider whether information coming from the previous Cell state Ct-1 has to be kept-that it is relevant or important-or discarded, since it's not needed any more or unimportant.

f(t) = σ(Wf \* [ ht-1, xt] + bf)

**Step 2:** Gat can fill in. Determine what the informative inputs, ht-1 and xt, to add to the new current Cell state C.(t).

Wi \*[ht-1, xt] + bi = i(t)

tanh(Wc \* [ht-1, xt] + bC) C(temp) = tanh(Wc \* [ht-1, xt] + bC)

Information flows into and out of one LSTM cell in four steps.

**Step 3: Output gate activation.**

Calculate the new current cell state, C(t) at time step t.

C(t) = ft \* Ct - 1 + it \* C(temp)

**Step 4: Output Gate**

To calculate the new current hidden state at time step t, find out which of the inputs - ht-1 and xt - and the new current Cell state C(t) is going to be used.

o(t) = σ (Wo \* [ ht-1, xt ] + bo ) h(t) = o(t) \* tanh (Ct)

A diagram of a flowchart

Description automatically generated

f(t) = σ(Wf \* [ht-1, xt] + bf)

i(t) = σ(Wi \* [ ht-1, xt] + bi)

C(temp) = tanh (Wc \* [ ht-1, xt] + bC)

C(t)=ft \* Ct - 1 + it \* Ctemp

o(t) = σ (Wo \* [ ht-1, xt] + bo),

h(t) = o(t) \* tanh Ct

**Powerful solution to the vanishing gradient problem**

* Let y be the desired output the network is supposed to yield.
* Also, the neural network predicts the output y′.
* That gives the difference: the expected output minus the desired result, and provides the cost or loss represented by the cost/loss function.
* That means that the loss is huge: The network makes lots of mistakes, but the damage is small: Few errors will occur.

**The training/learning process's purpose is to:**

* Minimum weight and bias W and b to be determined which give a minimum loss function E over the complete training set. Hence, the neural network is trained by the back-propagation algorithms.

Backpropagation: In Feedforward Neural Network (FFNN)

Given a very simple feedforward neural network (FFNN) that has only three nodes:

* Input node: I1
* Hidden node: (H1 – HA1) representing a hidden layer
* Output node: (O1 – OA1) representing the output layer

A diagram of a network

Description automatically generated

A gradient is a measure of the rate at which a function's output changes for a small perturbation in its inputs. -Lex Fridman (MIT) A gradient in a deep learning or machine learning model is a storage that keeps track of information about how all weights are changing concerning cost or error.

A diagram of a simple forward neural network

Description automatically generated

**Hare expalanation daigram FFNN:**

* Af: Activity function of a layer
* I1 : Input 1
* H1 = W1 \* I1 is the value of Node 1 before going through Af
* HA1: result from Node 1: Value of Node 1 after the Af. has processed it.
* O1= W2 \* HA1: Value of the Output node before processed by the Af
* OA1 : Output

**Gradient Computation (in Simple RNN):**

A gradient is a measure of how much the output of a function changes if its inputs are infinitesimally perturbed. -Lex Fridman, MIT. In the context of a deep learning or machine learning framework, a gradient observes the change of all weights in relation to alterations of cost or error.

A diagram of a diagram of a system

Description automatically generated with medium confidence

**RNN: LSTM: Core Concepts: Gates**

Gates inside the LSTM cell

* This pathway involves gates that add and remove information from the flow or state of the cell, hence carrying very important information downstream in the sequence.
* In this direction, the LSTM cell takes the gate to modulate how much information travels along the processing path of the sequence chain.
* An LSTM cell includes three kinds of gates: forget gate, input gate, and output gate, each having different sigmoid activations.

Each of these gates can be viewed as a neural network which may be trained to decide whether an input given is important or not before using the information. Another easier way to explain each gate is by thinking of it like a layer in a neural network. This should be based on:

* In the process of training itself, each gate learns to decide.
* Information to be retained which is valuable.
* Some details don't bear remembering, as they are either unimportant or inconsequential.

A graph of a function

Description automatically generated

Activations are also sigmoid gates.

* Sigmoid activation ranges from 0 to 1.
* Sigmoid function to remember or forget the information of the model.
* Zero times any number is 0.
* Dwindling or "becoming forgotten.

One times any number is always that number. Status remains the same, or it is "preserved." A neural network can use the sigmoid activation function so as to:

* decide what information is important and save that information.
* can identify unnecessary data and "forget" it.

A graph of a function

Description automatically generated

**Tanh Activation Function in LSTM Cells:**

* And, the LSTM cells also include tanh activation functions.
* The values associated with Tanh activation functions span from -1 to 1.

**Forget Gate f in LSTM : Gates (t)**

* Whether to Keep or Discard: Irrelevant or unimportant information from the previous Cell state Ct-1.
* This will result in the Remember vector f(t), which updates the previous Cell state Ct-1. Values of this vector lie in the range between 0 and 1.

**For these value generations, which algorithm does the forget gate use?**

* This gate verifies the relevance of information coming from the previous Cell state to identify which data the model needs to retain and discard or ignore by analyzing the input values provided (ht-1 and xt).
* During the training procedure, the gate should master this skill.

A diagram of a cell

Description automatically generated

Inputs are ht-1, the previous hidden state, and current inputs xt.

Outputs: f(t) = σ(Wf \* [ ht-1, xt ] + bf) "Remember" vector.

A numeric vector. Between 0 and 1.

1: "Fully applicable or relevant": stay 100%.

0: Completely irrelevant or inconsequential - remove 100%.

**LSTM: Gates: i(t)\* C -Input Gate temp**

Analyze the inputs ht-1 and Xt: this means deciding which information is relevant to add into the new current Cell state Ct. Run the input values h t-1 and xt through a tanh layer processing values between -1 and 1. The output will be a representation-a vector of numbers-of the key information extracted from the inputs, called the Save vector. The output is going to be added to the current Cell state C(t).

A diagram of a cell

Description automatically generated

First, the inputs go through the sigmoid function, which decides what needs to be kept. This concerns h\_{t-1} and x\_{t}.

**Result: I (t)**

* Those get fed into the tanh function which takes in ht-1 and xt, and returns values between -1 and 1.
* C as an output (temp)
* The element-wise multiplication of the two intermediate outputs at the input gate i(t) and C(temp )
* gives the final output or Result: It's the useful information from the inputs that will go into the new Cell state C(t) at time t.

A diagram of a cell

Description automatically generated

i(t) = Wi \* [ht-1, xt] + bi and previous hidden state ht-1 and current hidden state xt, Middle Output 1.

A list of numbers over the range 0 to 1.

1: "Completely relevant or of interest": retain 100%

0: "Completely irrelevant or unimportant": remove completely.

* Intermediate Output 2: tanh(Wc \* [ht-1, xt] + bC) C(temp) = tanh(Wc \* [ht-1, xt] + bC),
* A vector whose values are between -1 and +1.
* i(t) \* C(temp) i(t) \* C(temp) i(t) \* C(temp) i(t) \* C(temp)
* The generation of the outputs, which would represent the new Cell state C: here, t denotes valuable information.

**LSTM-C gates and Output gates t:**

* Calculate and return the new current Cell state C(t) at step t.
* The inputs at ht-1, xt, and the new current cell state Ct, which can decide what to use to build the new current hidden state ht at the current time step t.
* The focus vector, or output, should be a vector of values describing the new current h(.) at time step t.
* when computing the return of the new current Cell state \.
* f(t) \* Ct-1 = Forget gate output: f(t) \* Ct-1 = Input gate output : t \* C (temp)
* adaptive\_mem\_C(t) = ft \* Ct-1 + it \* C, Output(temp).
* The new current Cell state at time step t, C(t).
* Input at any time step: previous hidden state, ht-1; current hidden state, xt.
* Intermediate Output 1: o(t) = Wo \* [ht-1, xt] + bo o(t) = Wo \* [ht-1, xt] + bo o(t) = Wo \* [ht-1, xt] + bo
* A value between 0 and 1.
* 1: "Completely relevant or significant": retain 100%
* 0: "Completely irrelevant or unimportant", completely remove.
* Ct = the second intermediate output, tanh
* A vector with values between -1 and 1.
* The new hidden state h(t) = o(t) \* tanh( Ct) Focus vector is the result t).

A diagram of a cell

Description automatically generated

Three major gates—forget, input, and output—that correspond to three sigmoid activation functions are thought to be present in the LTSM cell.

In Summary, each of these gates in the LSTM neural network may be said to be a neural network in themselves, since each one of these receives some form of input, makes decisions based on learned parameters, and thereby influence the flow in the network. However, their parameters learn with training that would work out an optimum for network performance.

**5. PART IV: RNN: LSTM with Time-Series Data (20 Points)**

We have hare below Muli-Layer neural Networs below daigaram

A diagram of a network

Description automatically generated

There are many various series. First of all, we have to import some basic libraries. For visualization, we use Matplotlib. Now, we will use a specific dropout time series RNN. It is fully connected. In MLP, there are a number of layers. To create the time series input that we have to import from Keras, we will use a time series generator. And for scaling of data, we also need a Min Max scaler. And for a simple RNN, we really do want some data, which should be sequenced. It is part of a time series. In here, I will use the stock data. It contains a great amount of information.

A screenshot of a computer

Description automatically generated

This is a graphical Pandas labaries representation of the forms. Overall, 7 features, 1980 forms. All the attributes are variable. It is mentioned that the volume is an integer. Min, standard deviation, and max has been included in the statistical summary of the numeric properties.

A screenshot of a computer

Description automatically generated

We'll take the dataset of only one column so I am doing below steps.

A screenshot of a computer

Description automatically generated

A graph with lines and numbers

Description automatically generated

We will be dealing with a time series data comprising length and history data points. An input is a 60-point time series. It is recommended to split data into some for training and some for testing. It's 1981 - length of the dataframe; adding that to the total length of the sequence forms this number. So, in total, there are 1980 data points. Now, we find out what index within the total of the 1980 data points makes up the whole set. The index to divide the dataset has to be less than the length of the testing data. Thus, the final index for the datapoint was 1979.

A screenshot of a computer

Description automatically generated

In the Test data, 50 historic data are taken to give a prediction for the same. Details of record 5 data are to represent and last 5 records of the train that should be printed. Initial index is zero and Final index is 1979.

A screenshot of a computer

Description automatically generated

Now, we have to normalize the data within a range from 0 to 1. We can do better. We shall make use of a minmax scalar class and tool. Hence, if we find the fit with the data training, we set up and normalize the data. To train the modal, we will need to develop a time series generator. Input is usually fed into the modal in batch, though of course, batch is an option. In this context, batch means sample correction. At this time, the sample will be a time-series input sequence, so in fact, a batch is made up of samples with 32, being a time-series input sequences.

In such a case, using keras is not very useful to generate batches of this sort manually. The data points are indexed with a 60-point index. The values range from 1 to 60. Based on these, a subsequent index 61 can be predicted.

A screenshot of a computer

Description automatically generated

The next step in the phases phases of the project, we need to specify the number of features. That "feature" means the number of attributes or variables in the dataset; thus, the closing feature is 1. Before the use of a simple RNN layer provided by Keras, we will use the LSTM layer in the design of the neural network. I'm going to apply a new layer called Dropout in this LSTM neural network. We LSTM-fit time series. This one series generates the output; output feeds into the next layer. We will feed the feature output to another layer. That is, for instance, assuming we look at 1100 and may show the % drop, which is normally with 10-15 ranges. For a neural network, we're going to create keras sequentially. If the data has to stream across another layer, we need to setup the return sequence. Then there's the shape for a 3-dimensional input; really, it's all about the 2D, just with the additional batch size that needs to be fed in.

We are going to apply the same fit function as usually. We can see 100 visible epoch ids. For every stage 37 batches need to fit into the input sequence. We practice 37 steps. During initial runs, we go for 37 batches of the input sequence, and the loss is 0.0027 - where the loss shrinks in size.

A screen shot of a graph

Description automatically generated

The modal needs to be viewed. In the training phase, the modal does the prediction. We come to know the past about modal. We frame the most recent value in a data frame. We shall see how the mode does the training process when it is trained. We do need to validate values. We shall also require the test data for testing the modal. Another adjustment was made in the test data in batches with a batch size of 1. In other words, there is only one input sequence in each batch. Input time series to the generator for testing the modal. The value of four parameters will be fed in. The line of input sequence has 60 data points. Next, we will train the data and shift the modal to forecast into the time series. Since this is not a normalized value, it becomes necessary to transfer the predicted value. Finally, the predicted value is converted to a normalized value and is applied to the test datasets of 1980.

In this, 10% will be utilized for testing. Increasing the size of time-series data for training and testing increases correctness in the technique called Long-Short-Term Memory, abbreviated as LSTM. This is considered a required strategy to be able to show more precise results. This cell will include four layers which will work together in harmony in producing the output of both the cell state and the said output. These two get further fed to the next hidden layer. Unlike RNN, which contains only one layer of tanh, LSTMs contain three logistic sigmoid gates and a layer. In the case of the LSTM layer, 50 neurons have been considered. There are only 2 hidden layers of the neurons, each containing 32 neurons in them. The model has been created sequentially, and the model consists of six layers and contains two dropout layers. It has all its layers connected. The other tool to be provided will be used to fit the modal. Using a simple RNN neural network, the dropping of the layer by 0.01%, the creation of input sequences and batches to fit the modal, the input sequence is 60 characters, while the batch means sample correction. In this case, the time series input sequence is the sample. We hence infer that the batch size is 32 samples, and the dropout rate is 0.20. This sample is made up of a time series input sequence. Using Keras, there is no need to create batches manually as such. The data points are indexed at 60 points: Data points range from 1 to 60. We can hence predict the value at index 61 based on this. We then used 100 epochs for the time series and 40 for the forecast.

Summary of Core Parameters: Network Design

1. Data set used: Stock price dataset
2. Percentage of data for testing – Test percentage is 10%
3. How many layers of LSTM – 2 Layer LSTM
4. Number of neurons in each LSTM layer - 25
5. Any Dropout layers? Yes
6. If with Dropout layer: the percentage to dropout is 20%
7. Length of the time-series input sequence = 500
8. Batch size for training = 32
9. Batch size for testing = 1
10. Number of epochs for training = 100
11. Model used: LSTM Kera Sequential

**6. PART V: Redesign the LSTM Neural Network (10 Points)**

Using critical thinking and the experiences of working with time-series datasets, the student should propose some changes to the above network (PART IV) or the network training process, with which the network performance may be improved.

There are many different series. First of all, we have to import basic libraries. Matplotlib is for visualization. We will use a specially unique dropout time series RNN and it's fully connected. In MLP, there are several layers. We will use a time series generator in order to create the time series input that we have to import from Keras. Also, we need Min Max scaler for scaling. What we want for a simple RNN is some data that should be sequenced. It's some kind of time series basically. I'm going to use stock data in this case. This stock has a lot of information.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

It visualizes the shapes. This database contains 7 attributes and 1980 shapes. k All the attributes are continuous. It is mentioned that the volume is a discrete integer. The minimum, standard deviation, and maximum are provided in the numeric property summary.

A graph with blue lines

Description automatically generated

We will deal with a time series relating data point and length. Now, suppose an input of a 60-point time series is given. We need to split the dataset into some for training and some for testing. Now let's say the number of data points in the dataset are 1980. That number is the length of the data frame: it is the number that includes the complete length of the sequence now. This gives a total of 1980 rows in this dataset. Secondly, we need to know under what index each of these rows has in the whole dataset consisting of 2054 rows. Indexing should pre-compute the subtraction of the length of test data so that it can easily split the dataset later on. The indexing of the rows was done from 604 through to the end of the dataset. Now, in splitting data, we will be making use of the iloc function. The index, therefore, will be 1979.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

This test data includes 50 historic data points to make the prediction. Now, it should print information to record 5 data and last 5 records of training. The first index is zero and the last index is 1979.

Now, the data should be normalized into ranges from 0 to 1. We can do better. We will use the Minmax scaler class and tool. Hence, if we find a fit with the data training we set it up and normalize the data. To make the training of modal we have to develop a time series generator. The input is normally fed into the modal in batches, though batch could be an option, it means sample correction. Here, sample meant the time series input sequence. So there are 40 samples composing the batch. A time series input sequence composes this sample. Because using of keras, so it is not practically manual creation of batches. Data points index are 60-point. Data points show a range from 1 to 60.

A screen shot of a graph

Description automatically generated

The number of the feature should be specified at the phases of the project. The "features" here mean the number of attributes or variables in the dataset, and the closing feature is 1. Because the straightforward RNN layer which Keras provides will later be used, we will use first the LSTM layer in designing the neural network. And also a new layer called a dropout will be used in this LSTM neural network. We LSTM-ﬁt times series. This single time series is used to generate the output. This output would be given to the next layer. Still, another layer would be having the feature output. For example, by looking at 1100, one can mark the percent dropped which usually in a range of 10 to 15. To make things easy for the neural networks we will develop Keras in steps. In order to feed data to another layer one should make previous configuration for the return sequence. Finally, define the shape for a 3-dimensional input; we focus on the 2-dimensional, and the extra batch size that needs to be provided.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Here, we call the fit standard function. In all, there are 70 epoch IDs. In each phase, 40 batches shall fit in the input sequence. We prepare for 40 steps. First runs will run 40 batches of the input sequence, and the loss is 0.0011. Loss decreases in magnitude.

We must see the modal. In the training process, a Modal makes a prediction. We learn the history of modal. We frame the latest value. Now we'll see the way the mode when it is in a training process will carry out the training process of it. We need to validate the values. Also, we will be needing a test data for testing the modal. And further preprocessed test data in a batch size of 1. This means each batch contains one input sequence. To test the model, feed in the input time series to a generator. For every input, there are four parameters. So, an input sequence line contains 60 data points. Later, we will train the data, change the model to forecast into a time series, and the predicted value has to be transferred then into an unnormalized value.

A graph with blue lines and orange lines

Description automatically generated

**Summary of Core Parameters: Network Design**

1. Data set used: Stock price dataset
2. Percentage of data for testing – Test percentage is 10%
3. How many layers of LSTM – 2 Layer LSTM
4. Number of neurons in each LSTM layer - 50
5. Any Dropout layers? Yes
6. If with Dropout layer: the percentage to dropout is 20%
7. Length of the time-series input sequence = 2479
8. Batch size for training = 40
9. Batch size for testing = 1
10. Number of epochs for training = 70
11. Model used: LSTM Kera Sequential

**7. PART VI: Compare Network Performance (10 Points)**

--) Compare their performance based on the design and the core parameters applied to the LSTM neural network in PART IV and the network in PART V.

--) Write a report on the results of comparing the performance of the two networks and to provide a reasonable explanation to such results: Some improvement has been achieved. OR no significant change in the performance has been observed.

LSTMs are a variant of RNN and are very famous regarding long-term memory. A comparison is presented between the two regarding performance variation of the networks with regard to a set of variables that include dataset size and quantitative evaluation in terms of training time for a model. The results showed that, among the various algorithms reviewed, LSTM outperformed others in processing the same dataset by training time. The test dataset time steps have been covered one by one in parts IV and V below. It will go ahead take the actual projected value of the test set and feed it to the model for a forecast on the next time step. This is actually simulated to visualize a situation where stock market data is collected on a yearly basis and then used to make a prediction for the next month.

Now, to replicate, we will use the structure of the train and test datasets. It will involve the test dataset to compile all the forecasts. Later, an LSTM model best with 1 neuron will be fitted for 50 epochs. The batch size shall be 1 because we are going to perform a walk-forward validation, doing one-step forecasts for each of the last 12 months of data.

Therefore, online training, when a batch size has a value of 1, fits the model, and unlike batch training or mini-batch training, it has some fluctuation in the fit. We now modify the model by adjusting the following parameters: The results changed very little, and no remarkable change in performance was noticed.

* Percentage of data for testing - Increase test share from 10% to 20%
* Numbers of neurons in each LSTM layer: From 25 to 50
* With Dropout layer: if used, the dropout percentage increases from 10% to 20%
* Batch size for training: Increased from 32 to 40
* Number of epochs for training: increased from 40 to 100

**8. PART VII: Project Report (20 Points)**

**1 Write an introduction section to introduce the project**

RNNs are a class of neural networks where the output at one step is fed as input to the next. While in standard neural networks, every input and every output participates independently, in some applications, previous words are needed, for instance, predicting the next word of a phrase; thus, it is necessary to remember previous words. RNN was hence devised and it used a Hidden Layer so as to get over the problem. The pivotal unit of RNN is the Hidden state, it memorizes certain information in a sequence. Over time an RNN remembers everything. Since it would remember all the previous feeds, it would be useful only in the Time series prediction. LSTM: Long Short-Term Memory. The effective pixel neighborhood is further increased using even convolutional layers with the help of recurrent neural networks.

The LSTM recurrent unit has the purpose of "remembering" all of the previous information of the network, yet also "forgetting" information that isn't important. It does this by introducing a number of "gates," or levels of activation functions, for different purposes. Every LSTM recurrent unit maintains an Internal Cell State vector that суммаризует effective data that the unit in front of it decided to keep.

The key difference between RNN and LSTM architecture: the hidden layer in LSTM is a unit that contains gates, or a so-called gated cell. This involves four layers which work together in order to output not only the output of the cell but also its state; both of these items, in turn, are moved further on to the next hidden layer. Unlike RNNs that have only one layer of tanh, LSTMs are comprised of three logistic sigmoid gates and a single layer. A gate has been added in order to regulate how much data can flow in through the cell. It shall check how much information would be required at the next cell and how much it should drop. The range of output usually lies between 0 and 1, and it ranges from '0' - 'reject all' - to '1', 'includes it all '.

**2 Describe what the student has done in PART II, III, IV, and V in the final project**

In time series forecasting, it has to consider data correlation and pattern; hence, it depicts that it provides the outcome for the predictions on sequential data much like the functions of the human brain. While on the other hand, part II should capture the presence of words "but" and "terribly exciting"; by looking at the whole sequence, an RNN can then conclude that the meaning of this phrase has flipped from negative to positive. This reading of a full sequence provides us the context to make sense of it-just the basic principle of recurrent neural networks.

Although we changed the model in Parts IV and V, despite minimal changes in the parameters, hardly anything changed. Part III: LSTM is designed to be much better at catching the long-term dependencies, with different parts of states, and a hidden state encodes feedback information along with the original state and information about the past states.

**3 List what the student has learned from the models (Simple RNN and LSTM) and what he/she has experienced while working with them (build, train, test, and forecast)**

A recurrent neural network models time or sequence-based data. The applications involving RNNs range from NLP, stock price forecasting, energy demand forecasting, and many others. Recurrent neural networks can have the tendency to pick up features from sequences because they carry the hidden state from one step to another and mix up this hidden state with the input. One such variant of RNN, that consists of special and normal units, is the LSTM network.

What does it all the units matter?

This helps an LSTM unit's memory cell to remember information for a very long time. LSTMs are useful when the neural network needs to switch between using the current input and older information to make a prediction. One problem is that the recurrent neural network is biased towards the short-term memory.

For example, in the case of the flower, the tree that sandwiched the bear and the wolf, it fails to remember the priority of the wolf and the bear. That is, RNNs do not have a special capability to make use of and store long-term memory. At the place of RNN, LSTM networks can be better fit for the task, as they use long- and short-term memory.

**RNN works:**

* The entering is done by memory, joining to the present situation.
* The output is the projection of the future state of the input.
* It also includes in input the output for the neural network in its next generation.
* On the other hand, the LSTM does the following: This neural network remembers the long-term memory, short-term memory, and both.
* Every step in the cycle requires the interaction between long and short-term memories. Thus, our memories of the short term, long term, and predictive kind are strengthened.

**4 Write a conclusion section to conclude the project report.**

Based on this research and course, we learned about RNN, its pros and cons, and how LSTM differs and overcomes the weak points compared to the plain vanilla RNN model. LSTMs are often referred to as "advanced RNNs." Vanilla RNNs: RNNs not making use of the cell state, they only have hidden states, and the RNN does store information in those hidden states. On the other hand, LSTM also contains a hidden state besides cell states. The cell state is regulated by "gates", which has the authority to add or remove information from the cell. Theoretically, this "cell" should enable LSTM to support long-term dependency.

Also, which one of these has the long-term memory of information? Also, while moving from RNN to LSTM-Long Short-Term Memory, the control mechanism will regularize increasingly the flow and mixing of inputs per training weights. Because of this, the outputs would be much easier to handle. Thus, LSTM provides the most control and hence better results.

These come, however, at the expense of higher running expenses and complexity. In this regard, another project imported a data file divided into portions for training and testing of the model. 'How to generate simultaneously an LSTM model with several parameters drop out, number of layers, thick layer, etc.

**9. PART VIII: Final Presentation Videos: YouTube Links**

[**https://youtu.be/sCLqwr0IC1E**](https://youtu.be/sCLqwr0IC1E)

[final project presentation video Yog Chaudhary (youtube.com)](https://www.youtube.com/watch?v=i_ICt1cXQVc)

**Thank You So Much**