CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

This chapter explains about the implementation of the program. The data is already in CSV format and processed using Python 2.7 platform. First of all after the program reads the CSV file is to normalization data. After normalization data, then randomize weights for all variables (date, temperature, price sales) and hidden layer

Below is the code to allow python read csv file.

```
import csv
with open('datasetsku.csv') as csvfile:
  readCSV = csv.reader(csvfile, delimiter=',')
  line = 0
  datasets = []
  date = []
  selling = []
  total_sales = []
  temper = []
  stock = []
  countrows = list(readCSV)
  for row in tqdm(countrows, desc='Data Processed: '):
    if line == 0:
      line += 1
    else:
      if datasets is None:
      date = row[0]
      selling = row[1]
      total_sales = row[2]
      temper = row[3]
      stock = row[4]
      datasets = {'Date' : [date],
                  'Selling' : [selling],
                  'total_sales' : [total_sales],
                  'temperature' : [temper],
                  'stock' : [stock]}
      else:
        date = row[0]
        selling = row[1]
        total_sales = row[2]
        temper = row[3]
        stock = row[4]
```

```
33.         datasets1 = {'Date' : [date],
34.                         'Selling' : [selling],
35.                         'total_sales' : [total_sales],
36.                         'temperature' : [temper],
37.                         'stock' : [stock]}
38.         datasets.append(datasets1)
```

The first line to allow python read csv file.line 2 to open the csv file named datasetsku.csv. line 3 to read the datasetsku.csv file. lines 4 until 38 to create an array that save the contents of the datasetsku.csv file.

Below is the code to normalized data.

```
39. mindate = min(superdate)
40. maxdate = max(superdate)
41. minselling_price = min(superselling_price)
42. maxselling_price = max(superselling_price)
43. mintotal_sales = min(supertotal)
44. maxtotal_sales = max(supertotal)
45. mintemperature = min(supertemperature)
46. maxtemperature = max(supertemperature)
47. minstock = min(superstock)
48. maxstock = max(superstock)
49. no_date = []
50. no_selling = []
51. no_total = []
52. no_tempe = []
53. no_stock = []
54. iz = 0
55. for looping in tqdm(datasets, desc='Data Normalization: '):
56.       normal_date = (superdate[iz] - mindate) / (maxdate - mindate)
57.       normal_selling = (superselling_price[iz] - minselling_price) / (maxselling_price - minselling_price)
58.       normal_total = (supertotal[iz] - mintotal_sales) / (maxtotal_sales - mintotal_sales)
59.       normal_temperature = (supertemperature[iz] - mintemperature) / (maxtemperature - mintemperature)
60.       normal_stock = (superstock[iz] - minstock) / (maxstock - minstock)
61.     no_date.append(normal_date)
62.     no_selling.append(normal_selling)
63.     no_total.append(normal_total)
64.     no_tempe.append(normal_temperature)
65.     no_stock.append(normal_stock)
66.     iz += 1
```

Lines 39 until 48 for look the minimum and maximum values of data that has been saved in arrays called datasets. Lines 49 until 53 create a variable for each data. Lines 54 until 66 to normalize the data that has been saved in each array.

Bellow is the code to randomize the weight of all variable

```
67.     for i in range(n_hiddenlayer):
68.         tempArrWeight = []
69.         for j in range(n_var + 1):
70.             tempWeightInput = random.random()
71.             tempArrWeight.append(tempWeightInput)
72.         WeightInput.append(tempArrWeight)
```

Lines 67 and 69 contain looping to randomize all the variables as many as multiple hidden layers. Line 68 to create a temporary variable to contain the weight. Lines 70 until 72 to get a random weight and enter it into an array named WeightInput.

Bellow is the code to randomize the weight for hidden layers

```
73.     for i in range(n_hiddenlayer + 1):
74.         tempWeightHidden = random.random()
75.         WeightHidden.append(tempWeightHidden)
```

Line 73 for looping as many hidden layers that been input and add bias. Lines 74 and 75 to randomize weight and saved it in an array named WeightHidden.

After randomize the weights, the algorithm first performs a process named forward propagate. In this process, the goal is to calculate the output and compare it with the expected outputs that have been set or input, but before being input to each neuron in hidden layers, the neuron must be activated first.

Bellow is the code for activated each neuron to hidden layer

```
76. for j in range(len(WeightInput)):
77.     tempArrWeight = WeightInput[j]
78.     activation = 0.0
79.     transfer_activation = 0.0
80.     for k in range(len(tempArrWeight)):
81.         tempWeight = tempArrWeight[k]
82.         tempInput = tempArrInput[k]
83.         tempActivation = tempWeight * tempInput
84.         activation = activation + tempActivation
85.         transfer_activation = 1.0 / (1.0 + pow(exp(1),
    (activation * -1 )))
86.         tempArrHidden.append(transfer_activation)
```

Line 76 for looping as many hidden layers. Line 77 to hold the weight to temporary named tempArrWeigh.Lines 81 and 82 to saved weights from the input and hidden layer weights. Line 83 for multiply weight input and hidden layer. Line 84 is to add multiply results with bias. Line 85 is to activate the hidden layer's weight and get hidden layer value.Line 86 to saved the calculation results into an array named tempArrHidden.

Bellow is the code to activated each neuron to output layer.

```
87. output_net = 0.0
88. for l in range(len(WeightHidden)):
89.     tempWeightHidden = WeightHidden[l]
90.     hidden_value = tempArrHidden[l]
91.     temp_output_net = tempWeightHidden * hidden_value
92.     output_net = output_net + temp_output_net
93.     output = 1.0 / (1.0 + pow(exp(1),(output_net * -1 )))
94.     error = ((Input4 - output)**2)
```

To calculate the output formula is not different from calculating the hidden layer weight. Line 88 for looping as many hidden layers as input. Line 90 for multiply weight and value in hidden layer. Line 92 is to add multiply results with bias. Line 93 is to get the output value.

After calculating the output, the next step is to calculate the error. The formula for calculating the error is "expectation - the output raised 2". Formula for calculating the error can be seen in line 94. If the error smaller than the target error, then the loop will stopped and saved the weight during the loop stops as the best weight and it used for the data testing process, but if the error is greater than the target, the repeat error is continued and the weight will be recalculated using a formula that involves the learning rate of AI.

Bellow is the code for calculating the new weight for hidden layer.

```
95. delta = (Input4 - output) * output * (1 - output)
96. for p in range(len(tempArrHidden)):
97.     hidden_value = tempArrHidden[p]
98.     delta_weight = lrate * delta * hidden_value
99.     currWeight = WeightHidden[p]
100.        newWeight = currWeight + delta_weight
101.    newHiddenWeight.append(newWeight)
```

Line 95 for counting delta. Line 96 for looping as many as hidden layer. Line 97 and line 99 to get weight hidden layer and the value of the hidden layer.Line 98 is a formula to calculate the increment or the difference between the current weight and the best weight.Line 100 is a formula to add the current weight with the delta weight to make it closer to the best weight.Line 101 to saved a new weights to the newHiddenWeight array.

After the new hidden layer calculated, the next step is to calculate the new weights for all variables.

Bellow is the code for calculating the new weight for all variable.

```
102.  for q in range(n_hiddenlayer):
103.      tempWeightHidden = WeightHidden[q+1]
104.      delta_net = delta * tempWeightHidden
105.      hiddenValue = tempArrHidden[q+1]
106.      delta_hidden = delta_net * hiddenValue * ( 1 - hiddenValue)
107.      arrCurrWeight = WeightInput[j]
108.      arrNewWeight = []
109.      for r in range(len(tempArrInput)):
110.          inputValue = tempArrInput[r]
111.          delta_weight = lrate * delta_hidden * inputValue
112.          currWeight = arrCurrWeight[r]
113.          newWeight = currWeight + delta_weight
114.          arrNewWeight.append(newWeight)
115.      newInputWeight.append(arrNewWeight)
```

To calculate new weights, all variables are basically the same as calculating the new weights for hidden layers. The difference from this calculation is the delta or increment that be used is different from the delta that calculated on the hidden layer.first, delta_net must be calculated from delta multiplication with the current hidden weight of a hidden layer the code written in line 103. After that, delta_hidden calculated using delta_net then multiplied by the hidden layer value multiply. The code written in line 106. And to calculate the delta_weight multiply the alpha (learning rate), delta_hidden and value of the variable.

Bellow is the code to save new weight for testing data

```
116.  WeightHidden = newHiddenWeight
117.  WeightInput = newInputWeight
```

Lines 116 and 117 to store new weights that have been calculated.This step will be repeated with forward propagation until the error is smaller than target error or epoch have reached limit. To get the total sales and stock output, you have to do training twice to get the best weight to calculate the total sales and stock predictions. The only difference is the variable used by total sales and stock.

The testing process in the program only using forward propagate process. After the output has been calculated, the error also calculated. And the error is needed to calculate the error rate (MSE). The formula to calculate the error rate is square of error. If the error rate is smaller than the targeted error rate,then this algorithm is successful.

Bellow is the code to calculate the error rate

```
118.   temperror_total = math.sqrt(error_total_sales[0])
119.   temperror_stock = math.sqrt(error_stock_sales[0])
```

Lines 118 and 119 to calculate the error rate of total sales and stock.setelah itu output yang di keluarkan di denormalisasi.

Bellow is the code to denormalization

```
120.   realOutput = (output * (max_stock - min_stock)) + min_stock;
121.   realOutput = (output * (maxtotal_sales - mintotal_sales)) + mintotal_sales;
```

formula for denormalization is (output * (maxD – minD)) + minD. Line 120 for calculate stock output prediction and 121 for calculate total sales output prediction.

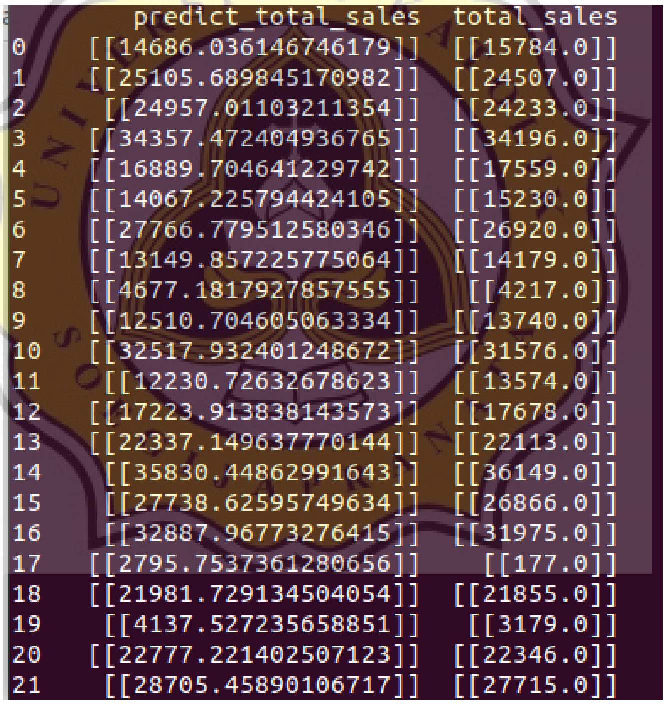
5.2 Testing

Testing is complete with hidden layer, max epoch, learning rate, the target error inputed by the user and 365 data. Below this is some data that has been tested and output has been denormalized.

Table 5.1: Table Prediction Result

Date	Sales Price	Temperature	Total Sales	Predict Total Sales	Stock	Predict Stock
01/04/2019	11800	35	15784	14686.036146746179	19680	18881.965592161592
02/04/2019	12000	38	24507	25105.689845170982	26110	26491.793774620124
03/04/2019	11500	36	24233	24957.01103211354	24380	24609.75999603692
04/04/2019	12000	36	34196	34357.472404936765	35500	35954.235735603994
05/04/2019	11500	33	17559	16889.704641229742	23040	22986.76448972044
06/04/2019	11800	36	15230	14067.225794424105	20400	19743.390996906328
07/04/2019	12000	33	26920	27766.779512580346	28660	29347.83509160447
08/04/2019	11000	33	14179	13149.857225775064	20370	19960.981442845798
09/04/2019	11500	36	4217	4677.1817927857555	7920	7753.304555534092
10/04/2019	11500	34	13740	12510.704605063334	13980	12757.420209003212
11/04/2019	11000	30	31576	32517.932401248672	32750	33778.403843228385
...	...	...	...	...	...	...
24/03/2020	12000	37	13198	12477.61012527671	17820	16963.0621695331
25/03/2020	12000	31	19013	19260.611230398183	24840	25240.75976072743

26/03/2020	11000	30	20496	21445.510400390158	21720	21845.181967524844
27/03/2020	12000	32	17633	17588.772484254718	18170	17330.262837293943
28/03/2020	11500	38	16197	16089.12766488111	17230	16469.16979128524
29/03/2020	12000	38	12121	11355.978035295542	13580	12487.804350868459
30/03/2020	11000	35	3237	4584.63884605142	3710	5555.537836916685



	predict_total_sales	total_sales
0	[[14686.036146746179]]	[[15784.0]]
1	[[25105.689845170982]]	[[24507.0]]
2	[[24957.01103211354]]	[[24233.0]]
3	[[34357.472404936765]]	[[34196.0]]
4	[[16889.704641229742]]	[[17559.0]]
5	[[14067.225794424105]]	[[15230.0]]
6	[[27766.779512580346]]	[[26920.0]]
7	[[13149.857225775064]]	[[14179.0]]
8	[[4677.1817927857555]]	[[4217.0]]
9	[[12510.704605063334]]	[[13740.0]]
10	[[32517.932401248672]]	[[31576.0]]
11	[[12230.72632678623]]	[[13574.0]]
12	[[17223.913838143573]]	[[17678.0]]
13	[[22337.149637770144]]	[[22113.0]]
14	[[35830.44862991643]]	[[36149.0]]
15	[[27738.62595749634]]	[[26866.0]]
16	[[32887.96773276415]]	[[31975.0]]
17	[[2795.7537361280656]]	[[177.0]]
18	[[21981.729134504054]]	[[21855.0]]
19	[[4137.527235658851]]	[[3179.0]]
20	[[22777.221402507123]]	[[22346.0]]
21	[[28705.45890106717]]	[[27715.0]]

Illustration 5.1: Result Prediction total sales

	predict_stock	stock
0	[[18881.965592161592]]	[[19680.0]]
1	[[26491.793774620124]]	[[26110.0]]
2	[[24609.75999603692]]	[[24380.0]]
3	[[35954.235735603994]]	[[35500.0]]
4	[[22986.76448972044]]	[[23040.0]]
5	[[19743.390996906328]]	[[20400.0]]
6	[[29347.83509160447]]	[[28660.0]]
7	[[19960.981442845798]]	[[20370.0]]
8	[[7753.304555534092]]	[[7920.0]]
9	[[12757.420209003212]]	[[13980.0]]
10	[[33778.403843228385]]	[[32750.0]]
11	[[12368.223207337052]]	[[13670.0]]
12	[[18150.640359470202]]	[[18840.0]]
13	[[22366.88851090065]]	[[22610.0]]
14	[[36807.598550226925]]	[[36420.0]]
15	[[34674.273267104705]]	[[33990.0]]
16	[[33981.658067067976]]	[[32930.0]]
17	[[7463.111656869255]]	[[7640.0]]
18	[[24754.28272544607]]	[[24640.0]]
19	[[5367.259896199805]]	[[3700.0]]
20	[[32412.256574599483]]	[[31420.0]]

Illustration 5.2: Result Prediction stock

```
346  [[29183.945071813032]]  [[28110.0]]
347  [[32790.944543239704]]  [[31680.0]]
348    [[33208.31798965602]]  [[32170.0]]
349    [[9007.379158608961]]    [[9520.0]]
350  [[26932.082668226303]]  [[26280.0]]
351    [[33648.02471482039]]  [[32570.0]]
352    [[27025.71355235279]]  [[26330.0]]
353  [[26128.990992819443]]  [[25500.0]]
354    [[23212.66823280533]]  [[23080.0]]
355     [[35063.6120210775]]  [[34200.0]]
356     [[26115.4382370733]]  [[25430.0]]
357    [[6931.506604912095]]    [[6550.0]]
358     [[16963.0621695331]]  [[17820.0]]
359    [[25240.75976072743]]  [[24840.0]]
360  [[21845.181967524844]]  [[21720.0]]
361  [[17330.262837293943]]  [[18170.0]]
362    [[16469.16979128524]]  [[17230.0]]
363  [[12487.804350868459]]  [[13580.0]]
364    [[5555.537836916685]]    [[3710.0]]

[365 rows x 2 columns]
Accuracy = [92.4832308]%,MSE Total Sales Prediction = 3.0% ,MSE Stock Prediction
 = 3.7%
```

Illustration 5.3: Result Accuracy and MSE

From the result above, the prediction is still in range of the error tolerance. The error from that result is 3.0% and 3.7%. because the results are less than 5%, then this study can be said to be successful with an accuracy of 92.4832308%.