CHAPTER 1: INTRODUCTION

1.1 Introduction

Diabetes is a disease caused by hyperglycemia (high blood glucose level). Diabetes affects an estimated 3-4% of the world's population (half of whom are undiagnosed), making it one of the major chronic illnesses prevailing today. It is caused by hyperglycemia resulting from defects in insulin secretion, insulin action, or both. The chronic hyperglycemia of diabetes is associated with long-term damage, dysfunction, and failure of various organs, especially the eyes, kidneys, nerves, heart, and blood vessels. This deficiency leads to destruction of the b-cells of the pancreas with consequent insulin deficiency to abnormalities that result in resistance to insulin action and reaction process. Deficiency of insulin results from inadequate insulin secretion. This Improper insulin secretion and defects in insulin action is the primary cause of hyperglycemia. So, the significance of insulin dose is clearly visible.

In this paper, we have taken the challenge of predicting insulin chart for Diabetic patients. We have taken thirty-six month's data (insulin chart) for code 33 (Regular insulin dose) of a patient. Assuming thirty-six month's data as training data, next thirty-six month's charts is predicted and compared with the actual data. We have used RNN model to predict the insulin chart by taking the data of a diabetic patient. RNN has solved variety of problems such as- speech recognition, language modeling, translation, image captioning etc. RNN is mainly used for where the gap between the relevant information and the place that it's needed is small. But there are also cases where we need more contexts. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information. We have solved this problem by using a special kind of RNN called LSTM (Long Short Term Memory) which is designed to avoid the long-term dependency problem. We have trained 67% &tested 33% of our given dataset for RNN. We use RNN for solving sequence of insulin level of hospital patient at a time. ANN is a popular way to identify unknown and hidden patterns in data which is suitable for predicting insulin data. For the prediction of insulin level, we use 80% train on data and 20% predict on the train data. We use Predictive Apriori for searching if one takes insulin at breakfast & at lunch, then what will be the accuracy to take in insulin at dinner time. Predictive Apriori mining is the association rule for prediction of insulin level.

1.2 Motivation

Diabetes prevalence has been rising more rapidly in middle and low-income countries. The number of people with diabetes has risen from 108 million in 1980 to 422 million in 2014. The global prevalence of diabetes among adults over 18 years of age has risen from 4.7% in 1980 to 8.5% in 2014. A lack of insulin, or an inability to adequately respond to insulin, can each lead to the development of the symptoms of diabetes.

For a diabetes patient, insulin dose is necessary to control the level of glucose. The doctor of the patient also has to know the required insulin dose from previous records of doses & from patient's current calculated blood sugar level. This has inspired us to make a research on how to predict the insulin dose level of a patient before every meal.

The prediction we have done through RNN has produced a very good result. Similar kind of insulin chart prediction for diabetes patients by Ravindra Nath was completed in December, 2013 using HMM (Hidden Markov Model). Rivindra's sequence prediction was great but it was time consuming. We have made that sequence prediction faster using Recurrent Neural Network. Also, HMM doesn't work for lot of data. LSTM in RNN addresses this issue & designed to perform quite well for long term dependencies and works faster. LSTM maximizes the prediction accuracy rate by minimizing error at each iteration.

Lastly, we want to tell that there are many researches going on to discover newer methods to predict insulin dose. Despite having all those methods, we believe our research data will help doctors to predict almost accurate insulin dose of diabetes patients.

1.3 Objective

The specific objectives of our project are as follows:

a) Insulin level sequence prediction using recurrent neural network long short term memory.

b) Discover meaningful pattern and frequent insulin level using Predictive Apriori Algorithm.

c) Prediction on train data with loss function and accuracy using Neural Network.

1.4 Contribution

Contributions in the project are as follows:

For our research, at first, we have preprocessed our raw data. After that, we have used neural network to find out the data loss rate & the data accuracy rate of our predicted insulin dose. Again, we have implemented LSTM (special kind of RNN used for long term dependency) to predict the insulin dose for a patient before every meal. After all, we have generated predictive Apriori algorithm to determine the most frequently taken insulin doses by the patients over full day period.

1.5 Outline

Chapter 1: This chapter represents about the motivation to work, specify the objectives and then the contribution that we have made.

Chapter 2: Description about Machine learning, deep learning and also about the algorithms RNN (LSTM), ANN and Predictive Apriori that we have implemented.

Chapter 3: Methodology. It shows the architectural view of our work.

Chapter 4: Implementation& flow chart of our research. The tools that have been used in the project.

Chapter 5: Result analysis. By generating different graphs, we have shown our prediction accuracy.

Chapter 6: At last in this chapter we have summarized our work and have pointed out our future plans.

CHAPTER 2: BACKGROUND STUDY

2.1 Background Study

Artificial Intelligence (AI) and Machine Learning (ML) are two very hot trendy expressions at this moment, and frequently seem to be used interchangeably. They are not exactly a similar thing, but the perception that they are can sometimes lead to some confusion. Broadly speaking, Artificial Intelligence is the extensive idea of machines being able to carry out tasks in a way that we would consider "smart". On the other hand, Machine learning is a utilization of manmade brainpower (AI) that provides systems the capacity to automatically learn and improve from experience without being explicitly programmed.

Machine learning centers around the improvement of PC programs that can get to information and utilize it learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Two of the most widely adopted machine learning methods are supervised learning and unsupervised learning – but there are also other methods of machine learning. Here's an overview of the most popular types.

- **Supervised Learning:** These algorithms are trained using labeled examples, such as an input where the desired output is known. For example, a piece of equipment could have data points labeled either "F" (failed) or "R" (runs). The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors. It then modifies the model accordingly. Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabeled data. Supervised learning is commonly used in applications where historical data predicts likely future events. For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim.

- **Unsupervised Learning:** This is used against data that has no historical labels. The system is not told the "right answer." The algorithm must figure out what is being shown. The goal is to explore the data and find some structure within. Unsupervised learning works well on transactional data. For example, it can identify segments of customers with similar attributes who can then be treated similarly in marketing campaigns. Or it can find the main attributes that separate customer segments from each other. Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decomposition. These algorithms are also used to segment text topics, recommend items and identify data outliers.

- **Semi-supervised Learning:** This algorithm is used for the same applications as supervised learning. But it uses both labeled and unlabeled data for training – typically a small amount of labeled data with a large amount of unlabeled data (because unlabeled data is less expensive and takes less effort to acquire). This type of learning can be used with methods such as classification, regression and prediction. Semi supervised learning is useful when the cost associated with labeling is too high to allow for a fully labeled training process. Early examples of this include identifying a person's face on a web cam.

- **Reinforcement learning:** It is often used for robotics, gaming and navigation. With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards. This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do). The objective is for the agent to choose actions that maximize the expected reward over a given amount of time. The agent will reach the goal much faster by following a good policy. So the goal in reinforcement learning is to learn the best policy.

Deep Learning is another concept of machine learning which is actually a technique for implementing machine learning. In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets,

TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

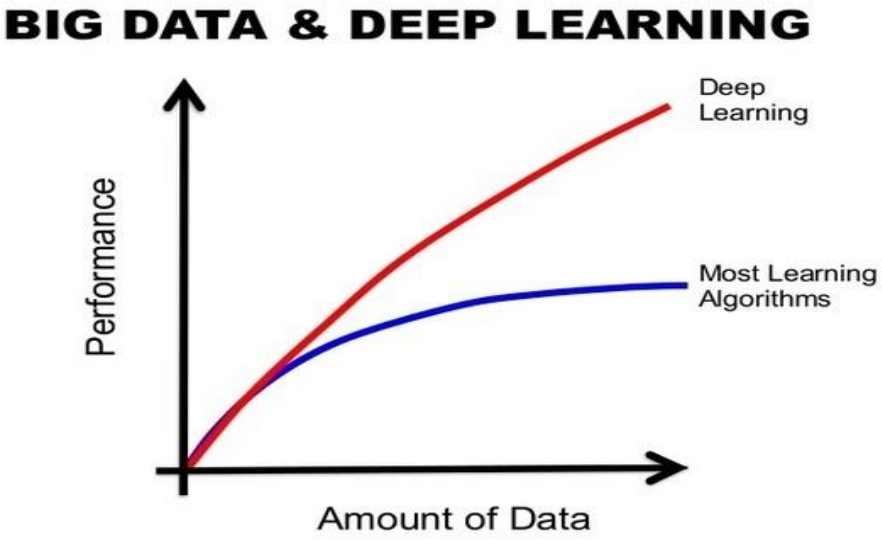


Figure: Performance comparison between Deep Learning & other Machine Learning Algorithms as illustrated in [9, Fig 2.1]

2.2 Artificial Neural Network (ANN)

The term 'Neural' is derived from the human (animal) nervous system's basic functional unit 'neuron' or nerve cells which are present in the brain and other parts of the human (animal) body. The development of neural networks has been key to teaching computers to think and understand the world in the way we do, while retaining the innate advantages they hold over us such as speed, accuracy and lack of bias.

A Neural Network is a computer system designed to work by classifying information in the same way a human brain does. It can be taught to recognize, for example, images, and classify them according to elements they contain. Essentially it works on a system of probability – based on data fed to it, it is able to make statements, decisions or predictions with a degree of certainty.

Artificial Neural Networks are the biologically inspired simulations performed on the computer to perform certain specific tasks like clustering, classification, pattern recognition etc. A neural

network acquires knowledge through learning & this knowledge is stored within inter-neuron connection strengths known as synaptic weights.

- **Artificial Neural Network's working method:**

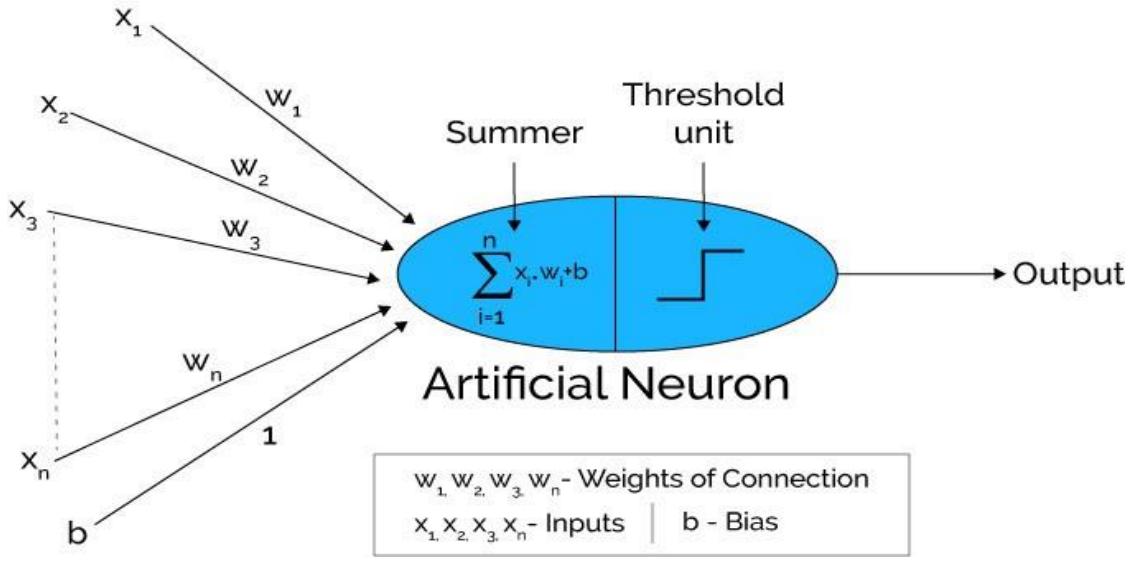


Figure: Working method ofArtificial Neural Network as illustrated in [4, Fig 2.2 (a)]

Artificial neural networks can be viewed as weighted directed graphs in which artificial neurons are nodes and directed edges with weights are connections between neuron outputs and neuron inputs.

The Artificial Neural Network receives input from the external world in the form of pattern and image in vector form. These inputs are mathematically designated by the notation x(n) for n number of inputs.

Each input is multiplied by its corresponding weights. Weights are the information used by the neural network to solve a problem. Typically, weight represents the strength of the interconnection between neurons inside the neural network.

The weighted inputs are all summed up inside computing unit (artificial neuron). In case the weighted sum is zero, bias is added to make the output not- zero or to scale up the system response. Bias has the weight and input always equal to '1'.

The sum corresponds to any numerical value ranging from 0 to infinity. In order to limit the response to arrive at desired value, the threshold value is set up. For this, the sum is passed through activation function.

The activation function is set of the transfer function used to get desired output. There are linear as well as the non-linear activation function.

Some of the commonly used activation function are—binary, sigmoidal (linear) and tan hyperbolic sigmoidal functions (nonlinear).

Binary—the output has only two values either 0 or 1. For this, the threshold value is set up. If the net weighted input is greater than 1, an output is assumed 1 otherwise zero.

Sigmoidal Hyperbolic—this function has 'S' shaped curve. Here tan hyperbolic function is used to approximate output from net input. The function is defined as—f (x) = (1/1+ exp (-$\boldsymbol{\sigma}$x)) where $\boldsymbol{\sigma}$—steepness parameter.

- **Architecture of Artificial Neural Networks:**

A typical neural network contains a large number of artificial neurons called units arranged in a series of layers. In typical artificial neural network, comprises a different layer –

Input layer: It contains those units (artificial neurons) which receive input from the outside world on which network will learn, recognize about or otherwise process.

Output layer: It contains units that respond to the information about how it's learned any task.

Hidden layer: These units are in between input and output layers. The job of hidden layer is to transform the input into something that output unit can use in some way.

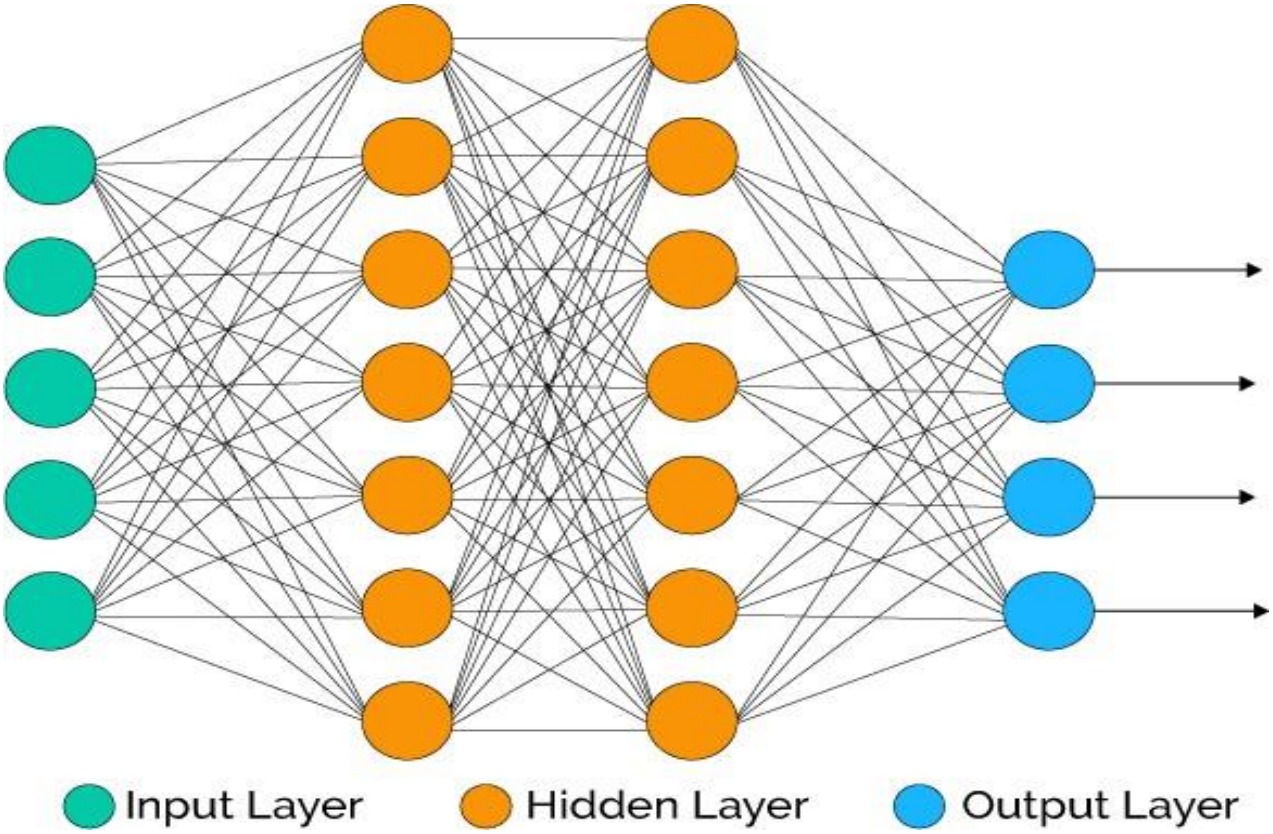


Figure: Architecture of Artificial Neural Networks as illustrated in [4, Fig 2.2 (b)]

Most neural networks are fully connected that means to say each hidden neuron is fully connected to every neuron in its previous layer (input) and to the next layer (output) layer.

2.3 Recurrent Neural Network (RNN)

Suppose, A Bangladeshi guy living in the U.S. He has a constant flow of money from home to U.S. and vice versa. If the USD is stronger in the market, then the Bangladeshi Currency (Taka)goes down, hence, a person from Bangladesh buys a dollar for more takas. If the dollar is weaker, one needs to spend fewer rupees to buy the same dollar. Predicting how much a dollar will cost tomorrow can guide our decision making and can be very important in minimizing risks and maximizing returns. Traditional neural networks can't do this, and it seems like a major shortcoming. It's unclear how a traditional neural network could use its previous record to predict the U.S. dollar rate for this currency exchange event.

Again, for example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases,

where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.

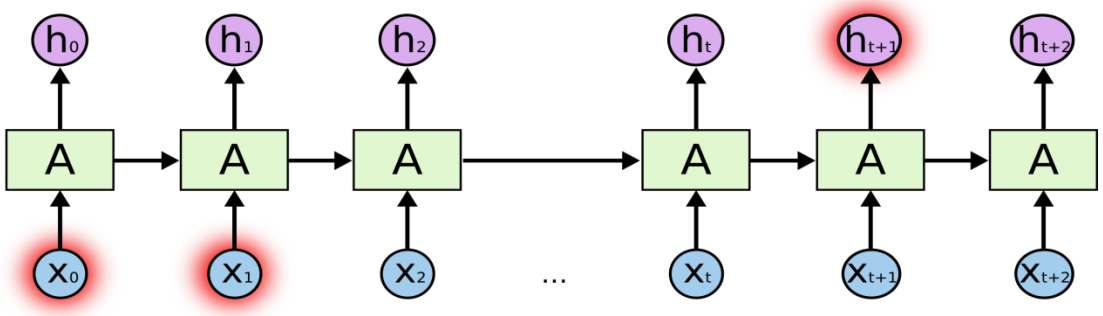


Figure: Chain-like nature of Recurrent Neural Network as illustrated in [1, Fig 2.3(a)]

But there are also cases where we need more contexts. Consider trying to predict the last word in the text "I grew up in France… I speak fluent French." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information. Yet Thankfully, Long Short Term Memory networks (LSTM) are a special kind of RNN, capable of learning long-term dependencies.

LSTM: In the late 90s, LSTM was proposed by Sepp Hochreiter and Jurgen Schmidhuber, which is relatively insensitive to gap length over alternatives RNNs, hidden Markov models, and other sequence learning methods in numerous applications [2]. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.

In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

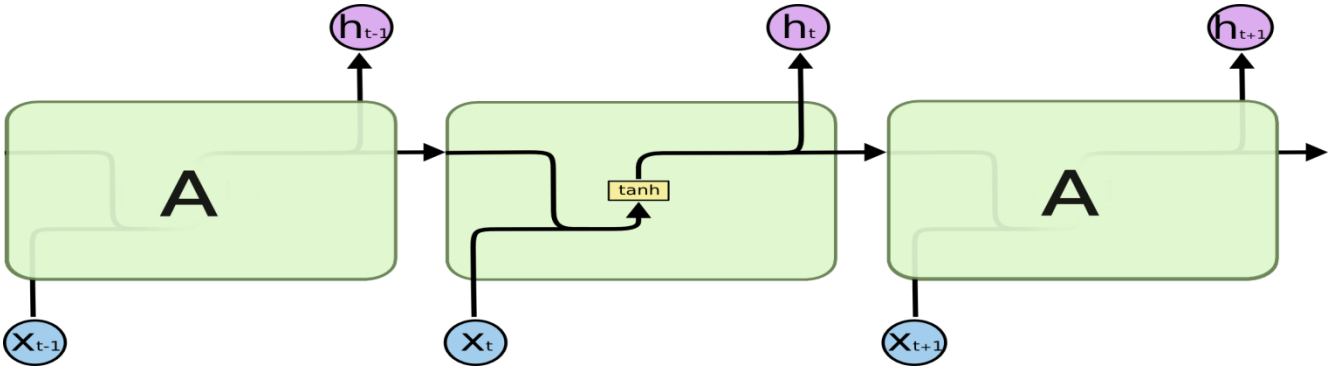


Figure: Repeating module in a standard RNN contains a single layer as illustrated in [2, Fig 2.2(b)]

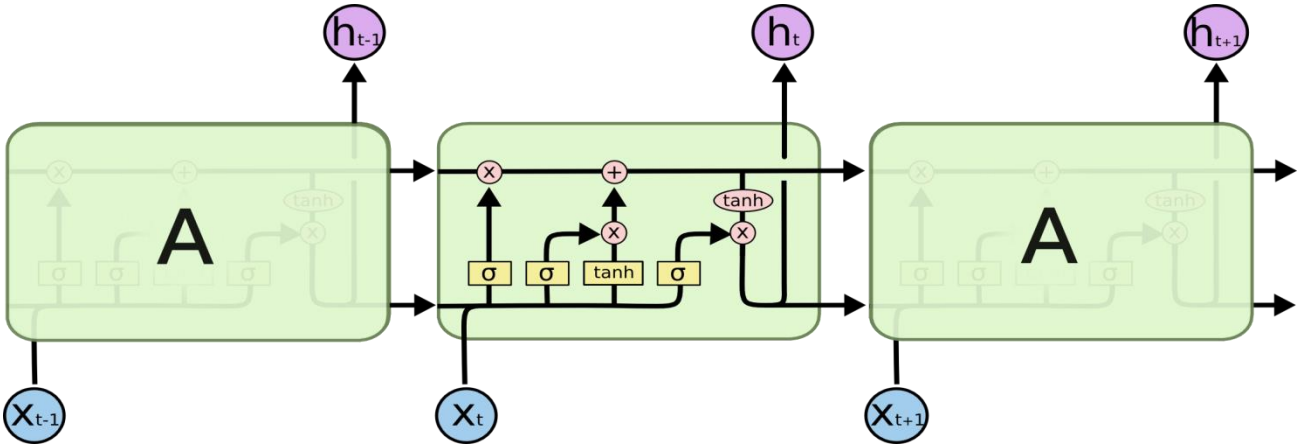


Figure: Repeating module in an LSTM contains four interacting layers as illustrated in [2, Fig 2.2(c)]

This model is organized in cells which include several operations. LSTM has an internal state variable, which is passed from one cell to another and modified by operation gates.

Forget Gate

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

This is a sigmoid layer that takes the output at t-1 and the current input at time t, concatenates them into a single tensor, and applies a linear transformation followed by a sigmoid. Because of the sigmoid, the output of this gate is between 0 and 1. This number is multiplied with the internal state and that's why the gate is called a forget gate. If ft=0, then the previous internal state is completely forgotten, while if ft=1, it will be passed through unaltered.

Input Gate

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

The input gate takes the previous output and the new input and passes them through another sigmoid layer. This gate returns a value between 0 and 1. The value of the input gate is multiplied with the output of the candidate layer.

$$C_t = \tanh\ (\ W_c.\ [h_{t-1},\ x_t] + b_c\ )$$

This layer applies a hyperbolic tangent to the mix of input and previous output, returning a candidate vector to be added to the internal state.

The internal state is updated with this rule:

$$C_t = f_t * C_{t-1} + i_t * C_t$$

The previous state is multiplied by the forget gate and then added to the fraction of the new candidate allowed by the output gate.

Output Gate

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

$$h_t = O_t * \tanh C_t$$

This gate controls how much of the internal state is passed to the output and it works in a similar way to the other gates.

The three gates described above have independent weights and biases, hence the network will learn how much of the past output to keep, how much of the current input to keep, and how much of the internal state to send out to the output.

2.4 Predictive Apriori Algorithm

Apriori algorithm is a classical algorithm in data mining. It is used for mining frequent item sets and relevant association rules. It is devised to operate on a database containing a lot of transactions, for instance, items brought by customers in a store.

Predictive Apriori Algorithm:

This algorithm searches with an increasing support threshold for the best 'n' rules concerning a support-based corrected confidence value. A rule is added if the expected predictive accuracy of the rule is among the 'n' best and it is not subsumed by a rule with at least the same expected predictive accuracy. This is also a confidence based association rule but in this rule ranked are sorted according to “predictive accuracy”. It tries to maximize predictive accuracy of an association rule rather than confidence in apriori.

Predictive Accuracy:

Predictive accuracy is generally used for the Predictive Apriori rule measurement. According to Scheffer , definition of predictive accuracy is as follows: Let D be a data file with n number of records. If [x → y] is an Association Rule which is generated by a static process P then the predictive accuracy of [x →y] is c([x → y])=P[n] satisfies y|n satisfies x]where distribution of r is given by the static process P and the Predictive Accuracy is the conditional probability of x→n and y→n.

CHAPTER 3: METHODOLOGY

3.1 Overview of the System:

In the large diabetes dataset, it is difficult to predict all insulin doses from data. We here predict only insulin dose code=33 for the prediction. Among 36 months data are trained here for predicting. We try to find which algorithm best for predicting the sequence of Diabetes Data.

Our works for predicting insulin are as follows:

1. Take our raw diabetes data

2. Preprocess the raw data and find the data which contains 33 codes.

3. Apply RNN LSTM for predicting next state sequence of insulin level.

4. Apply ANN on insulin level with accuracy.

5. Apply Predictive algorithm to find insulin level frequent item and Association rule with accuracy.

We first take raw diabetes dataset which contain 70000 data of various diabetes patients. We only take code=33 which contain insulin dose where we want to measure the next insulin level of long sequence data.

For these thinking we choose to any algorithm that predict 36 months breakfast to lunch or dinner prediction with a consume process time. Here we found many algorithms that predict the next observable sequence for fifteen or 20 days. But these algorithms don't predict 2000 days data at a time. We select RNN long short-term memory for solving these algorithms. For prove we use best prediction algorithm neural network. We compare two algorithms to know which one gives best prediction. A last we use java machine learning tool weka to find the best possible frequent insulin items set and mining some association rules with accuracy take 90 % accuracy. Before Apply predictive Apriori, we eliminate 50% zeros from dataset and convert each numerical data as a string. We will analysis RNN rmse and Ann rmse and what accuracy gives ANN for each dataset. We will analyze the epochs and training data for ANN and Train data score and test data score for RNN lstm.

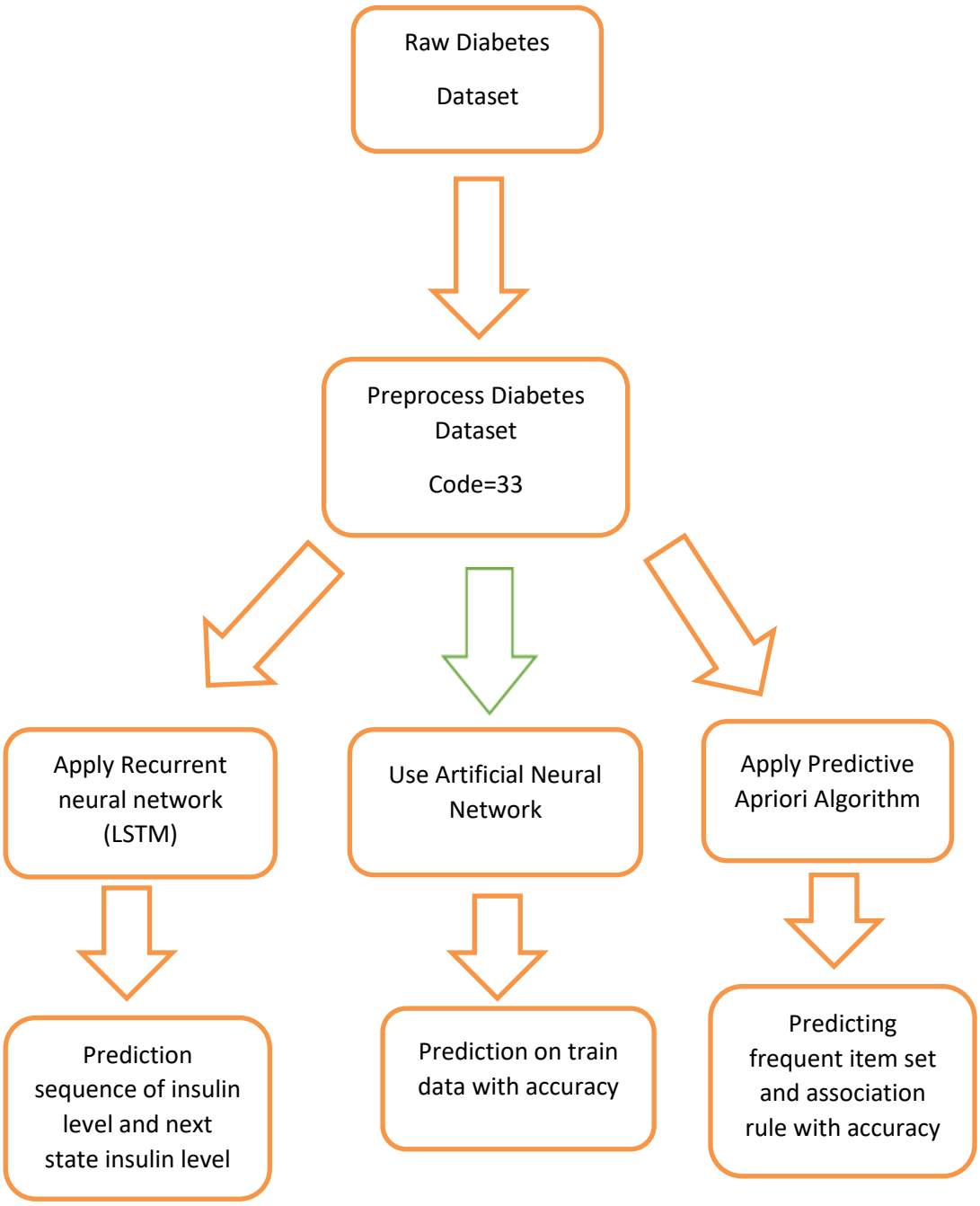


Figure 3.1: Structure of our Research Work

We first take raw diabetes dataset which contain 70000 data of various diabetes patients. We only take code=33 which contain insulin dose where we want to measure the next insulin level of long sequence data.

For these thinking we choose to any algorithm that predict 36 months breakfast to lunch or dinner prediction with a consume process time. Here we found many algorithms that predict the next observable sequence for fifteen or 20 days. But these algorithms don't predict 2000 days data at a time. We select RNN long short-term memory for solving these algorithms. For prove we use best prediction algorithm neural network. We compare two algorithms to know which one gives best prediction. A last we use java machine learning tool weka to find the best possible frequent insulin items set and mining some association rules with accuracy take 90 % accuracy. Before Apply predictive Apriori, we eliminate 50% zeros from dataset and convert each numerical data as a string. We will analysis RNN RMSE and Ann RMSE and what accuracy gives ANN for each dataset. We will analyze the epochs and training data for ANN and Train data score and test data score for RNN LSTM.

Chapter 4: Implementation

4.1 Data Set Information:

The dataset is used in our research was taken from https://archive.ics.uci.edu/ml/datasets/diabetes called UCI repository. Diabetics patient records can be obtained from two sources: an automatic electronic recording device and paper records. The automatic device has an internal clock to timestamp events, whereas paper records provide
"logical time" slots (breakfast, lunch, dinner, bedtime).Diabetic files consist of four fields per record.(1) Date in MM-DD-YYYY format (2) Time in XX:YY format (3) Code (4) Value.

The code field is deciphered as follows: 33=Regular Insulin dose 54=NPH insulin dose, 35=Ultralente insulin dose, 48=Unspecified blood glucose measurement, 57=unspecified blood glucose measurement etc.

We have taken 36 month’s data (insulin chart) for code 33i.eReguler insulin dose of a patient. Assuming that this huge data works as training data, next one month chart is predicted and then it will be compared with the actual data.

Original Data: Preprocessed Data: Code=33

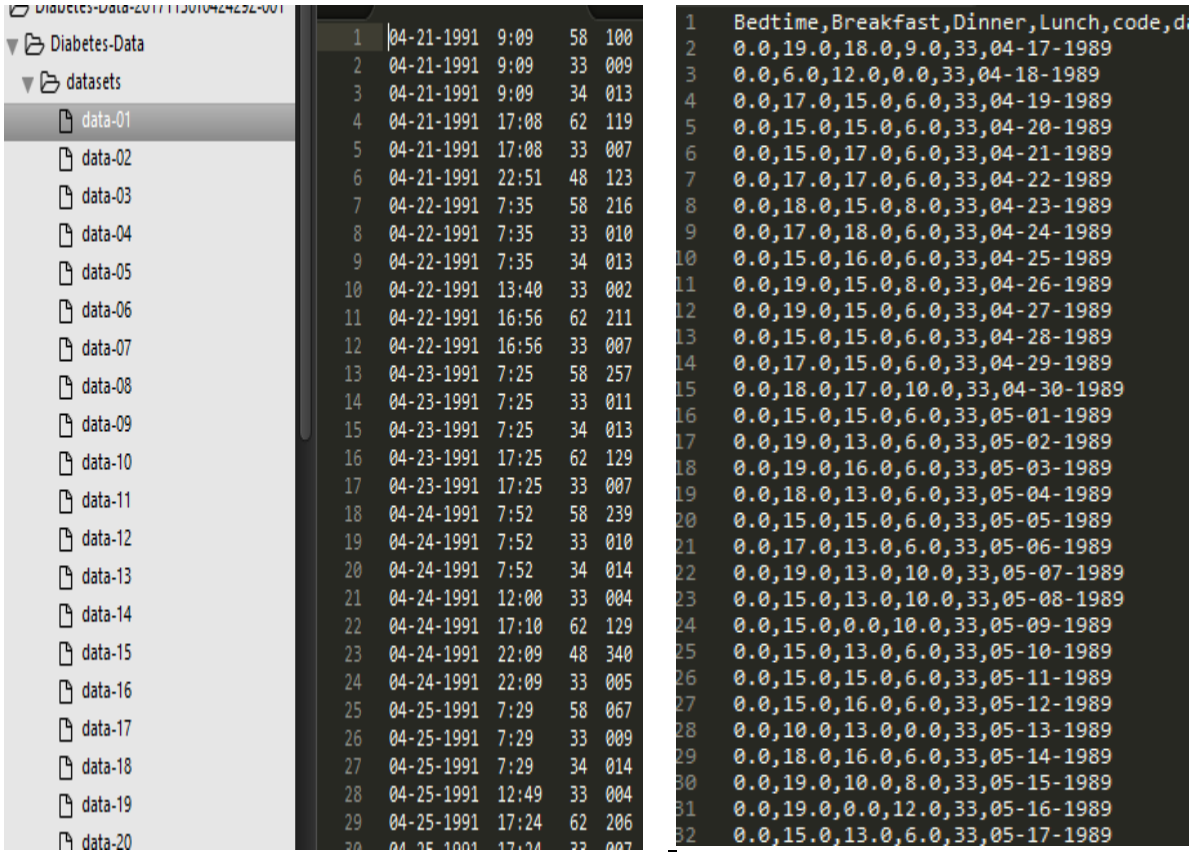


Fig: 4.1.1 Original dataFig: 4.1.2 Preprocessed Data

Implementation code:

1. First we first take all in a individual file. Where we take all time to convert into each time as breakfast, dinner, lunch, bedtime.

2. Then calling function take 1 file, read it and take code 33 from that. Then extended data frame to store file what number of file read. After extended which data file, it reads added with the current file .Then stores it in data frame.

```
def get_time_table(time):
    time = time.replace(":", "")
    time = int(time)
    if time < 1200:


19
return "Breakfast"
elif 1200 <= time < 1600:
        return "Lunch"
elif 1600 <= time < 2000:
        return "Dinner"
elif 2000 <= time < 2400:
```

```
        return "Bedtime"


def get_presented_data(temp_data, _code):
data_list = []
    for row in temp_data.iterrows():
        if row[1][2] == _code:
            single = {"date": row[1][0], "time":
get_time_table(row[1][1]), "code": row[1][2], "level":
row[1][3]}
data_list.append(single)

temp_all_data = [
        {'date': data_list[0]['date'], 'code':
str(data_list[0]['code']), 'Breakfast': 0, 'Lunch': 0,
'Dinner': 0, 'Bedtime': 0}]
i = 0
    for data in data_list:
        if temp_all_data[i]['date'] != data['date']:
temp_all_data.append(
                {'date': data['date'], 'code':
str(data['code']), 'Breakfast': 0, 'Lunch': 0, 'Dinner': 0,
'Bedtime': 0})
i += 1
temp_all_data[i][data['time']] = data['level']
    return temp_all_data
```

4.2 Implementation diagram:

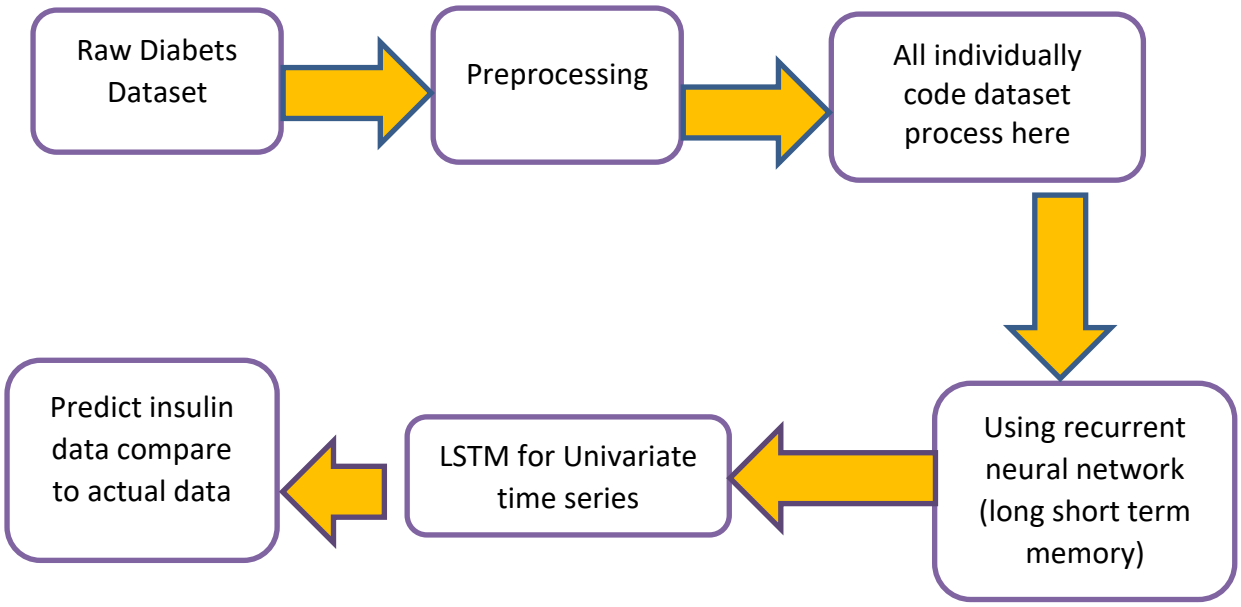


Figure 4.2: Implementation Diagram of our Research Work

4.3 Recurrent Neural Network Long Short Term Memory

Procedure:

1. Load dataset from csv file.

2. Fit random seed for predictability.

3. Normalize dataset.

4. Split into train and test data.

5. Reshape input to be samples, time steps and features.

6. Create and fit LSTM neural network.

7. Make prediction.

8. Invert prediction.

9. Calculate root mean square.

10. Shift train and test data for plotting.

Description on procedural:

1. We can write a simple function to convert our single column of data into a two-column dataset: the first column containing 36 month's (t) days insulin column for breakfast or lunch or dinner and the second column containing next 36month's (t+1) insulin, to be predicted.

   Before getting first import all of function. This assumes a working scipy environment with keras deep learning installed.

```
import numpy

import matplotlib.pyplot as plt

from pandas import read_csv

import math

import keras

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error
```

2. Fix random number seed to ensure our results are reproducible.

fix random seed for reproducibility

numpy.random.seed(7)

3. We can also use the code from the previous section to load the dataset as a Pandas data frame. We can then extract the NumPy array from the data frame and convert the integer values to floating point values, which are more suitable for modeling with a neural network.

```
# load the dataset

dataframe = read_csv('person.csv', usecols=[2],
engine='python', skipfooter=3)

dataset = dataframe.values
```

4. LSTMs are sensitive to the scale of the input data, specifically when the sigmoid (default) or tanh activation functions are used. It can be a good practice to rescale the data to the range of 0-to-1, also called normalizing. We can easily normalize the dataset using the **MinMaxScaler** preprocessing class from the scikit-learn library.

normalize the dataset

```
scaler = MinMaxScaler(feature_range=(0, 1))

dataset = scaler.fit_transform(dataset)
```

5. With time series insulin data, the sequence of values is important. A simple method that we can use is to split the ordered dataset into train and test datasets. The code below calculates the index of the split point and separates the data into the training datasets with 67% of the observations that we can use to train our model, leaving the remaining 33% for testing the model.

split into train and test sets

```
train_size = int(len(dataset) * 0.67)

test_size = len(dataset) - train_size

train, test = dataset[0:train_size,:],
dataset[train_size:len(dataset),:]
```

6. The function takes two arguments: the **dataset**, which is a NumPy array that we want to convert into a dataset, and the **look_back**, which is the number of previous time steps to use as input variables to predict the next time period — in this case defaulted to 1.

This default will create a dataset where X is the total row of insulin in breakfast or lunch or dinner or breakfast at a given time (t) and Y is insulin level for breakfast or dinner or lunch or bedtime at the next time (t + 1).

```
# converts an array of values into a dataset matrix

def create_dataset(dataset, look_back=1):

     dataX, dataY = [], []

     for i in range(len(dataset)-look_back-1):

          a = dataset[i:(i+look_back), 0]

          dataX.append(a)

          dataY.append(dataset[i + look_back, 0])

     return numpy.array(dataX), numpy.array(dataY)
```

7. The LSTM network expects the input data (X) to be provided with a specific array structure in the form of: *[samples, time steps, features]*.

Currently, our data is in the form: [*samples, features*] and we are framing the problem as one time step for each sample. We can transform the prepared train and test input data into the expected structure using **numpy.reshape()** as follows:

```
# reshape into X=t and Y=t+1

look_back = 1

trainX, trainY = create_dataset(train, look_back)

testX, testY = create_dataset(test, look_back)
```

8. We are now ready to design and fit our LSTM network for this problem.

The network has a visible layer with 1 input, a hidden layer with 4 LSTM blocks or neurons, and an output layer that makes a single value prediction. The default sigmoid activation function is used for the LSTM blocks. The network is trained for 100 epochs and a batch size of 1 is used.

```
# create and fit the LSTM network

model = Sequential()
```

```
model.add(LSTM(4, input_shape=(1, look_back)))

model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')


model.fit(trainX, trainY, epochs=10, shuffle=True,
batch_size=1, verbose=2)
```

9. Prediction:

Once the model is fit, we can estimate the performance of the model on the train and test datasets. This will give us a point of comparison for new models.

Note that we invert the predictions before calculating error scores to ensure that performance is reported in the same units as the original data

```
# make predictions

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

# invert predictions

trainPredict = scaler.inverse_transform(trainPredict)

trainY = scaler.inverse_transform([trainY])

testPredict = scaler.inverse_transform(testPredict)

testY = scaler.inverse_transform([testY])

# calculate root mean squared error

trainScore = math.sqrt(mean_squared_error(trainY[0],
trainPredict[:,0]))

print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(mean_squared_error(testY[0],
testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))
```

10. Finally, we can generate predictions using the model for both the train and test dataset to get a visual indication of the skill of the model.

Because of how the dataset was prepared, we must shift the predictions so that they align on the x-axis with the original dataset. Once prepared, the data is plotted, showing the original dataset in blue, the predictions for the training dataset in green, and the predictions on the unseen test dataset in red.

```
# shift train predictions for plotting

trainPredictPlot = numpy.empty_like(dataset)

trainPredictPlot[:, :] = numpy.nan

trainPredictPlot[look_back:len(trainPredict)+look_back, :] =
trainPredict

# shift test predictions for plotting

testPredictPlot = numpy.empty_like(dataset)

testPredictPlot[:, :] = numpy.nan

testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)
-1, :] = testPredict

# plot baseline and predictions

plt.plot(scaler.inverse_transform(dataset))

plt.plot(trainPredictPlot, label='train')

plt.plot(testPredictPlot, label='test')

plt.legend()

plt.show()
```

4.4 Artificial neural Network

4.4.1 Procedure

1. Load data

2. Define model

3. Compile model

4. Fit model

5. Evaluate and prediction we implement artificial neural network,Scipy(including Numpy)installed and we use keras and a backend (theano and tensorflow)installed.

Procedure description:

1. Whenever we work with machine learning algorithms that use a stochastic process (e.g. random numbers), it is a good idea to set the random number seed.

This is so that you can run the same code again and again and get the same result. This is useful if you need to demonstrate a result, compare algorithms using the same source of randomness or to debug a part of your code.

As such, it is a binary classification problem (onset of diabetes as 1 or not as 0). All of the input variables that describe each patient are numerical. This makes it easy to use directly with neural networks that expect numerical input and output values, and ideal for our first neural network in Keras.

We now load the file directly using the NumPy function **loadtxt()**. There are 3 input variables and one output variable (the last column). Once loaded we can split the dataset into input variables (X) and the output class variable (Y).

```
training_data_df = pd.read_csv("0-64.csv")

del training_data_df['date']

del training_data_df['code']


training_data = training_data_df

train, test = train_test_split(training_data, test_size=0.1)

X = train.drop('Lunch', axis=1).values

Y = train[['Lunch']].values


X_test = test.drop('Lunch', axis=1).values

Y_test = test[['Lunch']].values
```

2. Models in Keras are defined as a sequence of layers.

We create a Sequential model and add layers one at a time until we are happy with our network topology.

The first thing to get right is to ensure the input layer has the right number of inputs. This can be specified when creating the first layer with the **input_dim** argument and setting it to 3 for the 50 input variables.

we will use a fully-connected network structure with three layers.

Fully connected layers are defined using the Dense class. We can specify the number of neurons in the layer as the first argument, the initialization method as the second argument as **init** and specify the activation function using the **activation** argument.

We will use the rectifier ('**relu**') activation function on the first three layers and the sigmoid function in the output layer. It used to be the case that sigmoid and tanh activation functions were preferred for all layers. These days, better performance is achieved using the rectifier activation function. We use a sigmoid on the output layer to ensure our network output is between 0 and 1 and easy to map to either a probability of class 1 or snap to a hard classification of either class with a default threshold of 0.5.

We can piece it all together by adding each layer. The first layer has 50 neurons and expects 3 input variables. The second hidden layer has 100 neurons and finally, the output layer has 1 neuron to predict the class (onset of diabetes or not).

```
# Define the model

model = Sequential()

model.add(Dense(50, input_dim=3, activation='relu'))

model.add(Dense(100, activation='relu'))

model.add(Dense(50, activation='relu'))

model.add(Dense(1, activation='linear'))

model.compile(loss='mean_squared_error', optimizer='adam',
metrics=['acc'])
```

3. Now that the model is defined, we can compile it.

Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware, such as CPU or GPU or even distributed.

When compiling, we must specify some additional properties required when training the network. Remember training a network means finding the best set of weights to make predictions for this problem.

We must specify the loss function to use to evaluate a set of weights, the optimizer used to search through different weights for the network and any optional metrics we would like to collect and report during training.

In this case, we will use logarithmic loss, which for a binary classification problem is defined in Keras as "**mean_squared_error**". We will also use the efficient gradient descent algorithm "**adam**" for no other reason that it is an efficient default and metrics="acc" to find accuracy

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['acc'])

4. We have defined our model and compiled it ready for efficient computation.

Now it is time to execute the model on some data.

We can train or fit our model on our loaded data by calling the **fit()** function on the model.

The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the **nepochs** argument. We can also set the number of instances that are evaluated before a weight update in the network is performed, called the batch size and set using the **batch_size** argument.

For this problem, we will run for a small number of iterations (len(train_data)). Again, these can be chosen experimentally by trial and error.

```
# Train the model

model.fit(

    X,

    Y,

    epochs=len(training_data),

    shuffle=True

)
```

5. We have trained our neural network on the entire dataset and we can evaluate the performance of the network on the same dataset.

This will only give us an idea of how well we have modeled the dataset (e.g. train accuracy), but no idea of how well the algorithm might perform on new data. We have done this for simplicity, but ideally, you could separate your data into train and test datasets for training and evaluation of your model.

You can evaluate your model on your training dataset using the **evaluate()** function on your model and pass it the same input and output used to train the model.

This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics you have configured, such as accuracy.

```
test_error_rate = model.evaluate(X_test, Y_test, verbose=0)

print("The mean squared error (MSE) for the test data set is:
{}".format(test_error_rate))
```

4.5 Predictive Apriori Algorithm implementation with weka:

Implementation 1.We take string data For each column of bedtime, breakfast, dinner, lunch. Due to some numerical problem of row wise prediction we convert the dataset with like this Dinner=2,we write in dataset Dinner=”twoD”.Because if Apriori does not find any string they cannot find item set or association rule.

Implementation 2:

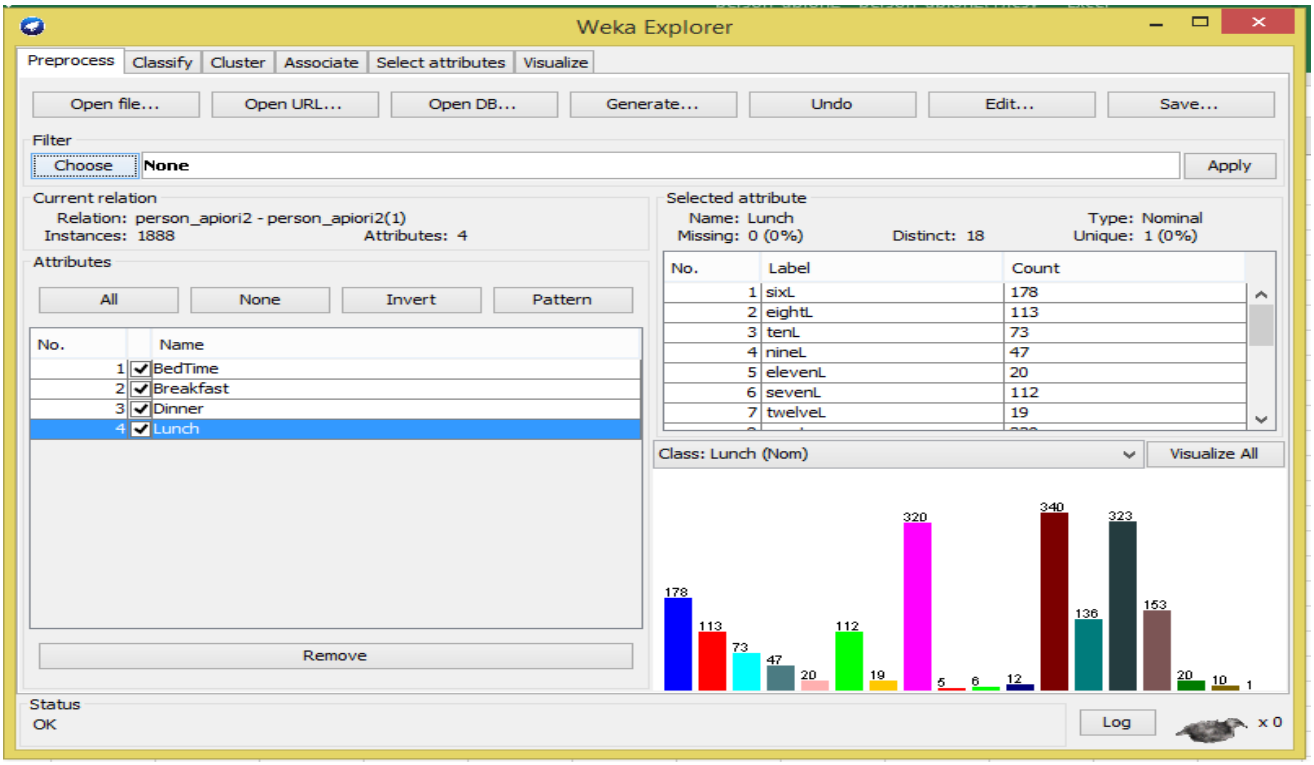
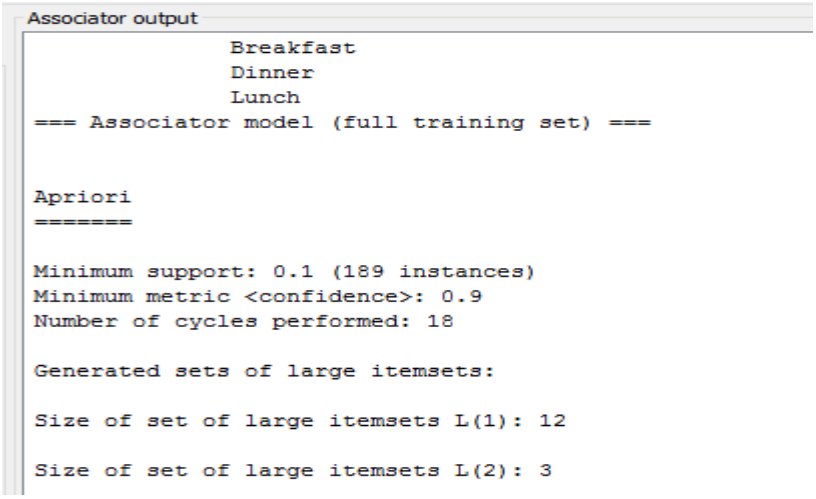


Fig: 4.5.1 Weka Implementation

Implementation 3:



Implementation 4:

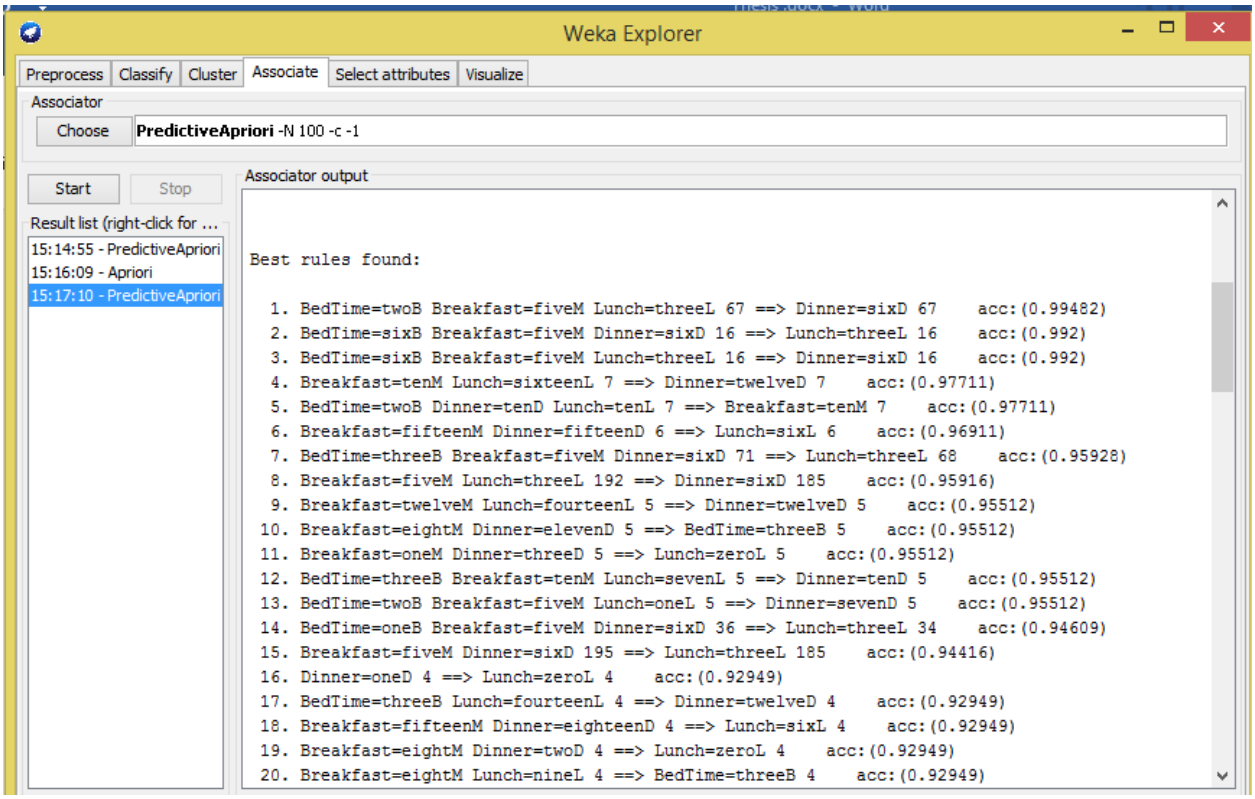


Fig: 4.5.2 Predictive Apriori

Chapter 5: Result and Analysis

5.1 Recurrent neural network LSTM prediction

After preprocessing data, we implement RNN LSTM for predicting next state sequence of insulin data. RNN takes data as a sequence .For this it is easy to predict for RNN to predict any sequence of data. We have a large 70 dataset of text file where each contain 1000 data. After preprocessing, we got only 130 data each file of code =33,which contains only insulin regular dose. We have four states of breakfast,lunch,dinner,bedtime.We here predict each of each state's insulin value by RNN.We measure 100 epochs for iteration to this algorithm. The performance of each prediction gives RNN a good test score RMSE or a bad test score RMSE or an average test RMSE.

5.1.1 for breakfast prediction

We train column 2 on the dataset from (0-14) dataset.Almost 655 data found after preprocessing. We train breakfast 67% and test 33%. Rnn predicts from columns 2.On the Figure 5.1.1.1,RNN lstm predicts well. After 100 epochs, loss will be decreasing. If epochs will increase, the prediction of 655 data will be quite accurate.

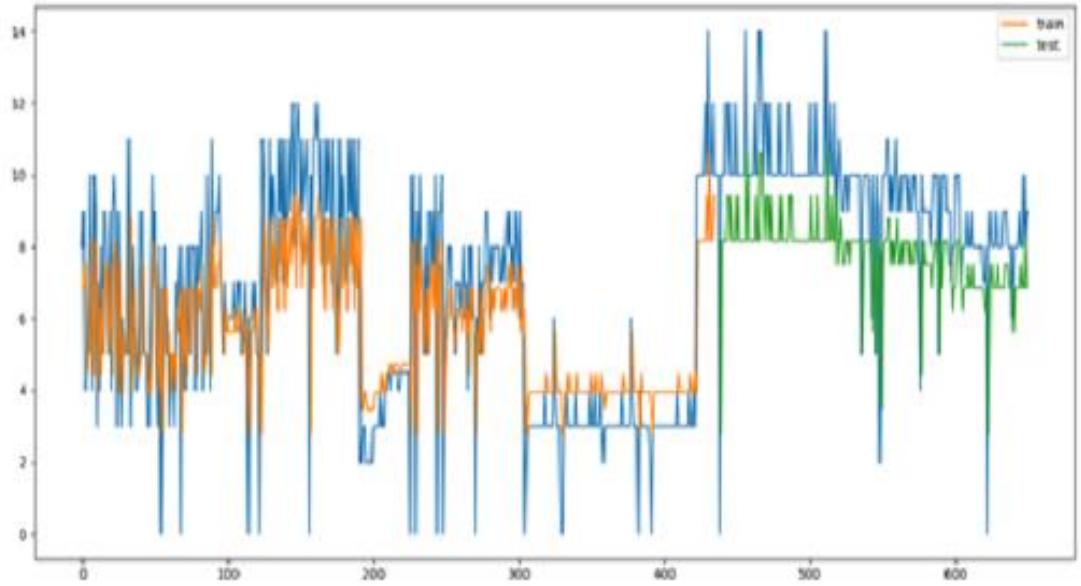


Fig: 5.1.1.1 Breakfast Prediction (0-14)

In Figure 5.1.1.1, RNN just predicts sequentially. For improving prediction need more epochs and data, it can predict very well. In this Figure, RNN seems done very good job.

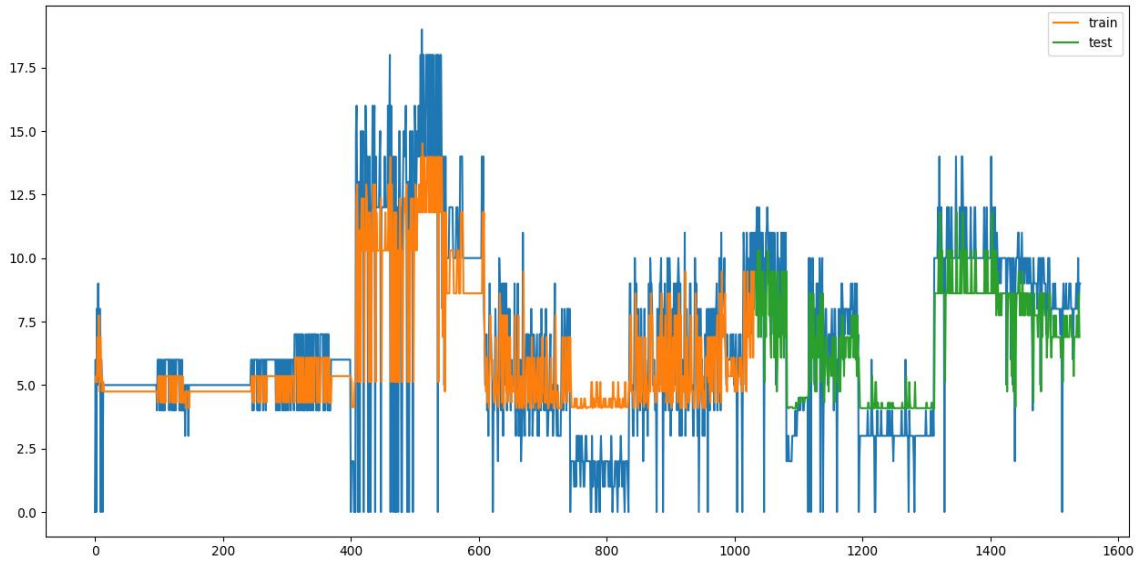


Fig: 5.1.1.2 Breakfast Prediction (0-31)

On the Figure: 5.1.1.2, dataset are better from previous .We take (0-31) dataset which contains 1595 data. Data prediction seems quite good .Blue color on the graph actually predict data, orange and green color graph actually train and test data.RNN improves for predicting in this graph. In this graph, we need too much epochs for better plot. But system will be slow. We keep it 100 because 1000 epochs need too much time.

Fig: 5.1.1.3 Breakfast Prediction (0-46)

On the Figure: 5.1.1.3 dataset, from (0-46) preprocessed data 2050,RNN evaluates this data performance better from previous 2 plots.RNN takes too many sequences once a time. When it takes more data, it gives prediction very well. In previous data plot, data will not fit properly due to less of data.For this RNN prediction gives graph plot too many wrong predictions.But when it gets this type of data,prediction of next state sequence will be better.On the 2050 data,graph fits very well and RNN predicts this data very well.

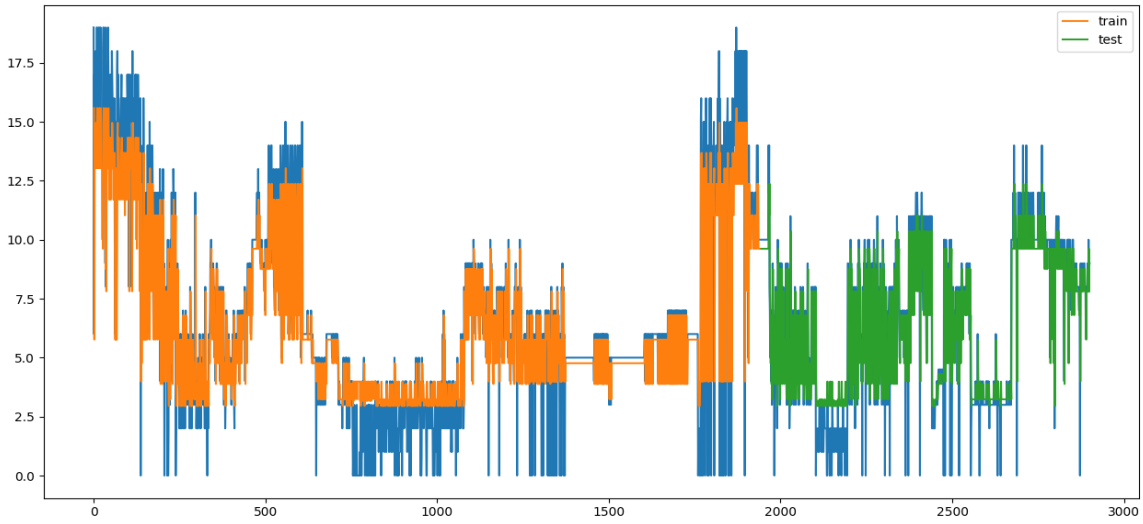


Fig: 5.1.1.4 Breakfast Prediction(0-46)

In Figure: 5.1.1.4,predictions are slightly better than previous plot.When it gets too much data,predicts is absolutely brilliant.Most of time predictions are slightly better.What is the benefit of RNN is, don't need to calculation each of the state.It is the biggest advantage of RNN.It predicts whatever gives sequence.This will be better if we give epochs 1000.RNN improves prediction in every state.

5.1.2 For Lunch prediction

In lunch, Insulin level was not too high. During this time, prediction RNN just predicts next state sequence.RNN takes sequence to its three gates where forget, input and output gates give the prediction of sequence.

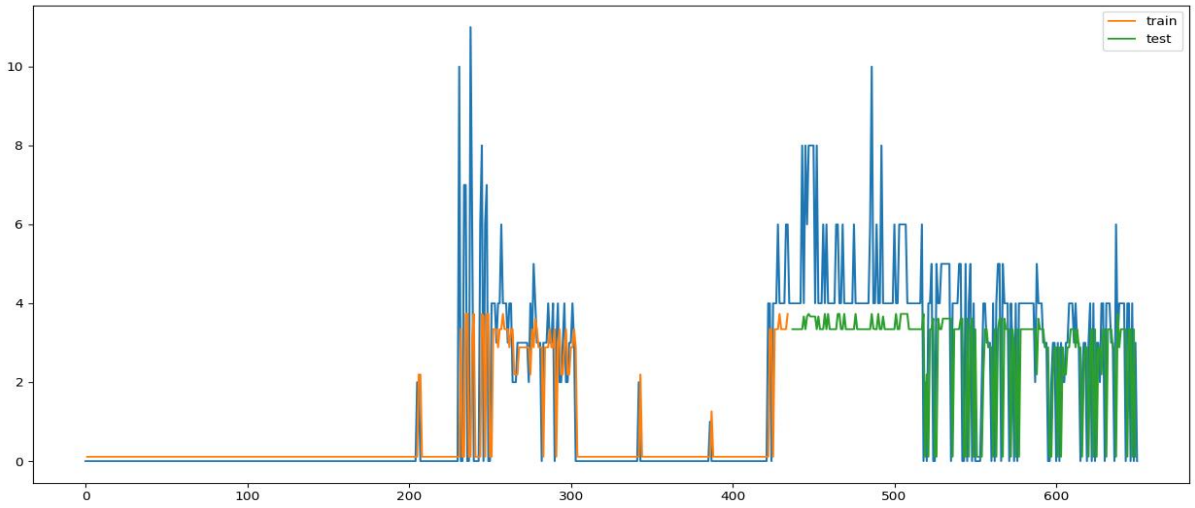


Fig: 5.1.2.1 lunch Prediction (0-14)

Here in Figure: 5.1.2.2.1, predicted data quietly under fit. We predict from dataset (0-14) 655 preprocessed data in 655 days. Each day of data are different from previous data.RNN needs this sequence where it has the ability to predict from its own gates.From the differences of other algorithm,itdoesn’t need to calculate each of these states.It predicts a sequence of 655 days data with a 100 epochs.

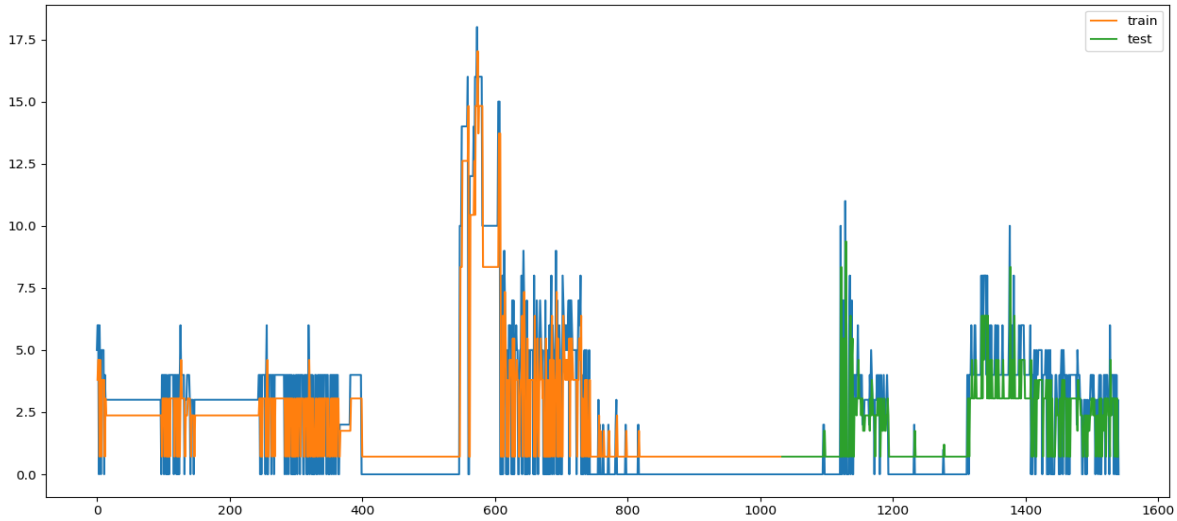


Fig: 5.1.2.2 lunch Prediction (0-31)

In Figure: 5.1.2.2, the insulin level of this graph is very low.We know insulin level of human body can take (2-11) .RNN predicts this graph overfitting. Predicted value does not differ too much from actual data.RNN does a good work after each day predicting. Dataset from (0-31) about 1595 data,RNNgets more data for predicting than previous plot.This plot seems very good prediction than the previous.

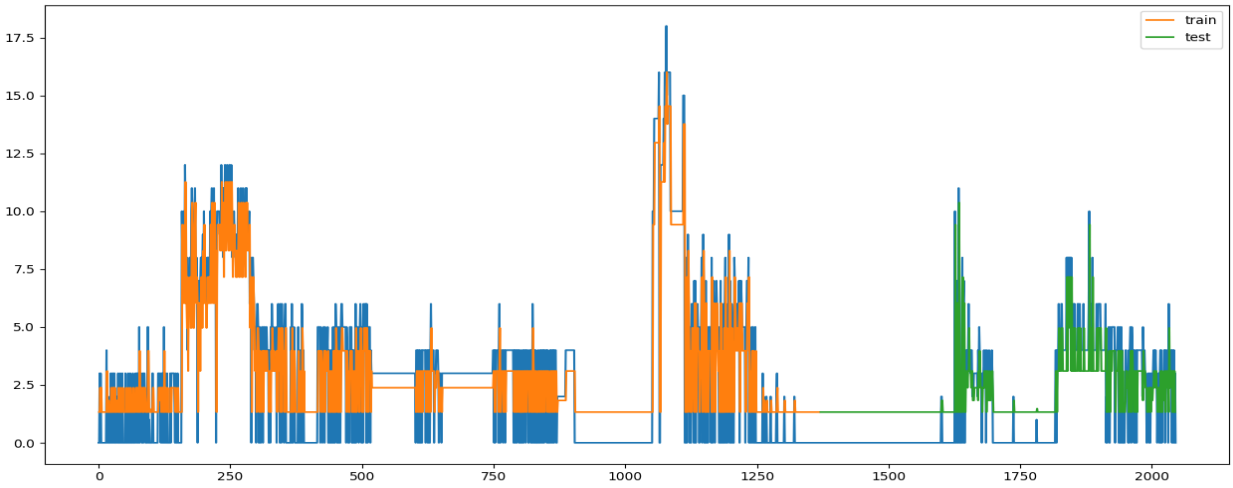


Fig: 5.1.2.3 lunch Prediction (0-46)

In Figure:5.1.2.3,Dataset from(0-46) contains data 2050,prediction on this data is improving .Figure:5.1.2.3 above seems very good after predicting than previous two plot.RNN gives this opportunity to every sequenceprediction,if we give too much data and epochs ,then the loss functions are decreasing ,then we find a very good plot.

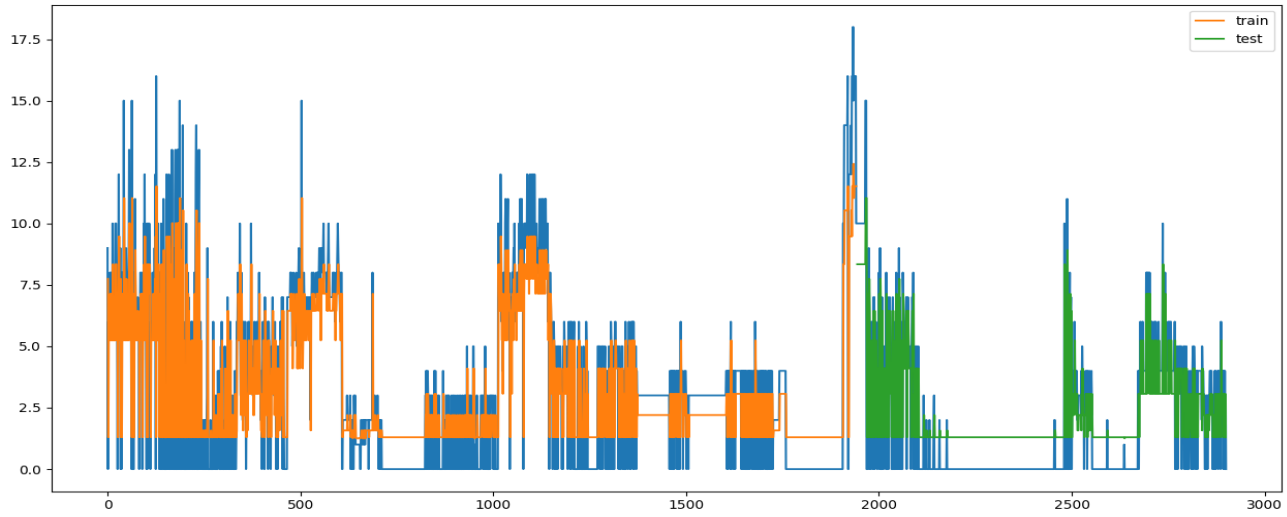


Fig: 5.1.2.4 lunch Prediction (0-64)

RNN predicts this well than previous three plots.Here dataset from (0-64) and 2095 data.RNNimpressively predict this large 2095 day data.Predicted data are plotted on the similar to actual data.Moreover, we have got 90% accuracy in this graph.Loss functions are decreasing here.RNNlstm proves that observable sequence of data prediction it is master of technique.

5.1.3 for Dinner Prediction

In Dinner time insulin level is high for each person.

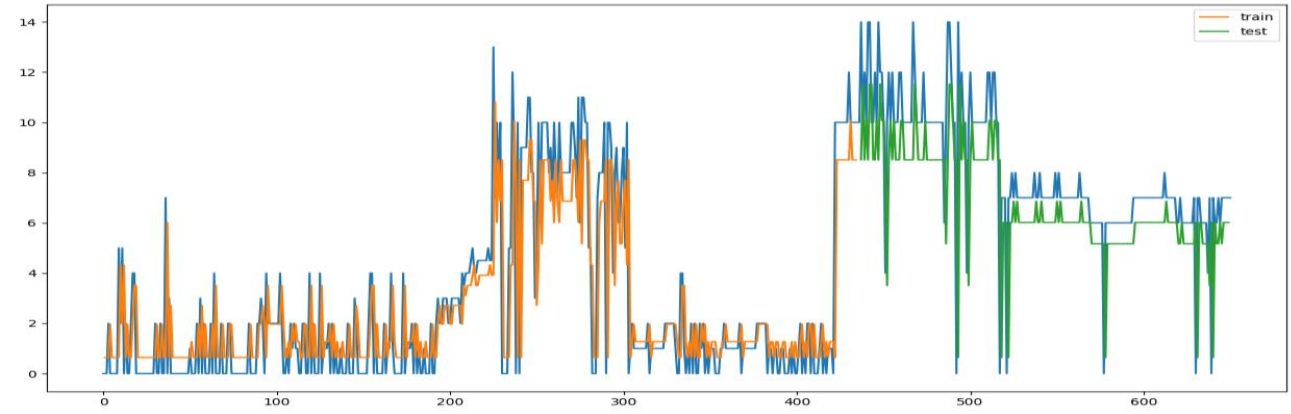


Fig: 5.1.3.1 Dinner Prediction (0-14)

In Figure: 5.1.3.1 predicted data looks so good .The improvement of prediction look RNN very strong.For 655 preprocessed data RNN easily canpredict.The accuracy of Figure: 5.1.3.1 very well.The number of epoch will increase, we will improve plotted data will be improved gradually.

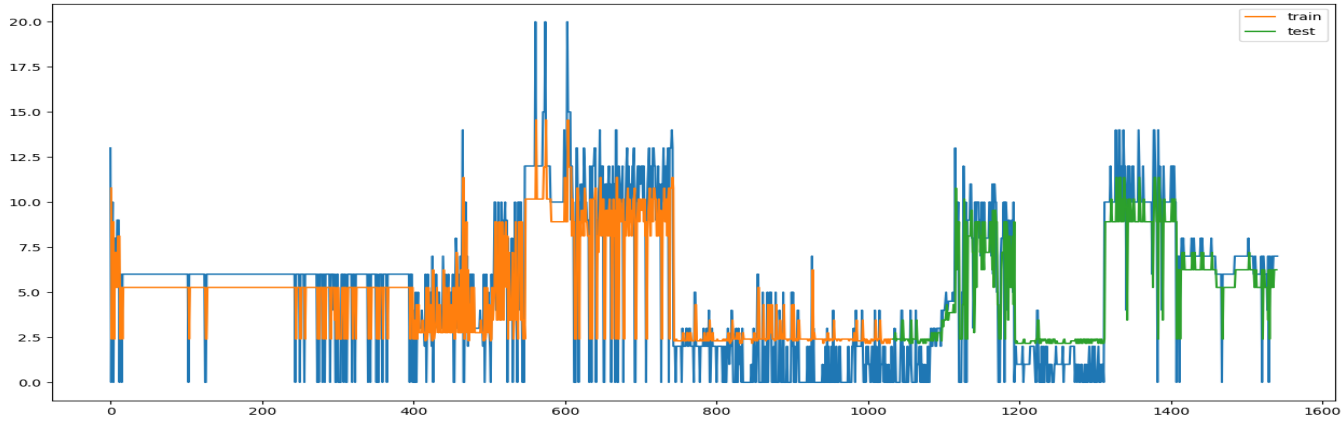


Fig: 5.1.3.2 Dinner Prediction (0-31)

In the Figure: 5.1.3.2 dataset from (0-31) where 1545 data is predicted on the actual data.RNNlstm predicted well.It seems data was fitting very well on the actual data.The whole graph predicted insulin level with the time series and epochs. If we improve epochs with RNN,graph will gradually be a better shape.

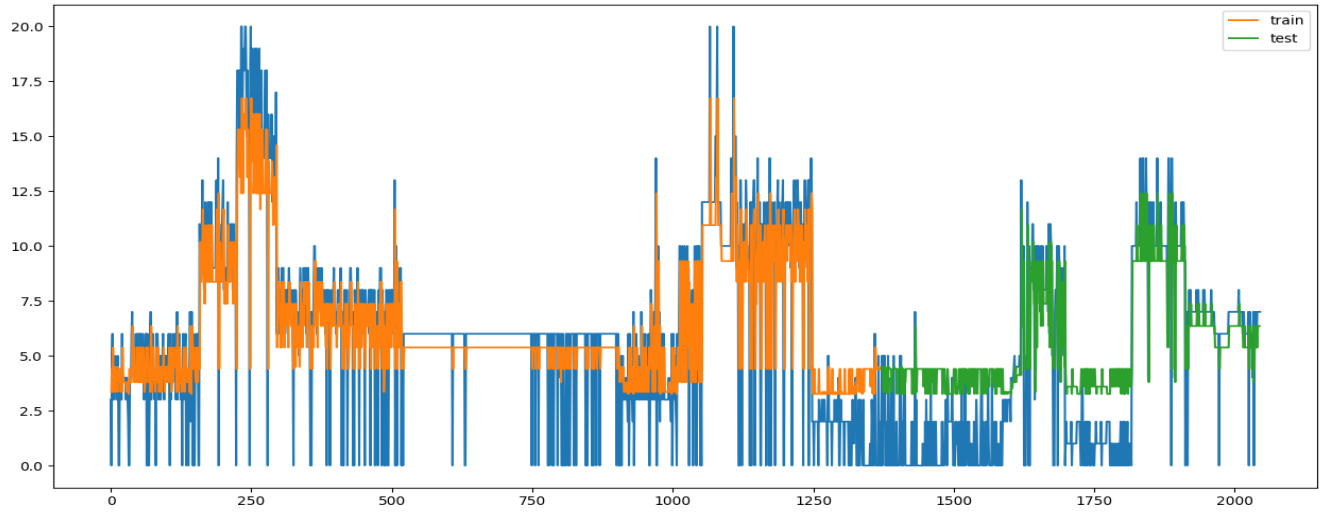


Fig: 5.1.3.3 Dinner Prediction (0-46)

In the figure: 5.1.3.3 dataset from (0-46),2050 preprocessed data predicted on the actual data. Predicted data are quite impressive on the actual data. Data are very fit on the actual data. RNN LSTM predicts these previous two plots well.

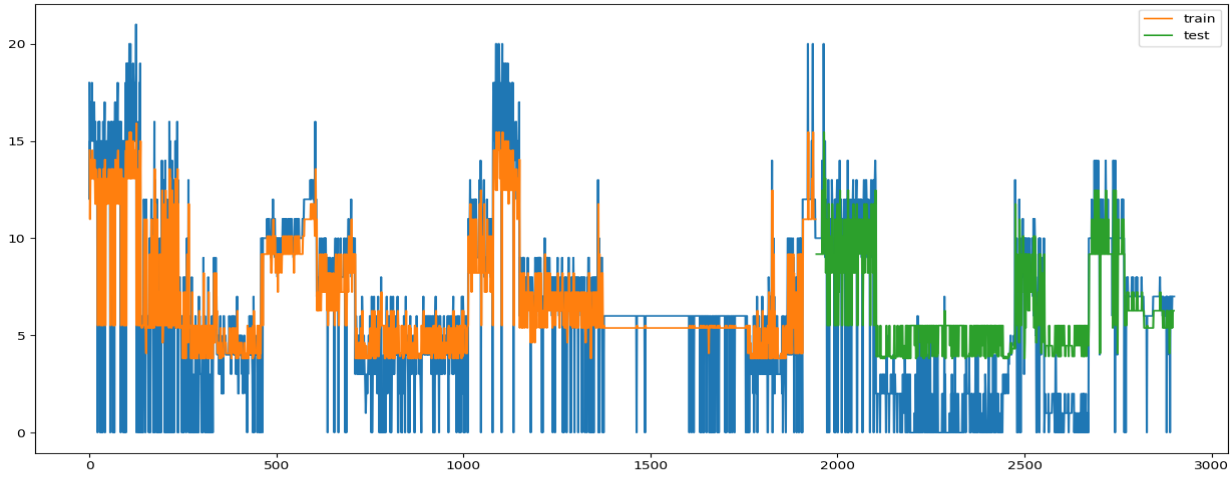


Fig: 5.1.3.4 Dinner Prediction (0-46)

In the figure: 5.1.3.4we predicted 2905 data on the actual value. Prediction looks good to this dataset. This prediction goes better because we take more data from previous three plots. When we take too many data, loss will be decreasing and predicted data will fit correctly fit on actual data.

5.1.4 for Bedtime Prediction

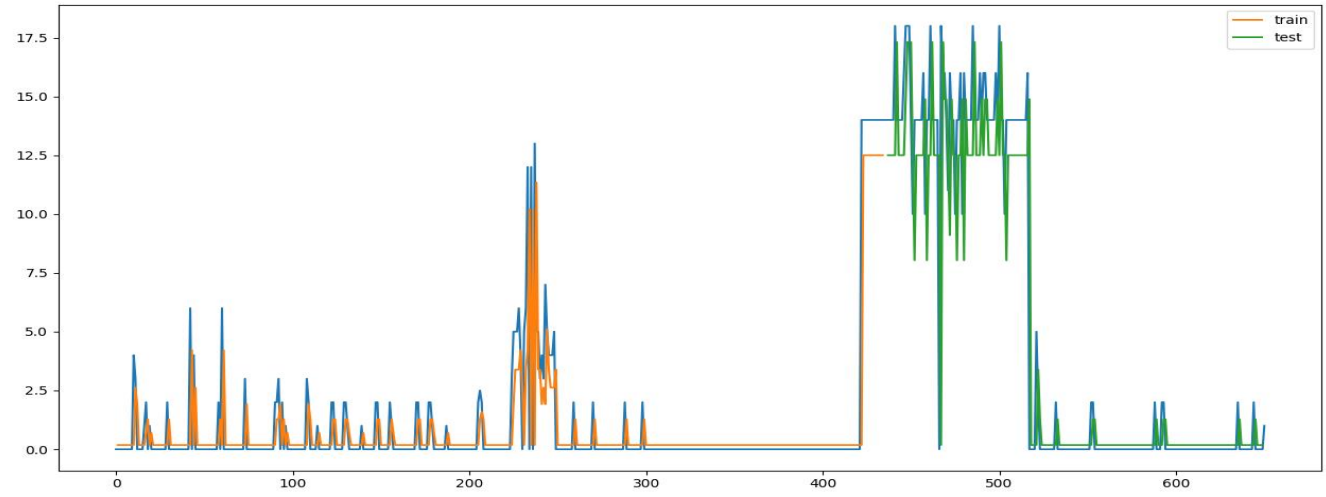


Fig: 5.1.4.1 Bedtime Prediction (0-14)

In the figure:5.1.4.1, most of time bedtime almost zero in the dataset.RNN try to predict for this low level insulin and predict almost looks good for this graph.

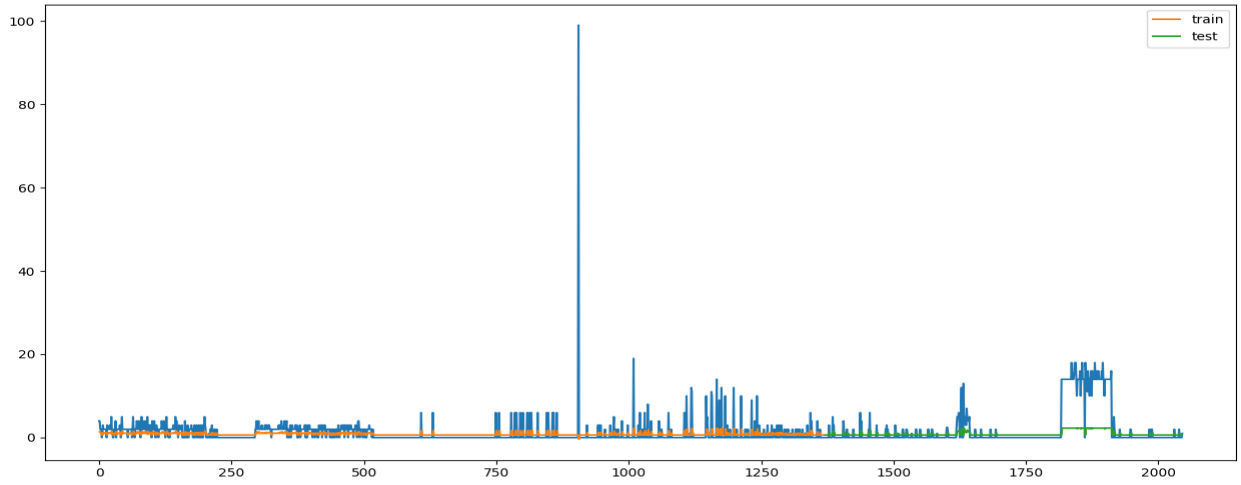


Fig: 5.1.4.2 Bedtime Prediction (0-31)

In the figure: 5.1.4.2 we take minimum 1545 data to predict bedtime sequence of next state prediction.Hence bedtime times show almost zero,so almost predict through these difficulty very well.

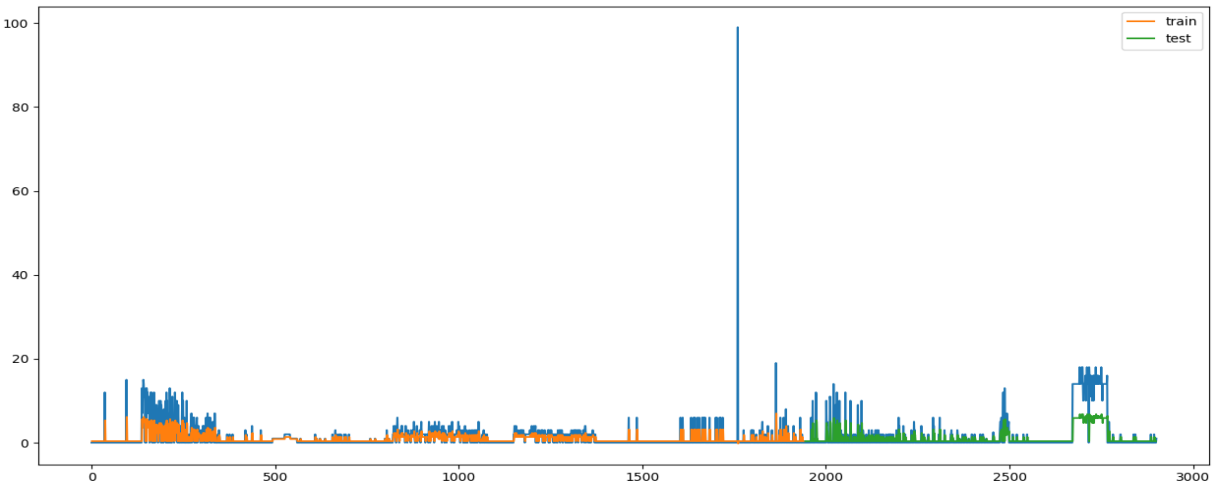


Fig: 5.1.4.3 Bedtime Prediction (0-46)

In the figure: 5.1.4.3 almost 2000 data was trained for predicting in bed time. Hence most of data is zero, graph does not show much, although it will predict well on dataset.

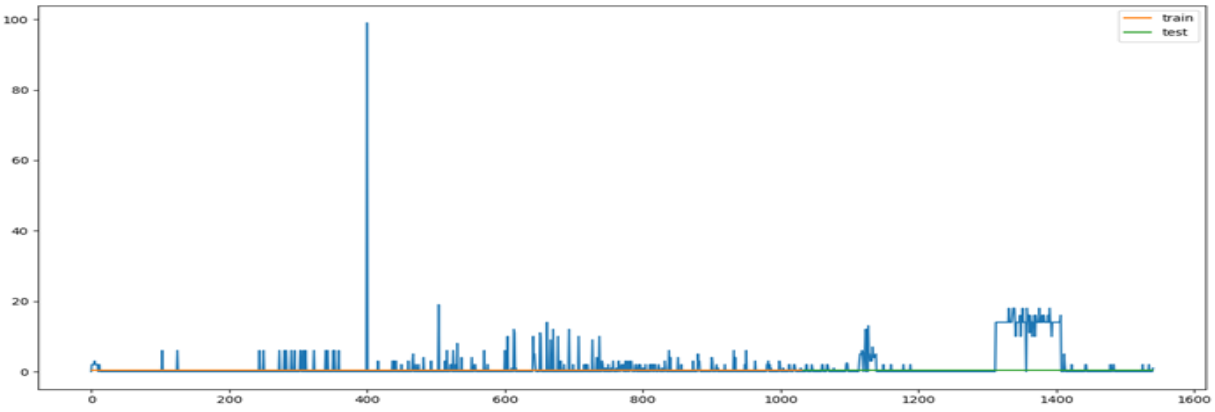


Fig:5.1.4.4Bedtime Prediction (0-64)

We take almost all preprocessed data, predict level was not too good because bedtime insulin level almost goes zero to this time.

5.2 Prediction train Score and test Score (RMSE)

Table 5.2.1:Breakfast Prediction

DataSet	Data(preprocessed)	Train Score(RMSE)	Test Score(RMSE)
0-14	655	2.38	2.37
0-31	1545	2.88	2.18
0-46	2050	2.64	2.28
0-64	2905	2.79	2.28

In the Table5.2.1,we divided our dataset 4 times and takes train score and test score upon the prediction of RNN lstm on the insulin level. When we take (0-14) 655 data, we see that RNN predicts that average insulin level prediction on train data is 2.38 and test data 2.37 where we see train data is greater than the test data .So we conclude that test data are underfitting on train data score. So RNN predicts insulin level very well.

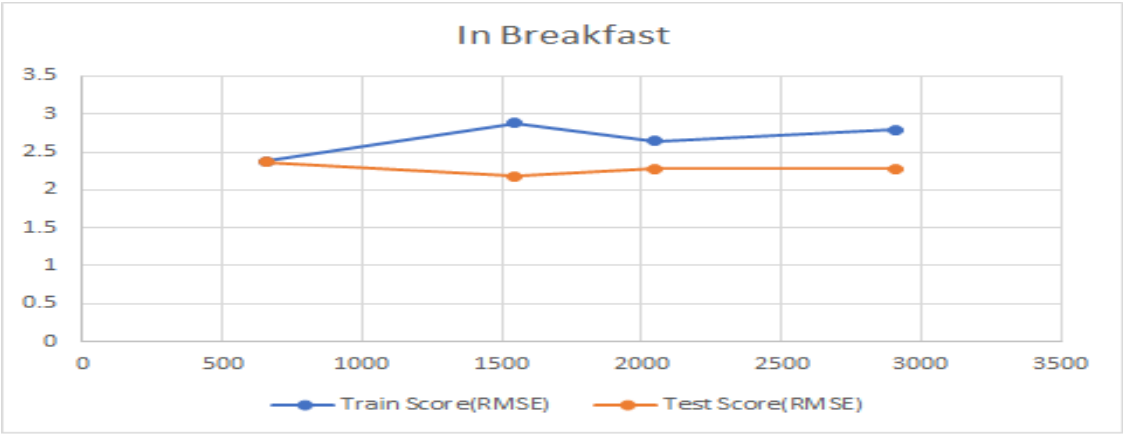


Fig: 5.2.1 Breakfast Prediction

In the Figure: 5.2.1 Train data goes high on the test data .So test data underfit over train data .RNN predicts well in breakfast. But some of reasons, prediction test score goes underfit. But prediction looks good.

Table 5.2.2: Lunch prediction

Dataset	Data(preprocessed)	Train Score(RMSE)	Test Score(RMSE)
0-14	655	1.20	2.19
0-31	1545	1.99	1.84
0-46	2050	2.21	1.80
0-64	2905	2.55	2.15

In the lunch time, we see for data 655 predicted score that means test score on average high on train score. For the next state observation test score is overfitting on the train score. So RNN predicts very good on the sequence of next state insulin level. For data 1545 train score goes underfit, because lunch time some person cannot take insulin. For data 2050 train score error 2.21 where test score error 1.80,quite good prediction by RNN.

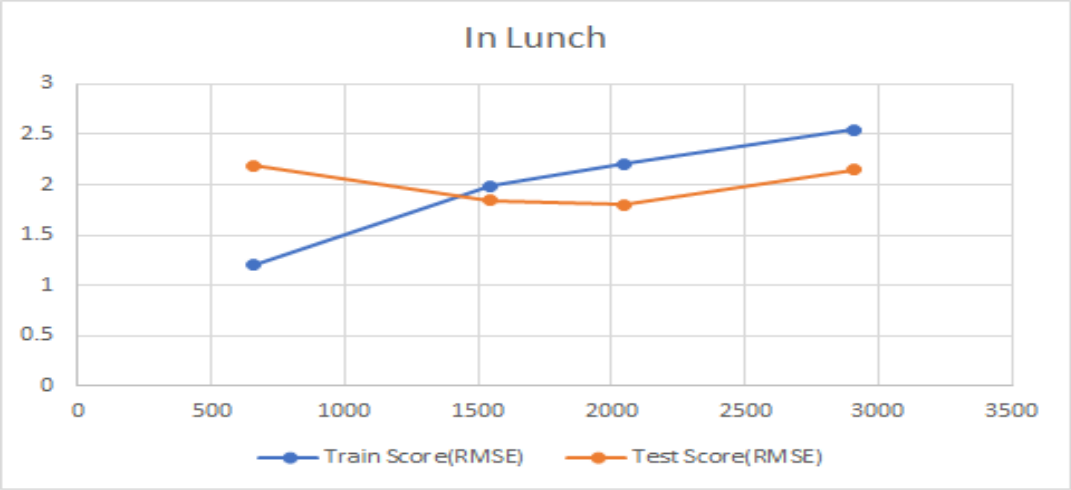


Fig: 5.2.2 Lunch Prediction

In the Figure: 5.2.2, Train score goes above of test score that means test score underfit in the prediction. On average the prediction is good for RNN LSTM, when we get 2050 data, train data slide predicted well upon the test data. In the Figure: 5.2.2 next step prediction, it predicts quite well. When we get 2905 data preprocessed, score will be increasing and loss function will be decreasing.

Table: 5.2.3 Dinner Prediction

Dataset	Data(preprocessed)	Train Score(RMSE)	Test Score(RMSE)
0-14	655	2.13	2.50
0-31	1545	2.76	2.36
0-46	2050	2.84	2.92
0-64	2905	3.20	3.55

In the Table:5.2.3, From the data (0-14) on dinner time test score upon insulin level prediction quite high that means test score on average outfit the train score in whole dataset. RNN predicted the next state of dinner time insulin very well. For 1545 data, test score gradually underfit, for 2050 data test score over fit and predict next state sequence very well. RNN computes this with time series .For time series prediction ,RNN gives a long sequence of data with 100 epochs. If we maximize this epoch, then it will be good for the loss function, both train score and test score.

Fig: 5.2.3 Dinner Prediction

In the Fig: 5.2.3 test score goes up over the train score .Test score perfectly fit on the actual data. For train score its prediction is quite well though.

Table: 5.2.4 Bedtime Prediction

Dataset	Data(preprocessed)	Train Score(RMSE)	Test Score(RMSE)
0-14	655	1.67	2.60
0-31	1545	3.57	6.20
0-46	2050	3.17	4.74
0-64	2905	2.97	3.26

In the Table:5.2.4from(0-14) data set test score over fit,from (0-31) data set test score overfit more and this time test score fail to predict from the actual data, from (0-46) test score overfit through train score,(0-64) test score overfit upon the train score.When data increases,lstm predicts well next step sequence.

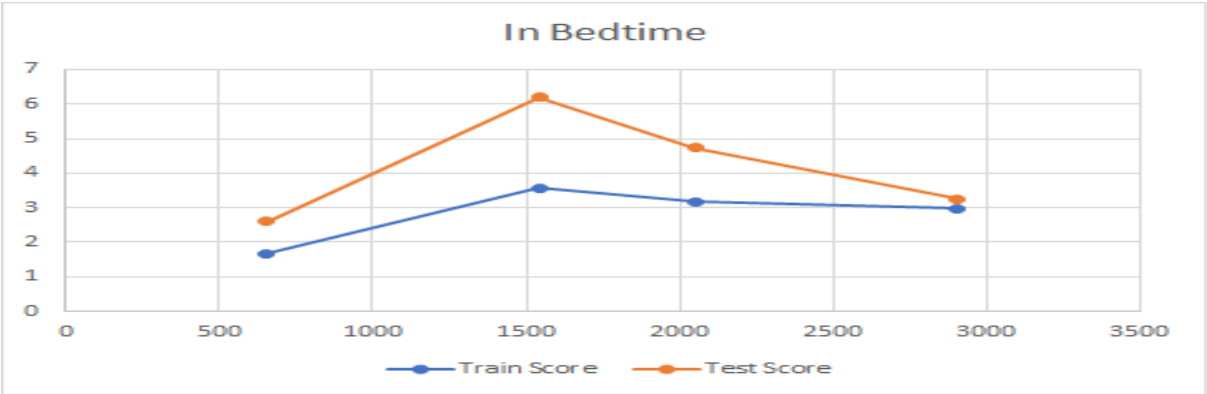


Fig:5.2.4 Bedtime Prediction

In the Figure: 5.2.4, test score of each dataset improves the train score. Test score overfits the train score and prediction is quite well on this Figure: 5.2.4.RNN LSTM just predicted well on actual data in bedtime.

5.3 Artificial Neural Network:

Artificial neural network combined of input layer, hidden layer and output layer. Here in ANN trained data with same number of epoch and calculate MSE and accuracyto prediction on train data and find how to measure loss and MSE based on train data with accuracy.

For Bedtime:

Table:5.3.1Bedtime Prediction

Data Set	Data	Epoch	Training Data	MSE (%)	Accuracy
0-14	655	655	588	4.758682193	39.39393939
0-31	1545	1545	1389	6.154442058	54.83870937
0-46	2050	2050	1844	5.524987688	48.29268307
0-64	2905	2905	2613	5.667004256	50.85910653

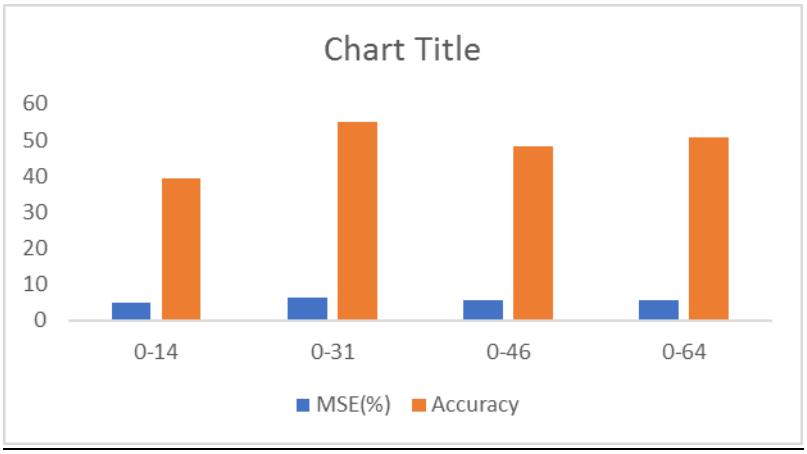


Fig:5.3.1 Bedtime Prediction

In the Fig:5.3.1 ,when we take small data with same number of epoch to predict on insulin level goes less accuracy .We take (0-14) dataset where 655 data with 655 epoch , find accuracy 39%,where large dataset predict on trained data with accuracy improved with 50%.

For Breakfast:

Table:5.3.2 Breakfast Prediction

Data Set	Data	Epoch	Training Data	MSE (%)	Accuracy
0-14	655	655	588	10.79077052	18.18181818
0-31	1545	1545	1389	13.21224167	25.16129023
0-46	2050	2050	1844	9.997701812	30.24390251
0-64	2905	2905	2613	10.95711449	17.86941581

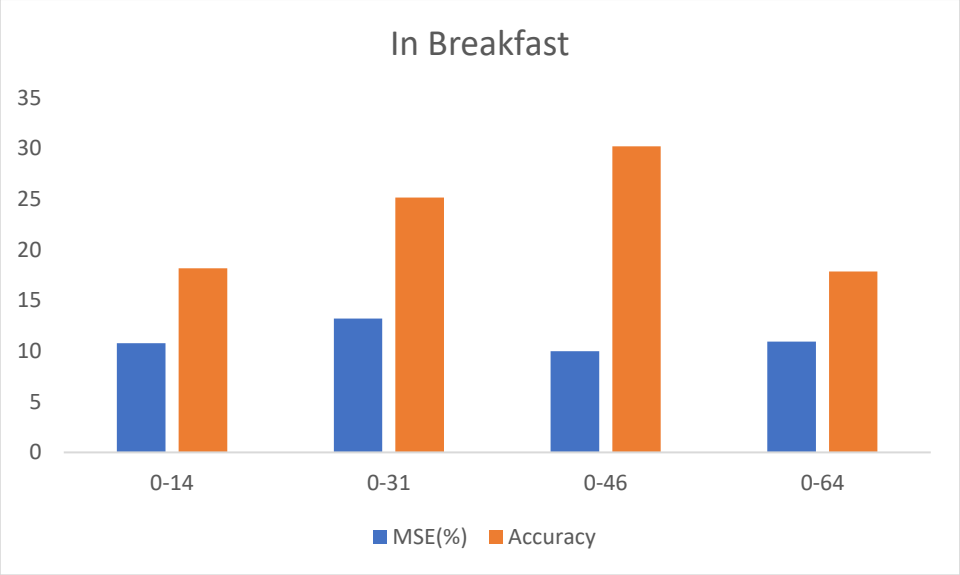


Fig:5.3.2 Breakfast Prediction

In the Figure:5.3.2, ANN we see, small datasets give too much error and accuracy does not improve, where (0-46) dataset ANN algorithm gives less error with high accuracy of insulin prediction. With same data type epoch in dinner prediction accuracy falls in the (0-64) dataset.

When large dataset goes less prediction of accuracy, prediction goes fail in time.

For Dinner:

Table:5.3.3Dinner Prediction

Data Set	Data	Epoch	Training Data	MSE(%)	Accuracy
0-14	655	655	588	26.19357392	21.21212121
0-31	1545	1545	1389	12.80214388	29.67741939
0-46	2050	2050	1844	14.44231148	27.31707321
0-64	2905	2905	2613	13.27945396	24.74226804

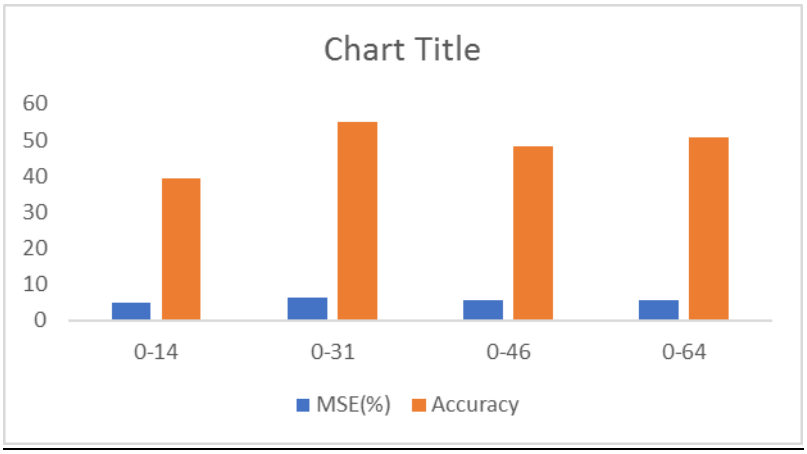


Fig:5.3.3 Dinner Prediction

In the figure:5.3.3,ANN gives prediction for small data with more errors and lessaccuracy.In big dataset, it will give small error with comfortable prediction accuracy. Epoch with same data type gives ANN a largest advantage with predictions accuracy and less error. Unfortunately (0-46) dataset give perfect accuracy for in all dataset.

For Lunch:

Table:5.3.4Lunch Prediction

Data Set	Data	Epoch	Training Data	MSE (%)	Accuracy
0-14	655	655	588	6.743752022	19.69697
0-31	1545	1545	1389	6.809755664	27.09677
0-46	2050	2050	1844	6.334257019	24.39024
0-64	2905	2905	2613	8.058921011	32.98969

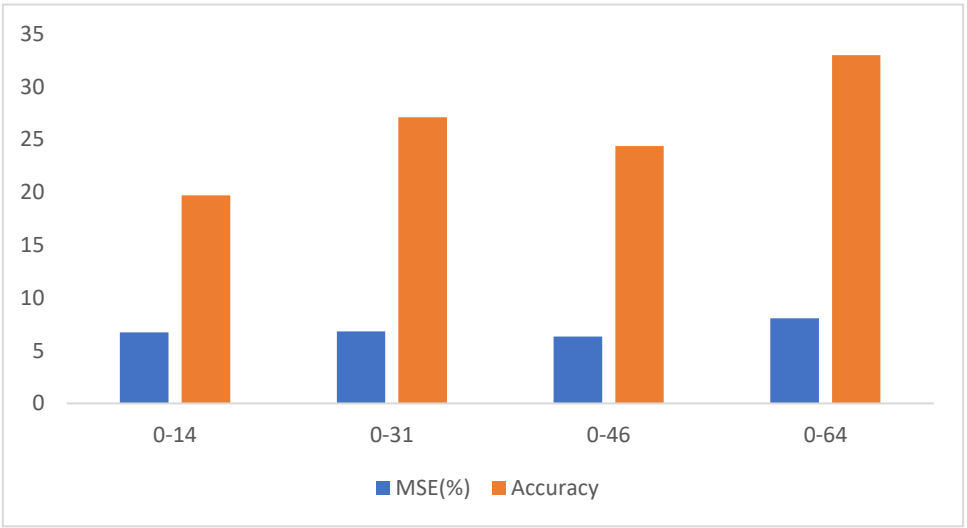


Fig:5.3.4 Lunch Prediction

In the table: 5.3.4 and Figure: 5.3.4, for prediction lunch time insulin, we find largest dataset to get most accuracy and most MSE error. ANN predicted lunch time with accurately and perfectly.

5.4 Comparison between RNN LSTM and Artificial Neural Network

RNN and ANN, both is different type neural network. ANN has three layer and RNN has three gates .We are likely to find out which algorithm give us the best prediction on the comparison with RMSE where RNN we use 100 epoch and ANN where we use same data type with same epoch.

For Bedtime:

Table:5.4.1BedtimeComparison

		RNN LSTM	ANN
Dataset	Data	RMSE (Epoch 100)	RMSE
0-14	655	2.7889	4.758682193
0-31	1545	12.7449	6.154442058
0-46	2050	10.0489	5.524987688
0-64	2905	8.8209	5.667004256

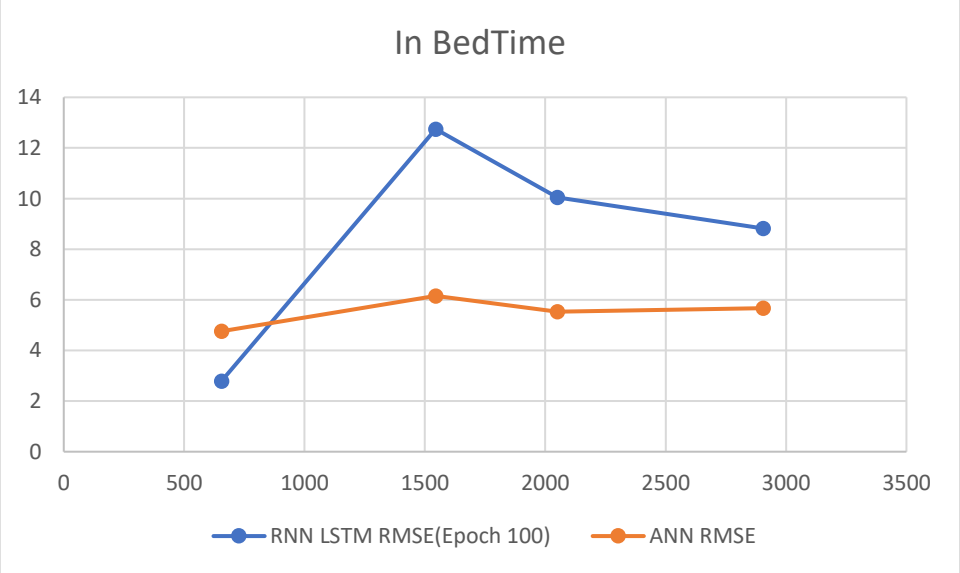


Fig:5.4.1 Bedtime Comparison

In the Figure:5.4.1,we see that for the bedtime are more RMSE with 100 epoch whereANN with same RMSE with same type of epoch.

For the comparison from dataset (0-64) ANN give better insulin level prediction than the RNN with 100 epoch. Though epoch was not same ANN gives perfect accuracy and prediction .ANN just gives perfect accuracy with RMSE.

In the graph shows that RNN goes high than the ANN .RNN fails here for prediction.Some memory gate may fail to this work.But ANN give less error with better accuracy.So ANN predicts better.

For Breakfast:

Table:5.4.2BreakfastComparison

		RNN LSTM	ANN
Dataset	Data	RMSE (Epoch 100)	RMSE
0-14	655	5.6644	10.79077052
0-31	1545	8.2944	13.21224167
0-46	2050	6.9696	9.997701812
0-64	2905	7.7841	10.95711449

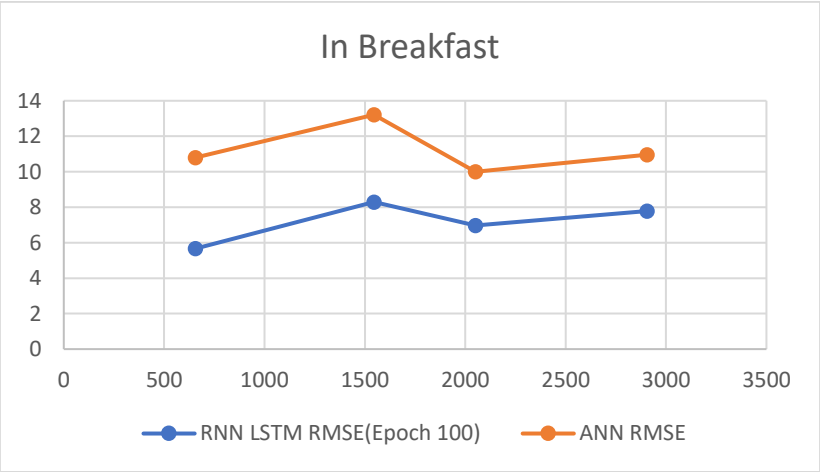


Fig:5.4.2BreakfastComparison

In the above figure:5.4.2, RNNlstm gives less RMSE with fewer epochs where ANN gives too much error with same type of errors.Here RNN predicts too well than the ANN.Solve this ANN,RNN use three gates to overcome this neural network prediction.

In the above graph RNN goes lower error with the comparison of ANN.So here RNN gives better prediction of insulin

For Dinner:

Table:5.4.3DinnerComparison

		RNN LSTM	ANN
Dataset	Data	RMSE(Epoch 100)	RMSE
0-14	655	4.5369	26.19357392
0-31	1545	7.6176	12.80214388
0-46	2050	8.0656	14.44231148
0-64	2905	10.24	13.27945396

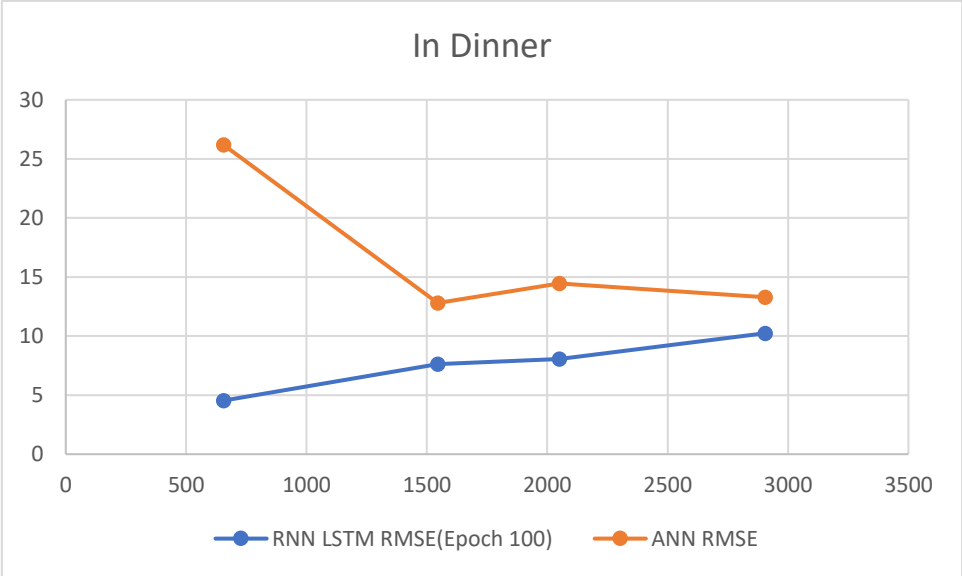


Fig:5.4.3 Dinner Comparison

In the figure: 5.4.3 for dinner RNN lstm gives less error with small epoch where ANN gives too much error where prediction on insulin. Here RNN gives better predict sequence of insulin where ANN gives less prediction with the comparison with RNN.

For Lunch:

Table: 5.4.4 Lunch Comparison

		RNN LSTM	ANN
Dataset	Data	RMSE (Epoch 100)	RMSE
0-14	655	1.44	6.743752022
0-31	1545	3.9601	6.809755664
0-46	2050	4.8841	6.334257019
0-64	2905	6.5025	8.058921011

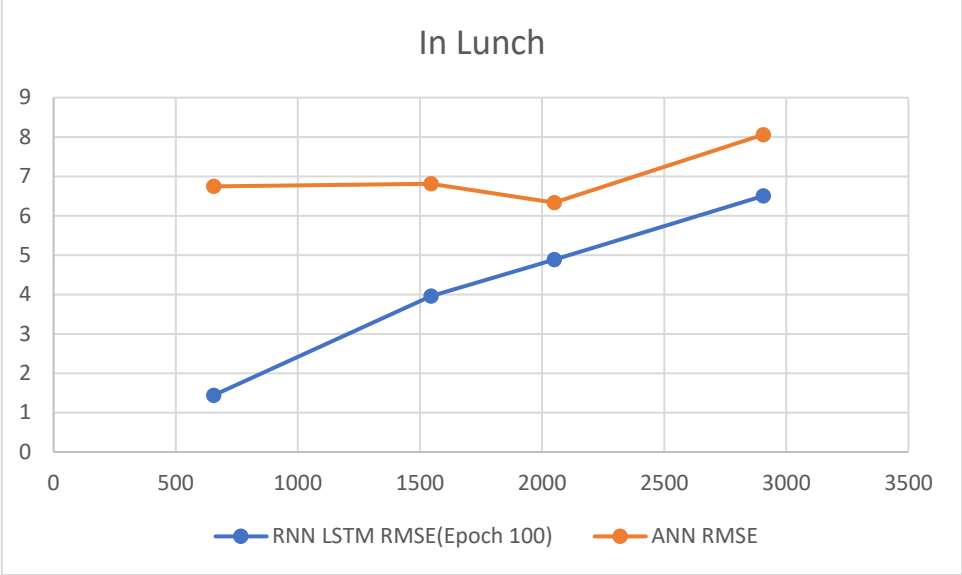


Fig: 5.4.4 Lunch Comparison

In theTable:5.4.4 comparison with RNN and ANN, RNN gives less errors prediction with 100 epoch and ANN gives too much errors with same type of data with same number of epoch. Here RNN gives better prediction than ANN.

In the Figure:5.4.4 RNN goes lower where ANN gives high errors with lots of epoch. here RNN gives perfect prediction with small epoch.

In all prediction with results and analysis that three out of four-column's sequence are predicted well by RNN. So, here we come to finish the decision that RNN gives with best prediction in the insulin level of a patient for the next state prediction and sequence prediction with less error than ANN prediction on train data.

5.5: PredictiveApriori Algorithm

In the Predictive Apriori algorithm, we use weka to find association rule. Due to numerical dataset we convert each data as a string like Dinner=insulin 9, we write dataset Dinner="nineD" for the finding frequent item sets and association rules .Here After predict Apriori implementation we find 100 association rules where in predictive Apriori doesn't give confidence level,it gives the accuracy of each association rule.We choose 92% accuracy to find best row wise pair for prediction insulin.

1. Bedtime=twoB Breakfast=fiveM Lunch=threeL 67 ==> Dinner=sixD 67 acc:(0.99482)
2. Bedtime=sixB Breakfast=fiveM Dinner=sixD 16 ==> Lunch=threeL 16 acc:(0.992)
3. Bedtime=sixB Breakfast=fiveM Lunch=threeL 16 ==> Dinner=sixD 16 acc:(0.992)
4. Breakfast=tenM Lunch=sixteenL 7 ==> Dinner=twelveD 7 acc:(0.97711)

5. Bedtime=twoB Dinner=tenD Lunch=tenL 7 ==> Breakfast=tenM 7 acc:(0.97711)
6. Breakfast=fifteenM Dinner=fifteenD 6 ==> Lunch=sixL 6 acc:(0.96911)
7. Bedtime=threeB Breakfast=fiveM Dinner=sixD 71 ==> Lunch=threeL 68 acc:(0.95928)
8. Breakfast=fiveM Lunch=threeL 192 ==> Dinner=sixD 185 acc:(0.95916)
9. Breakfast=twelveM Lunch=fourteenL 5 ==> Dinner=twelveD 5 acc:(0.95512)
10. Breakfast=eightM Dinner=elevenD 5 ==> Bedtime=threeB 5 acc:(0.95512)
11. Breakfast=oneM Dinner=threeD 5 ==> Lunch=zeroL 5 acc:(0.95512)
12. Bedtime=threeB Breakfast=tenM Lunch=sevenL 5 ==> Dinner=tenD 5 acc:(0.95512)
13. Bedtime=twoB Breakfast=fiveM Lunch=oneL 5 ==> Dinner=sevenD 5 acc:(0.95512)
14. Bedtime=oneB Breakfast=fiveM Dinner=sixD 36 ==> Lunch=threeL 34 acc:(0.94609)
15. Breakfast=fiveM Dinner=sixD 195 ==> Lunch=threeL 185 acc:(0.94416)
16. Dinner=oneD 4 ==> Lunch=zeroL 4 acc:(0.92949)
17. Bedtime=threeB Lunch=fourteenL 4 ==> Dinner=twelveD 4 acc:(0.92949)
18. Breakfast=fifteenM Dinner=eighteenD 4 ==> Lunch=sixL 4 acc:(0.92949)
19. Breakfast=eightM Dinner=twoD 4 ==> Lunch=zeroL 4 acc:(0.92949)
20. Breakfast=eightM Lunch=nineL 4 ==> Bedtime=threeB 4 acc:(0.92949)

For insulin prediction, all the analysis on RNN, ANN and predictive, we tell that if a patient or doctor want to know the next state prediction, we can easily tell them by applying RNN Long Short Term Memory algorithm which gives a higher prediction analysis than above all other algorithm.

Chapter 6: Conclusion & Future Work

6.1 Conclusion

We have described a machine learning approach of predicting insulin dose levels and presented the results of experiments. From the graphs we have shown the comparison between predicted data & data and it can be observed that results are very encouraging and reliable. Our method has passed all criteria of predicting a long sequence problem.We have trained our data in a time consuming process and fulfill the achievements. In conclusion we can say that if we train our system with more input data set, it generate more error free insulin prediction.

6.2 Future Work

A) Prediction for remaining Insulin dose codes:

We have done prediction using RNN for Regular Insulin dose (code=33). In future we will work for other Insulin doses like NPH insulin dose (code=34), Ultralente insulin dose (code=35).

B) Prediction using Beum Welch Algorithm:

We have reached our goal using LSTM (RNN). But we want to implement other sequence prediction algorithms like Beum Welch with this same data set to compare the results & also to see how much efficient is Beum Welch than our LSTM.

C) Compare with Randomized Algorithm:

We would like to compare the results by implementing randomized algorithm.

Bibliography

[1]"Understanding LSTM Networks -- colah's blog", Colah.github.io, 2017. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 29- Nov- 2017].

[2]"Time Series Prediction Using Recurrent Neural Networks (LSTMs) - DZone AI", dzone.com, 2017. [Online]. Available: https://dzone.com/articles/time-series-prediction-using-recurrent-neural-netw. [Accessed: 29- Nov- 2017].

[3]"Forbes Welcome", Forbes.com, 2017. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2016/12/06/what-is-the-difference-between-artificial-intelligence-and-machine-learning/#283aaee12742. [Accessed: 29- Nov- 2017].

[4]"Overview of Artificial Neural Networks and its Applications", Hacker Noon, 2017. [Online]. Available: https://hackernoon.com/overview-of-artificial-neural-networks-and-its-applications-2525c1addff7. [Accessed: 29- Nov- 2017].

[5]"A beginner's tutorial on the apriori algorithm in data mining with R implementation | HackerEarth Blog", Blog.hackerearth.com, 2017. [Online]. Available: http://blog.hackerearth.com/beginners-tutorial-apriori-algorithm-data-mining-r-implementation. [Accessed: 29- Nov- 2017].

[6]"Apriori algorithm", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Apriori_algorithm. [Accessed: 29- Nov- 2017].

[7]"What is Machine Learning? A definition - Expert System", Expertsystem.com, 2017. [Online]. Available: http://www.expertsystem.com/machine-learning-definition/. [Accessed: 29- Nov- 2017].

[8]"The Difference Between AI, Machine Learning, and Deep Learning? | NVIDIA Blog", The Official NVIDIA Blog, 2017. [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/. [Accessed: 29- Nov- 2017].

[9]"Deep Learning Use Cases - Data Science Pop-up Seattle", Slideshare.net, 2017. [Online]. Available: https://www.slideshare.net/dominodatalab/data-science-popup-seattle-deep-learning-use-cases. [Accessed: 29- Nov- 2017]