A Major Project Final Defense Report on

Construction and Implementation of Neural Network Library

Submitted in Partial Fulfillment of the Requirements for
the Degree of **Bachelor of Engineering in Computer engineering**
under Pokhara University

Submitted by:

Diwash Ale, 15358

Nisseem Rana Magar, 15377

Bikash Singh, 15371

Nischal Hada, 15373

Supervisor:

Kamal Chapagain

Date:

25nd Nov 2019

Department of Computer Engineering
NEPAL COLLEGE OF INFORMATION TECHNOLOGY
Balkumari, Lalitpur, Nepal.

ACKNOWLEDGEMENT

This project would not have been possible without the joint efforts of many individuals. It has been a pleasure for us to acknowledge the assistance and contributions that were very important and supportive throughout the project. We would like to extend our sincere thanks to all of them.

We owe special thanks to a number of people who has devoted much of their time and expertise without which it would have been very difficult for us to complete our project entitled "Construction of neural network library and its implementation".

We are highly indebted to our supervisor Kamal Chapagain for his valuable guidance throughout the project development period and for providing technical support with suggestions which helped our project to grow and foster to a certain level we didn't think of reaching in such a short period.

We would like to extend our special thanks to Roshan Chitrakar for his technical guidance, valuable suggestions, attention and time for our project.

Last, but not the least, we would like to thank our teachers and colleagues who have been knowingly or unknowingly the part of this project and lent support and views during the entire development time.

ABSTRACT

Artificial Neural Network (ANN) is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. This project is about understanding how neural networks work and designing our own neural network that can be trained to solve real world problem. Perceptron which is the basic element of neural network is created and are tested to solve logical gates. It can only solve simple problems such as OR, AND, NOT, NAND, etc gates but cannot deal with real world program due to lots of variables and complexity nature of such problem. Therefore, library for multilayer perceptron (also known as Neural Network) is created which is more advance version of perceptron. Logical gate problems such as XOR and XNOR that a single perceptron cannot solve are solved with this NN library. Later, this library is used to train NN that is able to recognize handwritten digit. This library is also used for creating an agent that learns to play game such as T-rex run (Google's Dino game) with the technique called Neuroevolution of augmenting topologies (NEAT). And finally, this library is used for creating a web interface of disease prediction system that can be used in a real world.

Keywords: Feed Forward, Fully Connected, Gradient Descent, Multilayer Perceptron, Neural Network, Neuroevolution (NEAT)

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

A neural network is a population of neurons interconnected by synapses to carry out a specific function when activated inside human brain. Artificial Neural Network (ANN) emerged from this drive for biologically inspired intelligent computing and went on to become one of the most powerful and useful methods in the field of artificial intelligence.

This Library contains functions that are required for a perceptron and neural network to work. Perceptron is the basic and key element for our Neural Network. It is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time. A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. We first implement perceptron algorithm that enable us follow the creation of multilayer perceptron algorithm.

With this library, neural network can be trained with known data as supervised learning that can use for solving problem such as logical gates, predicting values, finding pattern, etc. It has the feature that shows the structure of the network. The training process of the NN can also be visualized that show how the weights and the biases of the network changes with each iteration of training phases. With this library we can also implement genetic algorithm on neural network that is about evolving neural network to its best form which is used as a brain for as learning agent in playing video games, robots carrying out specific tasks, etc.

1.1 Problem Statement

Deep learning is one of many machine learning algorithms to enable a computer perform a plethora of tasks. Deep learning uses the concept of neural networks which is modeled after a human brain [1].

"If you want to learn something, why not create it?"

We build this library to understand deeper about neural network basic working mechanism, algorithm and mathematical implementation. We can add new feature, algorithm and even modify existing algorithm according to our need. It becomes easy to understand it because we created it by ourselves and can be constantly updated with new features and optimization. There are lots of popular library for neural network such as Google's TensorFlow, Microsoft Cognitive Toolkit (CNTK), Caffe from Berkeley Vision and Learning Center, Theano, Torch, etc. They contain lots of features that a person or organization may or may not want. These library and platform can be hard and confusing for beginner people or student that want get start with. Our library aims for those who is new to machine learning and neural network field; who want to learn about the basic concept and working mechanism of neural network.

1.2. Project Objectives

The main objective of our project is to construct and implement our own neural network library to solve logical gates problem, recognize the handwriting digits, create a learning agent and predict diseases based on symptom provided. There are some specific objectives which are as follows:

- Make our library as beginner friendly as possible.
- Make our library easily understandable for user about the mathematical concept of the neural network.
- Make our library easier to use and implement it with many variable and target.
- Add feature to our library that show the structure of neural network with weights and biases.
- Add feature to our library that can visualize the training process of the neural network.
- Implement this library to solve the entire logical gate problem.
- Implement this library by training it to find patterns and predict values.
- Implement this to create a brain for the learning agent that can play games.

1.3 Scopes and limitations

This Neural Network Library has the power of flexibility. Once established, it can be applied to almost anything relative to its field. This Library can work easily on problems with many variables. For a problem with a large number of constrained inputs, it's easy for this library to work out the answer.

This library is targeting the peoples who are interested in the field of Neural Networks and Machine learning. It has been structured in a very simple manner which makes it easy for the users (being new or inexperienced) to implement the library. Due to the simplicity of this library, it can be modified in any manner as required making it flexible/versatile. This library is capable of doing task such as pattern recognition, values prediction, performing NEAT algorithm for genetic learning, etc. The NEAT algorithm is well described in here [2]. Companies that deals with machine learning and predicting their result and outcome can use this library with ease without having to deal with more complex and advance side of most of the popular libraries.

That said, there are also some key limitations preventing this library from its full potential. This library is current based on feed forward neural network. System that requires other type of neural networks cannot be done with this current library. People with knowledge only about basic neural network but has no concept about programming can find it hard to implement this library for solving their problem. Space and time complexity can be issues since less effort have been done in optimization of this library rather focus on the working neural network.

2. LITERATURE REVIEW

This section consists of the description of literature study done on the different topics directly related to the project. It aims to provide readers a theoretical base for the project and develop an understanding of the nature of the project.

The book [3] is about step-by-step gentle journey through the mathematics of neural networks and making own using the Python computer language. It contains a guide that starts from very simple ideas and gradually builds up an understanding of how neural networks work. The ambition of this guide is to make neural networks as accessible as possible to as many readers as possible. Part 1 is about ideas. It introduces the mathematical ideas underlying the neural networks, gently with lots of illustrations and examples. Part 2 is practical. It introduces the popular and easy to learn Python programming language, and gradually build up a neural network which can learn to recognize human handwritten numbers, easily getting it to perform as well as networks made by professionals. Part 3 extends these ideas further. It is about pushing the performance of the neural network to an industry leading by large margin only simple ideas and code, test the network on own handwriting and get it all working on a Raspberry Pi.

Neural Networks have been around for long time. Neural networks have been applied to a very broad range of tasks in many different disciplines. Several hundred reports have been published in recent years on attempts to apply neural networks to specific problems [4]. There are many popular libraries implementing different type of neural network. For example, Genann[5] is a minimal, well-tested open-source library implementing Feedfordward artificial neural networks (ANN) in C. It's entirely contained in a single C source file and header file, so it's easy to add to this project. [5] has a focus on being easy to use but is also very extensible. It has features such as: ANSI C with no dependencies, contained in a single source code and header file, fast and thread-safe, easily extendible, implements back propagation training. It is also compatible with alternative training methods (classic optimization, genetic algorithms, etc.) and includes examples and test suite. It is released under the zlib license - free for nearly any use.

Similarly, one of the most popular library existing around is TensorFlow[6] which is a free and open-source software library for dataflow and differentiable programming across a range of

tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. Currently, it is the most famous deep learning library in the world. Google uses these deep neural networks to improve its services such as Gmail, Google Photo and Google search engine. They build a framework called Tensorflow to let researchers and developers work together on an AI model. It was first made public in late 2015, while the first stable version appeared in 2017. Tensorflow architecture works in three parts: Preprocessing the data, Build the model, Train and estimate the model. It is called Tensorflow because it takes input as a multi-dimensional array, also known as tensors. We can construct a sort of flowchart of operations (called a Graph) that we want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output. This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

3. PROPOSED METHODOLOGY

We have done our tasks by following these methodologies for the application of knowledge, skills, tools and techniques to a broad range of activities in order to meet the requirements of our project.

3.1. Software Development Life Cycle

The framework we have used for developing this project is Iterative model. This model brings the cyclic development in the waterfall model. New functionalities will be added as each cycle or iteration is completed. The phases of the model are: Analysis, Design, Coding and Testing. The library repeatedly passes through these phases in iteration with new feature addition and modification.

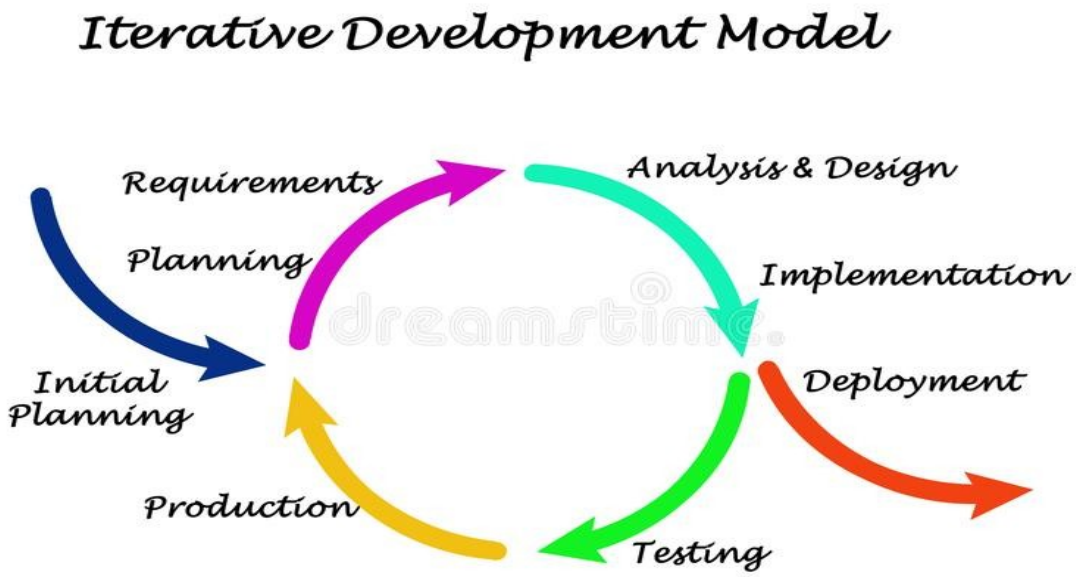

Figure 1: Iterative Model [7]

3.1.1. Requirement Analyze Phase

In requirement analysis phase, the different requirements that the final product was supposed to fulfill were determined, gathered and analyzed. The various requirements for our Neural Library were creation of perceptron, creation of feed forward neural network, implementing them, visualizing and finding better ways to demonstrate to non-technical user, finding right tools to use and gathering training data for pattern recognition of hand written digit. With successive iteration, this phase also involved the identification of requirements which were not fulfilled in the previous builds of the library, but are still needed for the development

3.1.2. Design Phase

In design phase, a software solution to meet the requirements gathered in the previous phase was designed. Design phase was largely comprised of designing structure of neural network during its creation, while implementation it for solving logical gate problem, finding pattern for predicting handwritten digits. At the later phase, we also created game, level, character and object design that is needed for implementing and testing of our Neuro evolution simulation.

3.1.3. Coding Phase

In implementation phase, the software was coded as per the design pattern mentioned earlier. This phase involved the creation of new features, plus the upgrade and enhancement of previously coded feature as per the final requirements.

3.1.4. Testing phase

In this phase, the source code was built and tested in some sample devices to find out any errors that exist in the library. The library's new feature was also tested. With each testing a list of changes to the library developed, is suggested and the changes will be applied to it. It was found that NN with the structure of large input, hidden or output nodes require large computational resource and power. Computer with low resource tend to hang and consume more time when testing the implementation of our library.

3.2. Tools and Technologies

Application software and tools used during the development of this project are:

S.N.	Name	Description/Purpose
1.	Python	Python is an interpreted, high-level, general-purpose programming language that lets you work quickly and integrate systems more effectively. It is the main programming language used in this project.
2.	NumPy	NumPy is a library for the Python. It adds support for large, multi-dimensional arrays and matrices, along with a large collection of high level complex mathematical functions to operate on these arrays.
3.	Matplotlib	Matplotlib is a plotting library for the Python and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like wxPython, Qt, or GTK+.
4.	TkInter	Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. It is python library for creating animation.
5.	scikit-learn	Scikit-learn is a free software machine learning library for the Python. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. But we used it in our project for datasets only.
6.	Pillow	Python Imaging Library is a free library for the Python that adds support for opening, manipulating, and saving many different image file formats.
7.	Pygame	Pygame is a cross-platform set of Python modules designed for writing video games with physic engine and graphic. It includes computer graphics and sound libraries designed to be used with the Python programming language.
8.	Django	Django is a Python-based free and open-source web framework, which

		follows the model-template-view architectural pattern. It is maintained by the Django Software Foundation.
9.	xlrd	xlrd is a library for reading data and formatting information from Excel files, whether they are .xls or .xlsx files.
10.	Visual Studio Code, Vim	Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS. It includes support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring. Vim is a clone, with additions, of Bill Joy's vi text editor program for Unix. They are the IDE used for writing and editing source code in this project.

3.3. System Design

This section documents the various steps taken during the design phase of our library.

3.3.1 Class Diagram

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagrams of key elements of our library are shown in figure below:

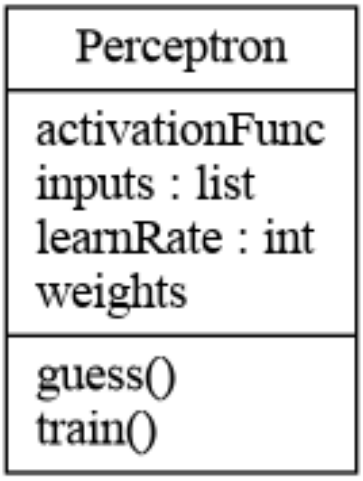

Figure 2: Perceptron Class Diagram

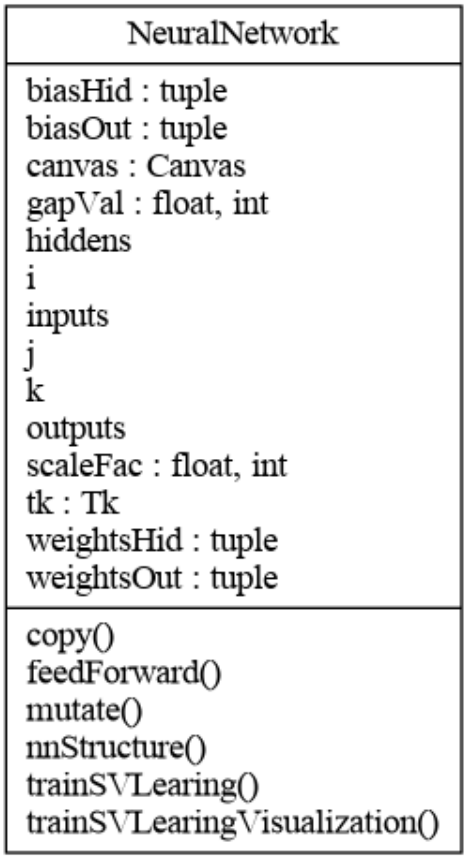

Figure 3: NeuralNetwork Class Diagram

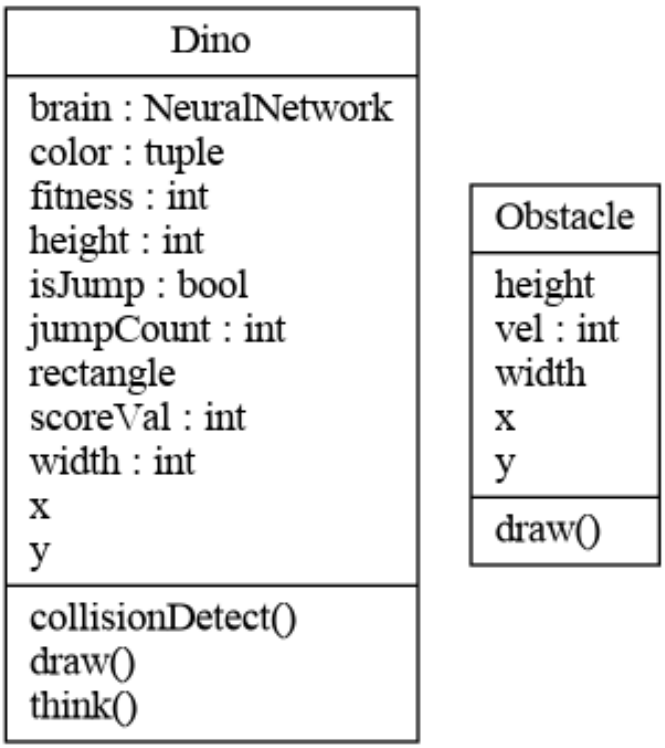

Figure 4: TRex game Class Diagram

3.3.2 Network Model Design

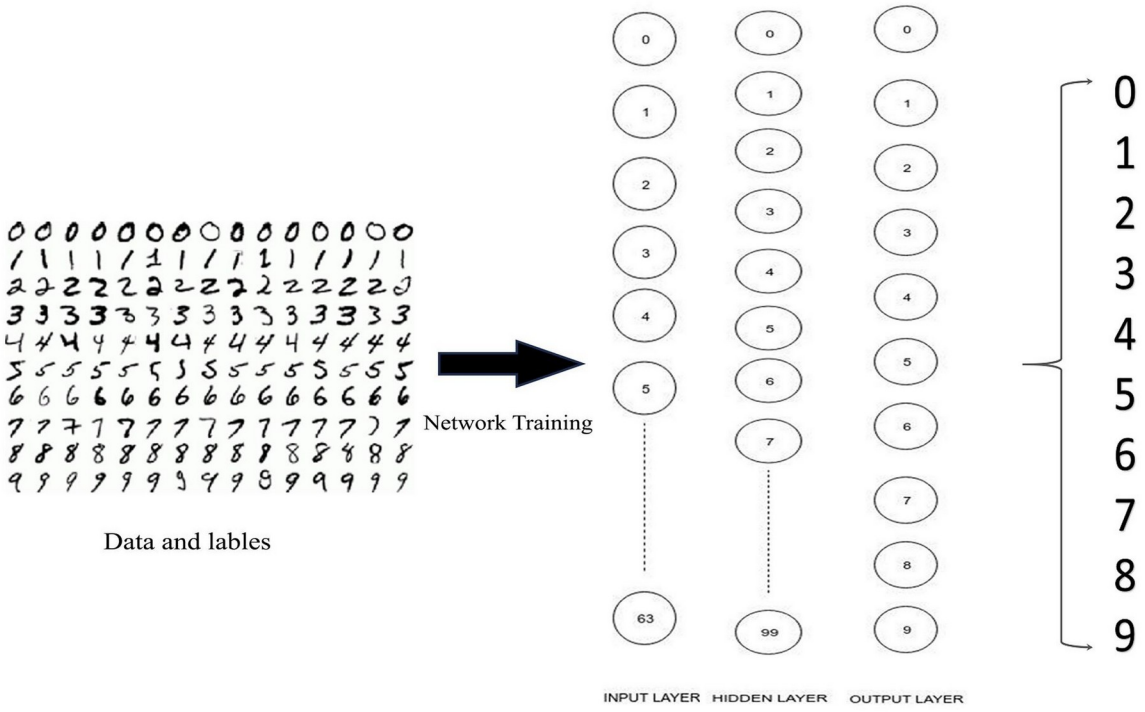

Figure 5: Image Recognition Model

The resolution of our training image of handwritten digit is 8x8 due to which there are 64 nodes in input layer, each representing pixel of an image. Second layer is hidden layer. It can contain any number of nodes. The more hidden nodes, the more data are needed to find good parameters, but can find more complex decision boundaries easily. Lastly, there is an output layer which contains 10 nodes. Each node in layer represents the output of the network which in our case is the prediction of the digit from 0 to 9.

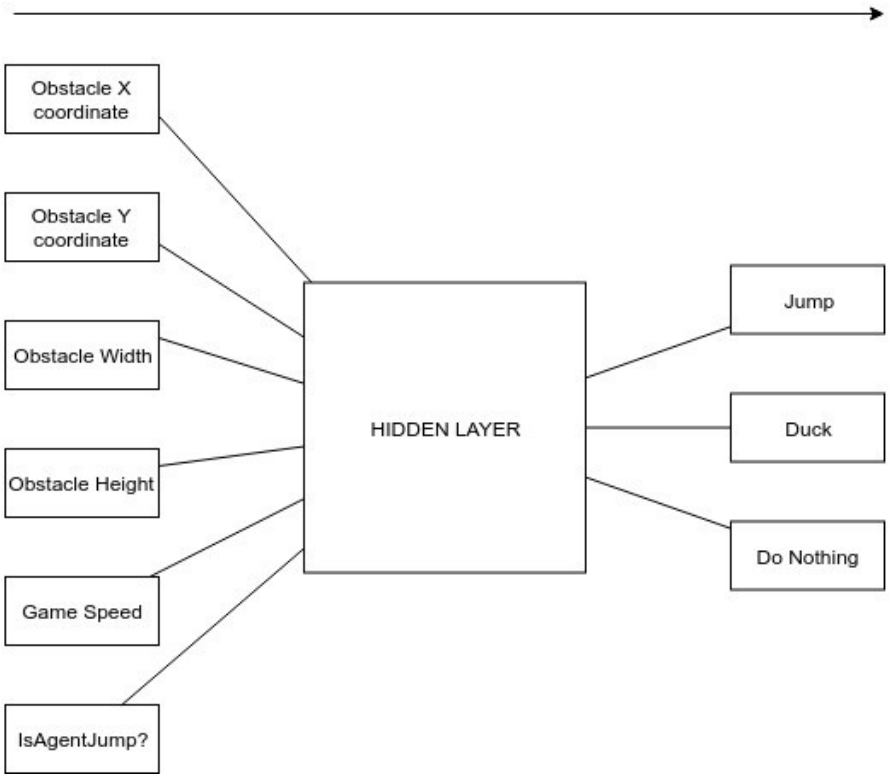

Figure 6: Learning Agent Model

To train our agent to play game, it needs to think as any human player does when playing any game. So, we need to pass the parameter that a player would look for when playing the game. In our Trex game, there are many factors that are looked for when doing any kind of action. We look for the coordinates of the obstacle, obstacle's dimension, game's speed and if our character is jumping or not. These inputs are pass to the network and it gives on any one of the outputs among jumping, ducking or do nothing.

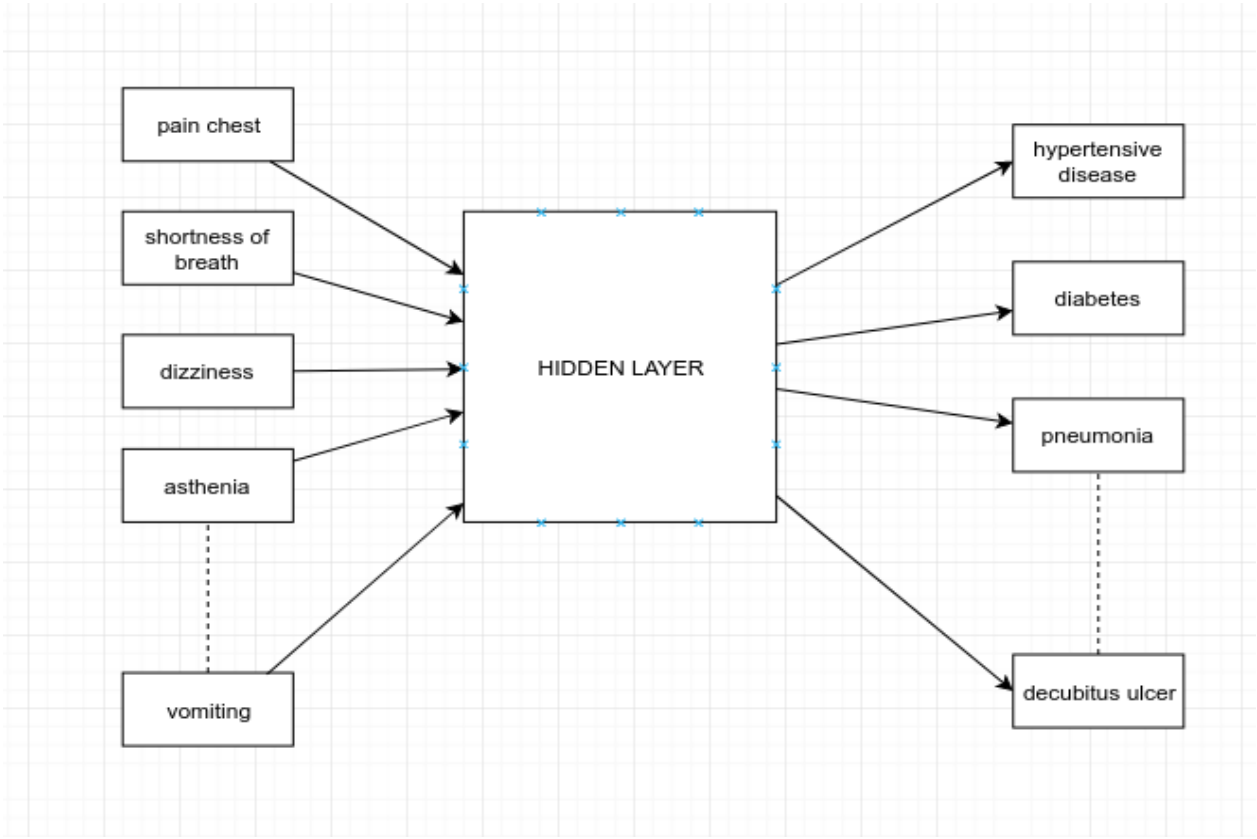

Figure 7: Disease-Prediction Model

Here in this network, disease symptoms are given in as input and the predicted disease are given out as output.

4. PHASES AND IMPLEMENTATION

As we have use Iterative methodology, our project was complete in 5 phases. These phases are explained below in details:

4.1. Phase 1 (Creation of Perceptron):

This phase included the development and implementation of perceptron and finding its limitations. We started by creating a class for perceptron. It includes constructor function that takes argument of number of input node and generates random weight based on it. It contains guess function that takes input data as argument; that sum the multiplication of weights and input values and pass it through activation function which generates the output. Finally, it contains train function that takes input data, known data and learning rate as argument. In this function, perceptron trains itself by changing weights and bias according to error between target and output of it.

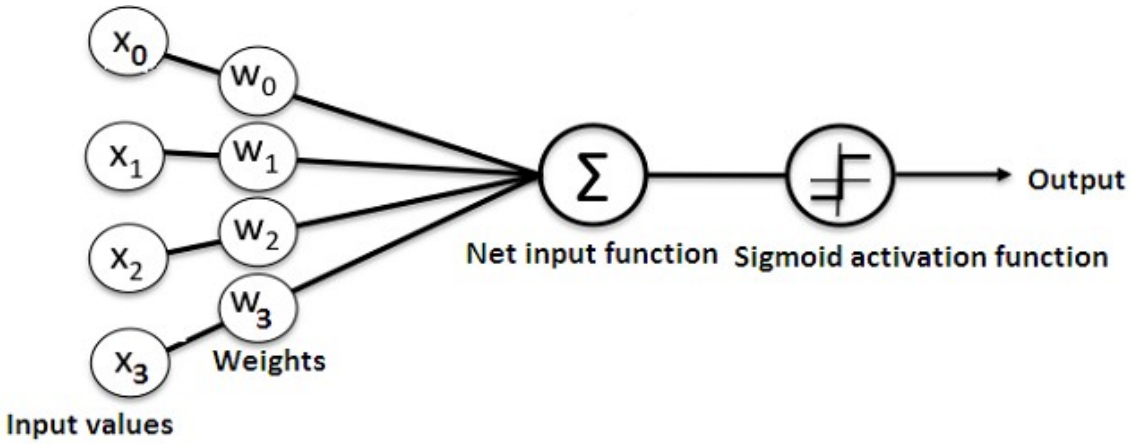

Figure 8: Perceptron

Implementation: After the creation of perceptron class, we needed to test it. We used this class to solve basic logical gates such as OR, AND, NOT, NOR and NAND. We even created visualization of it learning process and generation of output.

Limitation: Though perceptron was able to solve basic gates, but it cannot solve complex gate such as XOR and XNOR. It more technical term, perceptron is not able to solve non-linearly separable problems. This is why neural network comes into play.

4.2. Phase 2 (Creation of Neural Network):

This phase includes the creation of a Feed Forward Neural Network class. Creating neural network class is huge and can take time. So, we plan and separated difference mathematical concept, the function that are going to be used.

An NN comprises of the following components:

- An input layer that receives data and pass it on
- A hidden layer
- An output layer
- Weights between the layers
- A deliberate activation function for every hidden layer, we'll employ the sigmoid activation function.

Constructor: This function takes number of input nodes, hidden nodes and output nodes as argument and generates random weight and bias according to them.

FeedForward: This function takes input vector as argument. These input vectors get multiplied with hidden layer weight matrix, added with hidden layer bias vector and passed through sigmoid function to generate hidden layer output matrix. Similarly, those hidden layer output matrix gets multiplied with output layer weight matrix, added with output layer bias vector and passed through sigmoid function to generate the final output vector.

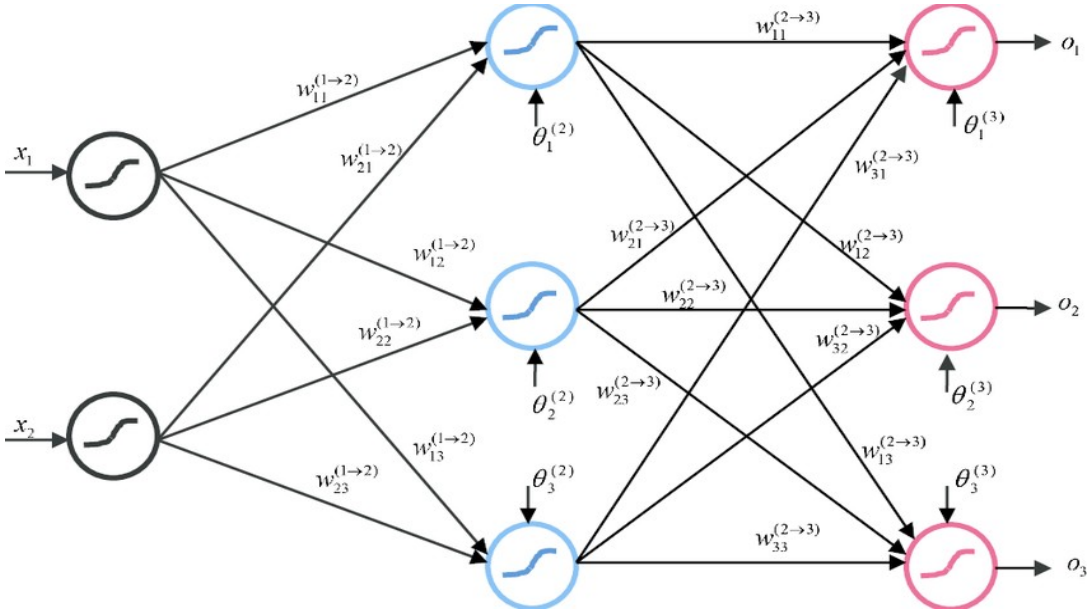

Figure 9: FeedForward Neural Network

Supervised Training: This function takes input vector, target vector and learning rate as an argument. Output vector from the given input vector is generated using above feedforward function. That output vector is compared with target vector and error vector is calculated. These are the output layer error. With output layer errors, we can calculate hidden layer errors. The output weight and bias matrix are modified according to output layer error using gradient descent [8]. Similarly, the hidden layer weight and bias matrix are modified according to hidden layer errors using gradient descent.

NN Structure: This function draws the network structure of the current Neural Network. It shows all the nodes in input, hidden and output layer. The lines drawn between nodes are weights. Weights with green color are positive weights and weights with red color are negative weights.

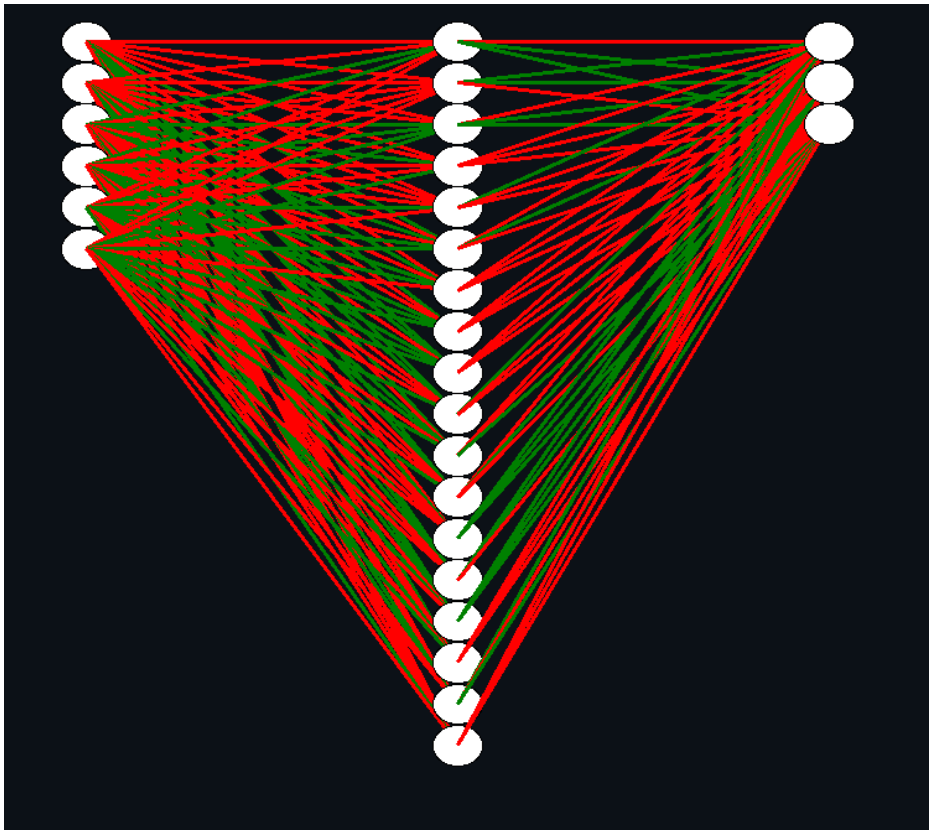

Figure 10: Neural Network Structure

NN Training Visualization: This function visualizes the training process of the network where weights and biases are constantly changing. This function is similar to supervised training function. It also takes input vector, target vector and learning rate as an argument. The only difference between these two functions is supervised training process is done in background and it is fast compared to this function because of more resource use due to visualization of network.

4.3. Phase 3 (Implementation of Neural Network):

In this phase we tested and implemented NN class in different tasks such as solving logic gates that perceptron could solve. We visualize its learning phase and how it generates output. Later we wanted our library to be tested in pattern recognition. So, we used sklearn datasets to get handwritten digit. We trained our NN with 1500 these data and tested with 200 data which was able to give accuracy around 80%. After training and testing, this program will provide a drawing canvas that allows user to draw which neural network will try to predict a digit based on that drawing. A new window will pop up when user hit save button that will show image that user drew and the predicted image of digit that NN was trained with.

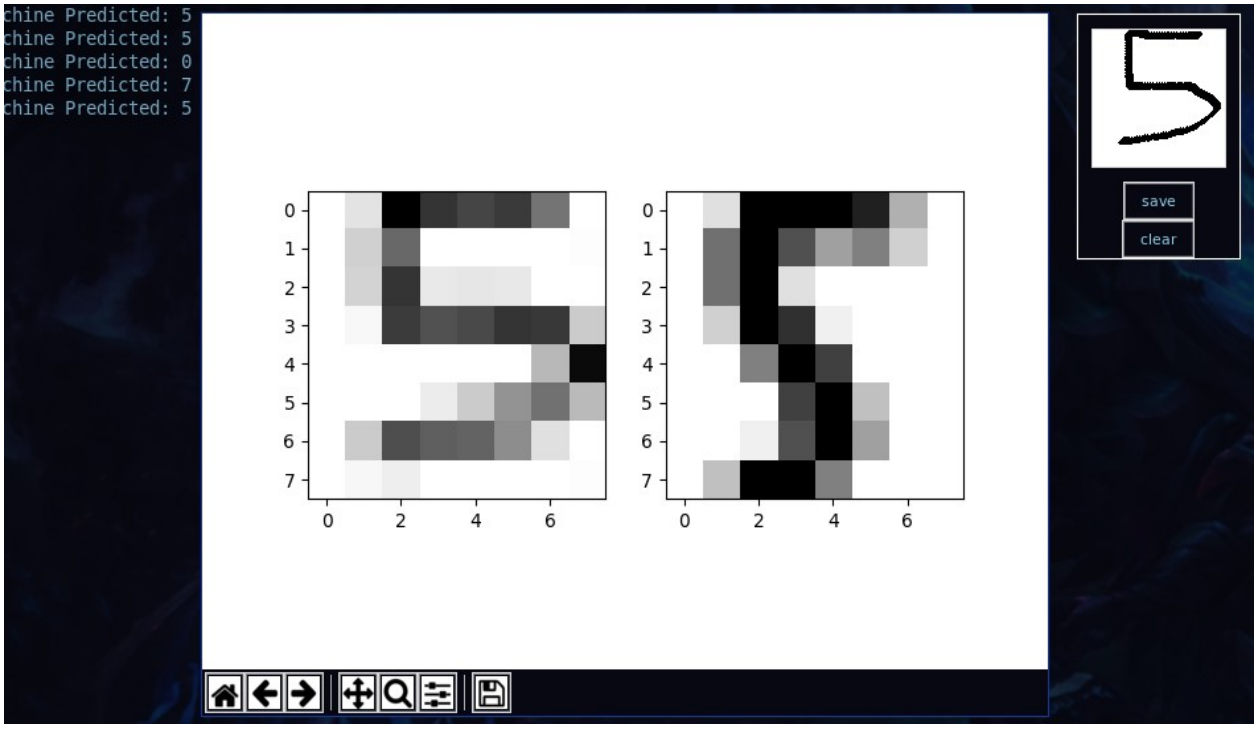

Figure 11: Digit Recognition Program

4.4. Phase 4 (Evolving Neural Network):

We used NEAT (Neuroevolution of augmenting topologies) algorithm that involves evolving neural network based on genetic algorithm. Neural network act as a brain for our game playing agent that perceive game environment, obstacle, and mechanism. With each generation, NN with the poor fitness value are filter out while the NN with high fitness value thrive to pass on its genes to next generation. This phenomenon is based on Darwin's Natural Selection which eventually leads to evolution of NN. With right amount of simulation, later generation can play the game at the pace human could only imagine.

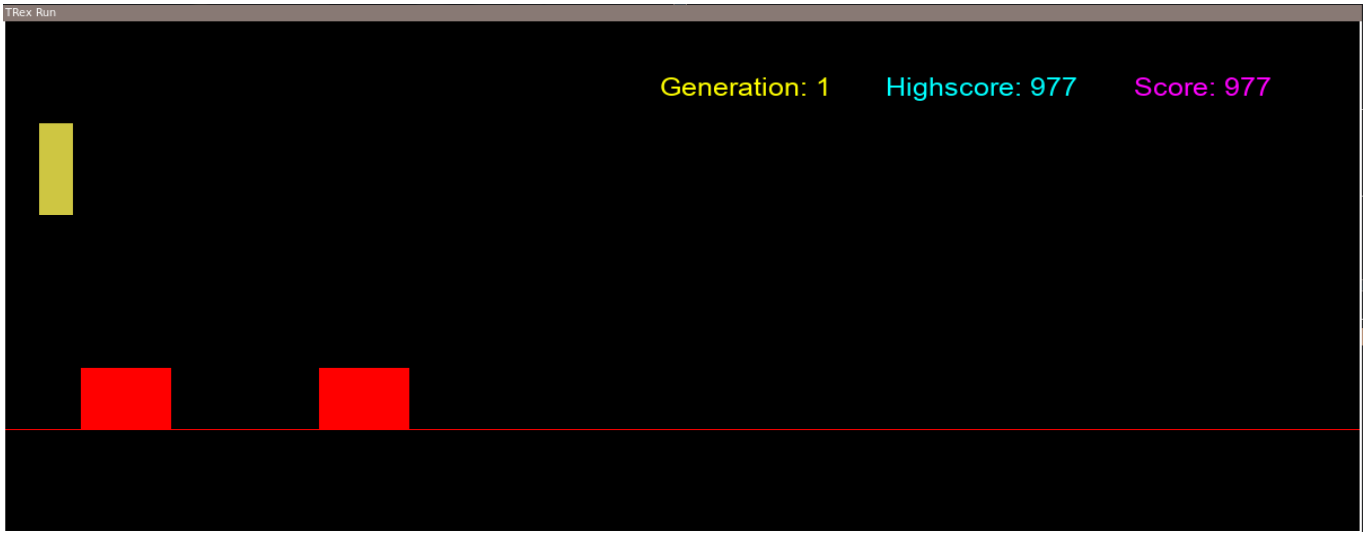

Figure 12: Trex-game-NEAT

4.5. Phase 5 (Real World Problem Implementation; Disease Prediction System)

We wanted to use this library into something that would be applicable for real world scenario. We thought of creating a system that will take symptoms of disease as input and give back the disease that is relevant to those symptoms. So, we created an application that consist of web interface as the front view for the user to input their symptoms and give out the predicted disease based on percentage of certainty. The web interface consists of 20 most common symptoms and search and select feature to choose symptoms.

On the backend we created a script file that takes dataset that consist of symptoms and their corresponding diseases. The dataset was taken from the source [9]. It is a knowledge database of disease-symptom associations generated by an automated method based on information in textual discharge summaries of patients at New York Presbyterian Hospital admitted during 2004. The first column contains the disease, the second the number of discharge summaries, and the associated symptom. Associations for the 150 most frequent diseases based on these notes were computed and the symptoms are shown ranked based on the strength of association. The method used the MedLEE natural language processing system to obtain UMLS codes for diseases and symptoms from the notes; then statistical methods based on frequencies and co-occurrences were used to obtain the associations.

Now after data cleaning and preprocessing was done, the symptoms were fed in as input data and its corresponding disease as target output. These symptoms and disease are represented by their position vector in overall symptoms list and overall disease list respectively. This is how that training process is carried out.

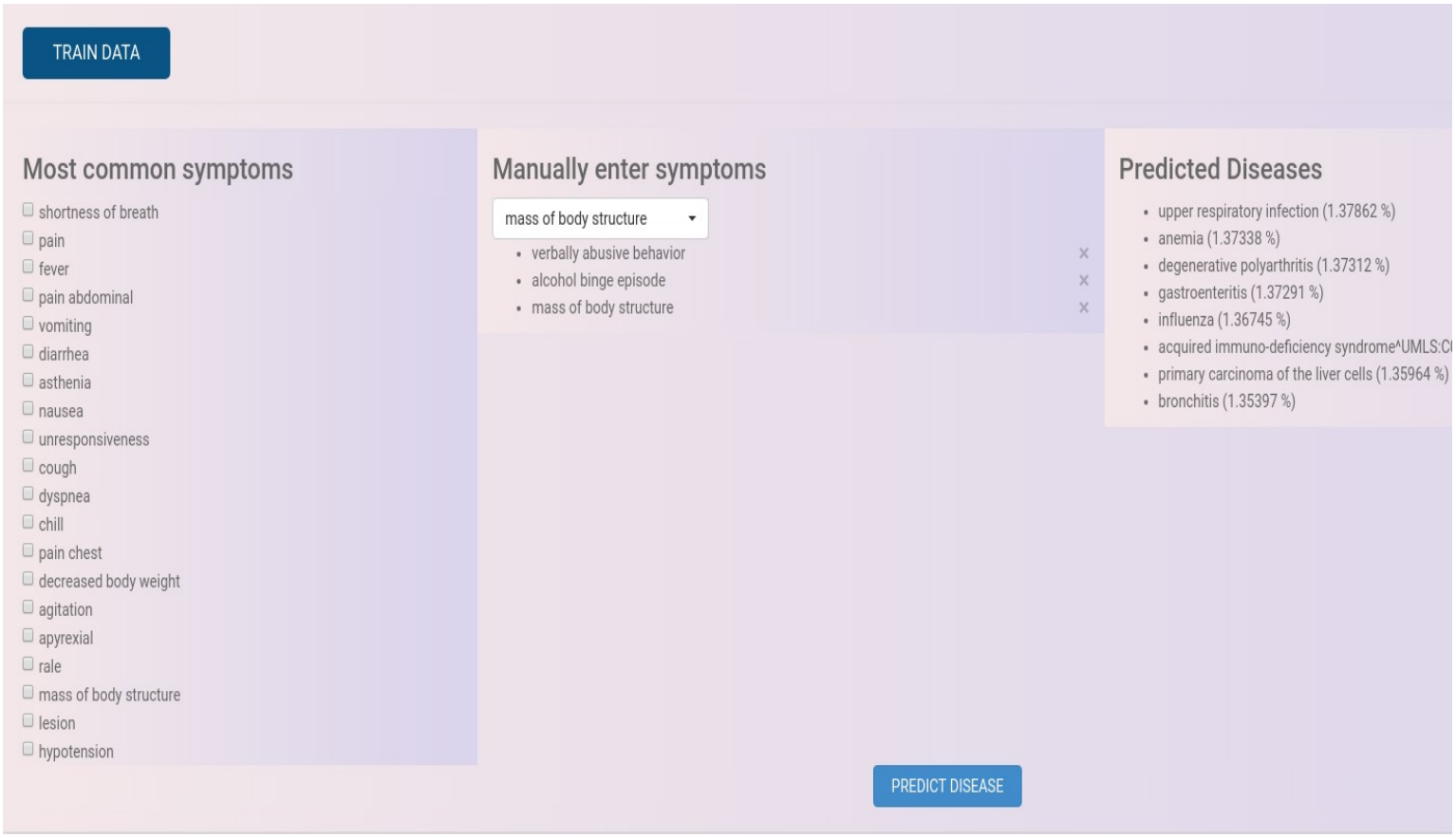

Figure 13: Disease Prediction Web Interface

5. RESULT AND DECISION

The main goal of our project is to create a Neural Network library from scratch. As per our meeting, the project development was split up into 5 phases. In the 1st phase we discussed about perceptron on how it works and different methods in creating and implementing perceptron, what functions should be used and where to obtain the training data from. After extensive research and coding we were able to create a working perceptron which was able to solve and visualize basic logic gates. In the 2nd phase we decided to create a Neural Network library based on feed-forward neural network. The testing of the NN library was conducted in phase 3rd. In this phase we tested the NN library by implementation it to solve XOR gate problem. We discuss the idea of creating a canvas which four coordinates: (0,0), (0,1), (1,0) and (1,1) act as an input for the neural network. The color at the canvas acts as an output for an input. In the beginning, output colors are random. But slowly after training NN, it starts learning and gives right output to its corresponding inputs.

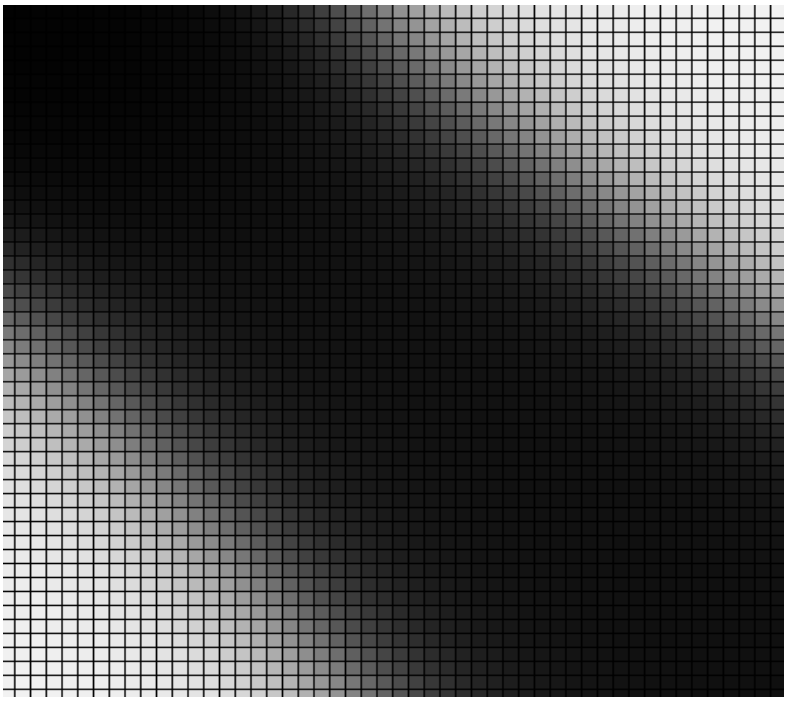

Figure 14: XOR gate Visualization

After the testing of library to solving logical gate, we wanted our library to be tested in pattern recognition. So, we used sklearn datasets to get handwritten digit. We trained our NN with 1500 these data and tested with 200 data. The result's accuracy of this testing was always around 80% which is a fair result for a Feed forward based Neural Network.

```
PS E:\Projects\neural_network_scratch\Implementation_NN> python .\5_nn_digit_recognition.py

Training -------


Testing
Accuracy %
0.8388888888888889
```

Figure 15: Handwritten-Digit-result

Finally, our last implementation of this library was to use it for disease prediction. The aim here was to do something useful with this library that could actually help people. In this phase, we discuss on how to create an interface that would be as beginner friendly as possible so that everyone can interact with the system and get their job done. So, we added 20 most frequently occurring symptoms to be easily selected. We also added a feature to search and select symptoms from our symptoms' data list so that user doesn't have to manually type their symptoms.

After the system design was completed, we tested our system on how accurate it gives the result. So, we choose a random disease from our dataset. The disease was 'decubitus ulcer' which has a symptoms systolic murmur, frail and fever. Then we fill these symptoms in our interface and hit the predict button. The test was done in four situations based on number of training. The first test was done without training, 2nd test was done with 1-time training, 3rd test was done with 2-time training and 4th test was done with 3-time training. The output of the system is shown in figure as below:

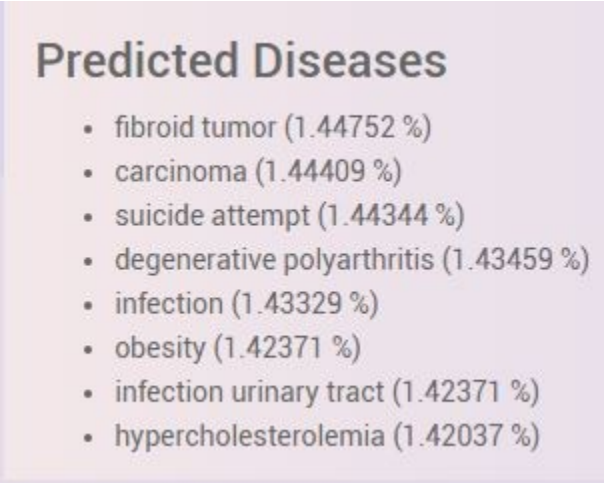

Figure 16: Without training

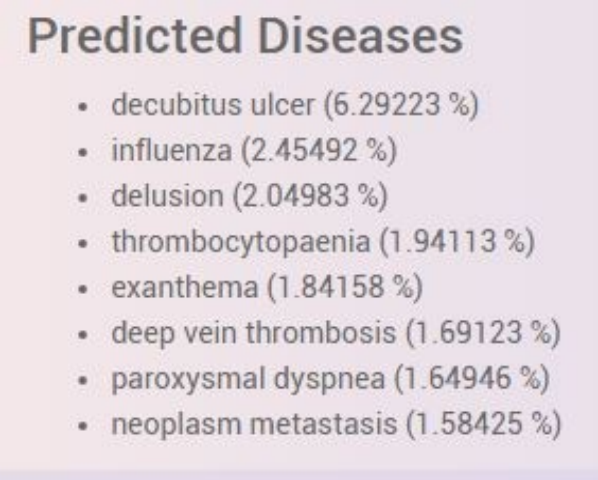

Figure 17: With 1-training

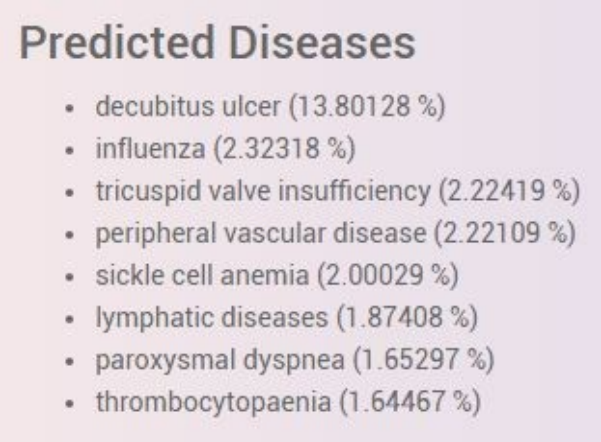

Figure 18: With 2-training

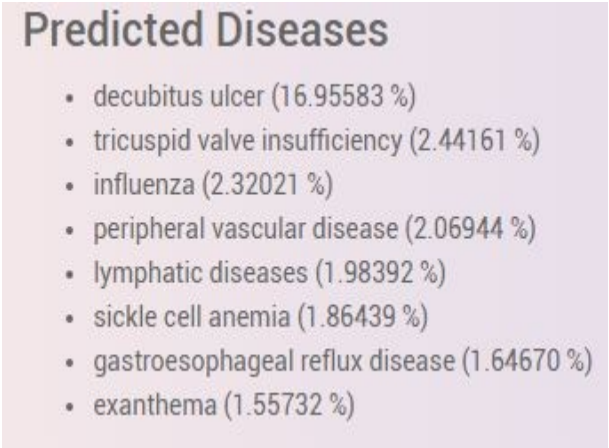

Figure 19: With 3-training

Without any training, the system randomly predicts diseases based on the initialization weights and biases value of neural network. But once the neural network is started being trained, it learns and tweak its weight and biases value according to give result of desired disease with respect to its symptoms. More amount of training result in more certainty of the prediction.

But these training amount should be limited and done at fix iteration of dataset, so that the neural network is not over fitted to those symptoms' values.

6. CONCLUSION

We successfully created a Neural network library that contains algorithm such as Feed Forward for generating output, Back Propagation using Gradient Descent for supervised learning, visualization of neural network structure and NEAT algorithm. This library is implemented by:

a. Creating a program that solve logical gate problem with visualization.
b. Creating a program that train NN with handwritten digits' datasets and allow users to draw an image that predicts digit based on its training.
c. Creating a simple game and learning agent with NN that learns to play this game.
d. Creating a web application that has interface which allow users to input symptoms and display disease probability based on the dataset it was trained upon.

7. RECOMMENDATIONS

The library and implemented application have some distinct aspects that can be improved further. This section provides some recommendations that can further enhance the usability, capability and versatility of the application.

1. Currently this neural network supports only two layers; hidden and output layer. Later, 3-layer network can also be implemented where there can be two hidden layers.

2. Right now only the feed forward network is implemented in this library. In coming days, more type of neural network such as Recurrent Neural Network (RNN), Convolution Neural Network (CNN), etc. can also be added.

3. Code optimization in both time and space complexity can be done so that the library uses less resources.

REFERENCES

[1] Perceptron (Basic Element of Neural Network) https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53 [Accessed on 24th June 2019]

[2] Neuroevolution of Augmented Topologies (NEAT) https://towardsdatascience.com/neat-an-awesome-approach-to-neuroevolution-3eca5cc7930f [Accessed on 25th June 2019]

[3] Tariq Rashid (2016). "Make Your Own Neural Network" https://www.goodreads.com/book/show/29746976-make-your-own-neural-network [Accessed on 30th July 2019]

[4] Neural Network (Two layer, Fully connected) https://github.com/nature-of-code/NOC-S17-2-Intelligence-Learning/tree/master/week4-neural-networks [Accessed on 30th July 2019]

[5] Genann: https://github.com/codeplea/genann [Accessed on 20th Aug 2019]

[6] Google's TensorFlow: https://www.guru99.com/what-is-tensorflow.html [Accessed on 22nd Aug 2019]

[7] Iterative Model: https://www.dreamstime.com/iterative-development-model-components-image121519084 [Accessed on 10th Aug 2019]

[8] Learning by Backpropagation using Gradient Descent https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html [Accessed on 13rd Aug 2019]

[9] http://people.dbmi.columbia.edu/~friedma/Projects/DiseaseSymptomKB/index.html [Accessed on 2nd Nov 2019]

APPENDIX

This appendix documents some of the most important source codes of the project.

Neural Network Library (neural_network.py)

```
#!/bin/python3
# Two-layer neural network (Fully connected)

import random
import math
import sys
from tkinter import *
import numpy as np

class bcolors:
   HEADER = '\033[95m'
   OKBLUE = '\033[94m'
   OKGREEN = '\033[92m'
   WARNING = '\033[93m'
   FAIL = '\033[91m'
   ENDC = '\033[0m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'

# Function return value between 0 and 1 for all element of matrix
def sigmoid(x):
   tmpMatrix = np.zeros(shape=(len(x),1))
   # f(x) = 1/(1+e^-x)
   for i in range(len(x)):
       # Argument is a large negative value, so it is calling exp() with a large positive value. It is very
easy to exceed floating point range that way
      # This condition solves that problem
      if (x[i][0]) < 0:
         tmpMatrix[i][0] = 1 - 1/(1+math.exp(x[i][0]))
      else:
         # Original sigmoid function
         tmpMatrix[i][0] = 1/(1+math.exp(-x[i][0]))
   return (tmpMatrix)
```

```
# Function return derivative value of sigmoid function for all element of matrix
def dSigmoid(x):
    tmpMatrix = np.zeros(shape=(len(x),1))
    # sig'(x) = sig(x)[1-sig(x)]
    for i in range(len(x)):
        tmpMatrix[i][0] = x[i][0] * (1 - x[i][0])
    return (tmpMatrix)

class NeuralNetwork:
    # Constructor
    def __init__(self, inputLen, hiddenLen, outputLen):
        self.i = inputLen # input
        self.j = hiddenLen # hidden
        self.k = outputLen # output

        self.scaleFac = 1
        self.gapVal = 200
        self.tk = Tk()
        self.tk.title("Drawing_float")
        widthSize = 800
        heightSize = 700
        self.canvas = Canvas(self.tk, width=widthSize, height=heightSize)

        # Creating weights and assiging random values
        # Weights for Hidden
        self.weightsHid = np.zeros(shape=(self.j,self.i)) # w
        for m in range(self.j):
            for n in range(self.i):
                randVal = random.uniform(-1,1)
                self.weightsHid[m][n] =  randVal

        # Weights for Output
        self.weightsOut = np.zeros(shape=(self.k,self.j)) # w'
        for m in range(self.k):
            for n in range(self.j):
                randVal = random.uniform(-1,1)
                self.weightsOut[m][n] =  randVal

        # Creating Bias and assiging random values
        # Bias for Hidden
```

```
        self.biasHid = np.zeros(shape=(self.j,1))
        for j in range(self.j):
            randVal = random.uniform(-1,1)
            self.biasHid[j][0] =  randVal

        # Bias for Output
        self.biasOut = np.zeros(shape=(self.k,1))
        for k in range(self.k):
            randVal = random.uniform(-1,1)
            self.biasOut[k][0] =  randVal

        # Algorithm that computes output based on weight and bias (similar to guess function in perceptron)
    def feedForward(self,inputs):
        self.inputs = inputs

        self.hiddens = self.weightsHid.dot(self.inputs) + self.biasHid
        self.hiddens = sigmoid(self.hiddens)

        self.outputs = self.weightsOut.dot(self.hiddens) + self.biasOut
        self.outputs = sigmoid(self.outputs)
        return self.outputs

    # Training NN using Supervised learning
    def trainSVLearing(self, inputs, targets, learningR):
        # Guess outputs from inputs
        self.feedForward(inputs)

        # Calculate output errors
        output_errors = targets - self.outputs

        # Output Gradient
        output_gradient = dSigmoid(self.outputs)

        # Delta Output weights
        # weightsOut_delta = learningR * output_errors * output_gradient · hiddens.transpose()
        # * represent hadamard product dot(·) represent matrix dot product

        tmpOut = learningR * output_errors * output_gradient
```

```
        weightsOut_delta = tmpOut.dot(self.hiddens.transpose())

        # New Output weights
        self.weightsOut += weightsOut_delta

        # new biasOut += learningR * output_errors * output_gradient
        self.biasOut+=tmpOut

        # Calculate hidden errors
        weightsOutTranspose = self.weightsOut.transpose()
        hidden_errors = weightsOutTranspose.dot(output_errors)

        # Hidden Gradient
        hidden_gradient = dSigmoid(self.hiddens)

        # Delta Hidden weights
        # weightsHid_delta = learningR * hidden_errors * hidden_gradient · inputs.transpose()
        # (*) represent hadamard product dot(·) represent matrix dot product

        tmpHid = learningR * hidden_errors * hidden_gradient

        weightsHid_delta = tmpHid.dot(self.inputs.transpose())

        # New Hiddens weights
        self.weightsHid += weightsHid_delta

        # new biasHid += learningR * hidden_errors * hidden_gradient
        self.biasHid+=tmpHid

    def nnStructure(self):
        # Best view upto 18 nodes
        frameRate = 60
        frameSpeed = int(1 / frameRate * 1000)
        widthSize = 800
        heightSize = 700
        self.canvas.pack()

        # For input layer
        inputNodes = [None] * self.i
        hiddenWLines = [None] * self.i
```

```
        y = 0
        # Starting Position
        yStart = 50 * self.scaleFac
        for m in range(self.i):
            # If object is out of canvas, small scale factor
            if(y >= heightSize):
                self.scaleFac = self.scaleFac * 0.999
                self.gapVal -= 0.8
                self.canvas.delete("all")
                return self.nnStructure()

            x1, y1 = 50/self.scaleFac, yStart+y
            x2, y2 = 90/self.scaleFac, yStart+y+40
             inputNodes[m] = self.canvas.create_oval(x1*self.scaleFac, y1*self.scaleFac, x2*self.scaleFac,
y2*self.scaleFac, fill="white")

            # For hidden weight line
            xic, yic = (x1+x2)/2, (y1+y2)/2
            yTmp = 0
            yStartTmp = 50 * self.scaleFac
            for n in range(self.j):
                x1, y1 = 350/self.scaleFac, yStartTmp+yTmp
                x2, y2 = 390/self.scaleFac, yStartTmp+yTmp+40
                xhc, yhc = (x1+x2)/2, (y1+y2)/2
                weight = self.weightsHid[n][m]
                if(weight > 0):
                    color = "red";
                else:
                    color = "green";
                                hiddenWLines = self.canvas.create_line(xic*self.scaleFac, yic*self.scaleFac,
xhc*self.scaleFac, yhc*self.scaleFac, fill=color, width=3)
                # GapsTmp
                yTmp = yTmp + self.gapVal
            # Gaps
            y = y + self.gapVal

        # For Hidden layer
        hiddenNodes = [None] * self.j
        y = 0
        yStart = 50 * self.scaleFac
```

```
        for m in range(self.j):
            # If object is out of canvas, small scale factor
            if(y >= heightSize):
                self.scaleFac = self.scaleFac * 0.999
                self.gapVal -= 0.8
                self.canvas.delete("all")
                return self.nnStructure()

            x1, y1 = 350/self.scaleFac, yStart+y
            x2, y2 = 390/self.scaleFac, yStart+y+40
                    hiddenNodes[m] = self.canvas.create_oval(x1* self.scaleFac, y1* self.scaleFac, x2*
self.scaleFac, y2* self.scaleFac, fill="white")

            # For output weight line
            xhc, yhc = (x1+x2)/2, (y1+y2)/2
            yTmp = 0
            yStartTmp = 50* self.scaleFac
            for n in range(self.k):
                x1, y1 = 650/self.scaleFac, yStartTmp+yTmp
                x2, y2 = 690/self.scaleFac, yStartTmp+yTmp+40
                xoc, yoc = (x1+x2)/2, (y1+y2)/2
                weight = self.weightsOut[n][m]
                if(weight > 0):
                    color = "red";
                else:
                    color = "green";
                        hiddenWLines = self.canvas.create_line(xhc* self.scaleFac, yhc* self.scaleFac, xoc*
self.scaleFac, yoc* self.scaleFac, fill=color, width=3)
                # GapsTmp
                yTmp = yTmp  + self.gapVal

            # Gaps
            y = y  + self.gapVal

        # For Output layer
        outputNodes = [None] * self.k
        y = 0
        yStart = 50* self.scaleFac
        for m in range(self.k):
            # If object is out of canvas, small scale factor
```

```
        if(y >= heightSize):
            self.scaleFac = self.scaleFac * 0.999
            self.gapVal -= 0.8
            self.canvas.delete("all")
            return self.nnStructure()

        x1, y1 = 650/self.scaleFac, yStart+y
        x2, y2 = 690/self.scaleFac, yStart+y+40
                outputNodes[m] = self.canvas.create_oval(x1* self.scaleFac, y1* self.scaleFac, x2*
self.scaleFac, y2* self.scaleFac, fill="white")
        # Gaps
        y = y  + self.gapVal

    # while True:
    self.tk.after(frameSpeed, self.tk.update())
    return (self.tk)

  def trainSVLearingVisualization(self,inputs,targets,learningRate):
    # Best view upto 18 nodes
    self.canvas.pack()
    widthSize = 800
    heightSize = 700
    frameRate = 60
    frameSpeed = int(1 / frameRate * 1000)

    # For input layer
    inputNodes = [None] * self.i
    hiddenWLines = [None] * self.i
    y = 0
    # Starting Position
    yStart = 50 * self.scaleFac
    for m in range(self.i):
        # If object is out of canvas, small scale factor
        if(y >= heightSize):
            self.scaleFac = self.scaleFac * 0.999
            self.gapVal -= 0.8
            self.canvas.delete("all")
            return self.trainSVLearingVisualization(inputs,targets,learningRate)

        x1, y1 = 50/self.scaleFac, yStart+y
```

```
        x2, y2 = 90/self.scaleFac, yStart+y+40
          inputNodes[m] = self.canvas.create_oval(x1*self.scaleFac, y1*self.scaleFac, x2*self.scaleFac,
y2*self.scaleFac, fill="white")

        # For hidden weight line
        xic, yic = (x1+x2)/2, (y1+y2)/2
        yTmp = 0
        yStartTmp = 50 * self.scaleFac
        for n in range(self.j):
           x1, y1 = 350/self.scaleFac, yStartTmp+yTmp
           x2, y2 = 390/self.scaleFac, yStartTmp+yTmp+40
           xhc, yhc = (x1+x2)/2, (y1+y2)/2
           weight = self.weightsHid[n][m]
           if(weight > 0):
              color = "red";
           else:
              color = "green";

                          hiddenWLines = self.canvas.create_line(xic*self.scaleFac, yic*self.scaleFac,
xhc*self.scaleFac, yhc*self.scaleFac, fill=color, width=3)
           # GapsTmp
           yTmp = yTmp  + self.gapVal

        # Gaps
        y = y + self.gapVal

     # For Hidden layer
     hiddenNodes = [None] * self.j
     y = 0
     yStart = 50 * self.scaleFac
     for m in range(self.j):
        # If object is out of canvas, small scale factor
        if(y >= heightSize):
           self.scaleFac = self.scaleFac * 0.999
           self.gapVal -= 0.8
           self.canvas.delete("all")
           return self.trainSVLearingVisualization(inputs,targets,learningRate)

        x1, y1 = 350/self.scaleFac, yStart+y
        x2, y2 = 390/self.scaleFac, yStart+y+40
```

```
                hiddenNodes[m] = self.canvas.create_oval(x1* self.scaleFac, y1* self.scaleFac, x2* self.scaleFac, y2* self.scaleFac, fill="white")

        # For output weight line
        xhc, yhc = (x1+x2)/2, (y1+y2)/2
        yTmp = 0
        yStartTmp = 50* self.scaleFac
        for n in range(self.k):
            x1, y1 = 650/self.scaleFac, yStartTmp+yTmp
            x2, y2 = 690/self.scaleFac, yStartTmp+yTmp+40
            xoc, yoc = (x1+x2)/2, (y1+y2)/2
            weight = self.weightsOut[n][m]
            if(weight > 0):
                color = "red";
            else:
                color = "green";
                    hiddenWLines = self.canvas.create_line(xhc* self.scaleFac, yhc* self.scaleFac, xoc* self.scaleFac, yoc* self.scaleFac, fill=color, width=3)
            # GapsTmp
            yTmp = yTmp  + self.gapVal

        # Gaps
        y = y  + self.gapVal

    # For Output layer
    outputNodes = [None] * self.k
    y = 0
    yStart = 50* self.scaleFac
    for m in range(self.k):
        # If object is out of canvas, small scale factor
        if(y >= heightSize):
            self.scaleFac = self.scaleFac * 0.999
            self.gapVal -= 0.8
            self.canvas.delete("all")
            return self.trainSVLearingVisualization(inputs,targets,learningRate)
        x1, y1 = 650/self.scaleFac, yStart+y
        x2, y2 = 690/self.scaleFac, yStart+y+40
                outputNodes[m] = self.canvas.create_oval(x1* self.scaleFac, y1* self.scaleFac, x2* self.scaleFac, y2* self.scaleFac, fill="white")
        # Gaps
```

```
        y = y  + self.gapVal
    self.tk.after(frameSpeed, self.tk.update())
    self.trainSVLearing(inputs,targets,learningRate)
    return (self.tk)

# For Neuro Evolution
def copy(self):
    return (self)

def mutate(self, mutationRate):

    def mutateElement(val):
        if (random.uniform(0,1) < mutationRate):
            # x = (random.uniform(0,100))
            x = 2 * random.uniform(0,1) - 1
            return x
        else:
            return val

    vfunc = np.vectorize(mutateElement)
    self.weightsHid = vfunc(self.weightsHid)
    self.weightsOut = vfunc(self.weightsOut)
    self.biasHid = vfunc(self.biasHid)
    self.biasOut = vfunc(self.biasOut)
```

Disease Prediction Script file (script.py)

```
import pandas as pd
import xlrd
from neural_network import NeuralNetwork
import numpy as np
from heapq import nlargest

# Initialization
nn = NeuralNetwork(402, 50, 134)
learningRate = 0.1
df = pd.read_excel('data.xlsx')

# Data preparation
disease_list = df.Disease.to_list()
# Removing spaces created from excel
while "\xa0" in disease_list: disease_list.remove("\xa0")

occurance_list = df.Count_of_Disease_Occurrence.to_list()
# Removing spaces created from excel
while "\xa0" in occurance_list: occurance_list.remove("\xa0")

symptom_list = df.Symptom.to_list()
# Remove duplicates
symptom_list_filter = list(set(symptom_list))

#-- Prepare Input and target vector and feed it to NN for training --#
def training():
   print("Training Data Started")

   # No of times data set is training to NN
   loop = 20
   for l in range(loop):
      # All inputs and target except last one
      print(str(int(l/loop*100))+"% completed")
      tmpInput = df.Disease.to_list()
      tmpInd = 0
      for i in tmpInput:
         if( i!= "\xa0"):
            inputVec = np.zeros(shape=(402,1))
```

```
            targetVec = np.zeros(shape=(134,1))
            ind = tmpInput.index(i)
            if (ind != 0):
               inputList = symptom_list[tmpInd:ind]
               for k in inputList:
                  vecInd = symptom_list_filter.index(k)
                  inputVec[vecInd] += 1
               targetVec[disease_list.index(i)-1] = 1
               nn.trainSVLearing(inputVec,targetVec,learningRate)
               tmpInd = ind

      # Last Inputs and Target
      inputVec = np.zeros(shape=(402,1))
      targetVec = np.zeros(shape=(134,1))
      j = len(symptom_list)
      inputList = symptom_list[tmpInd:j]
      for k in inputList:
         vecInd = symptom_list_filter.index(k)
         inputVec[vecInd] += 1
      targetVec[-1:] = 1
      nn.trainSVLearing(inputVec,targetVec,learningRate)
   print("------- TRAINING COMPLETE -------")

def getData():
   # Function that generates common symptoms and pass it as well
   # all the symptoms from data set
   from collections import Counter
   c = Counter(symptom_list)

   # Select only first 20
   words = c.most_common(20)
   top_symptom = []
   for i in words:
      top_symptom.append(i[0])
   return(top_symptom,symptom_list_filter)

def predict(checked_list):
   inputVec = np.zeros(shape=(402,1))
   for item in checked_list:
      if(item!=""):
```

```
        ind = symptom_list_filter.index(item)
        inputVec[ind]=1
result = nn.feedForward(inputVec)
result_norm = [float(i)/sum(result) for i in result]
top_prediction = nlargest(8, enumerate(result_norm), key=lambda x: x[1])
predicted_values = [[0 for x in range(8)] for y in range(0)]
for i in top_prediction:
    tmp = disease_list[i[0]]
    dis = tmp.split('_', 1)[-1]
    percent = float(i[1])*100
    per = "%.5f" % percent + " %"
    predicted_values.append([dis,per])
return predicted_values
```