Contents

Analysis 2

Investigation 2

Research 2

Neural Networks 3

Training a neural network 8

My investigation 12

Objectives 13

Main class diagram 15

IPSO 19

Documented design 21

Introduction 21

Packages 21

Server 21

Website diagram 22

Systems flowchart 23

Obtaining the dataset 27

Reading the Data 27

Formatting the data 29

Vectors 30

The neural network class 31

File structure 37

Algorithms 39

User Interface 48

Server 63

Testing 68

Conclusion 81

Evaluation 85

objectives 85

Improvements 87

Refrences: 87

Analysis

Investigation

For my project I decided that I would research and investigate the application of Artificial Intelligence in computer systems today. In particular, I chose to investigate machine learning, which is a branch of Artificial Intelligence. By the end of my research I wanted to find out what machine learning is already being used for and exactly how machine learning is being implemented so that I could implement a model myself. Recently, I was told by a friend of mine that their younger sibling was unable to/having difficulties learning to write, I wanted to investigate whether machine learning; image recognition in particular could be used to assist my friend's sibling in learning how to write. An example of this could be the implementation of a image recognition model in the backend of a handwriting tool, allowing the handwriting application to recognise and label a user's handwriting.

Research

What is machine learning

Artificial intelligence within computers is the ability for a computer to carry out tasks that ordinarily would require a human to do, and just like a human, an intelligent computer should be able to learn from experience and improve its ability to carry out certain tasks over time. Machine learning is an application of artificial intelligence, a computer is given data and is allowed to "learn" by recognising various patterns within the data. By doing this the computer is able to improve at its task.

How is machine learning implemented?

There are two types of learning that a system implemented with machine learning can undergo: supervised learning and unsupervised learning.

Supervised learning

In supervised learning a known input with a desired output is given to the machine learning model, the data is passed through the model and depending on the output (compared to the desired output) the model is tuned so that the output is closer to what is desired. This is repeated with large amounts of input, and as a result the model should be able to make a prediction on an unknown input.

Depending on the type of data you have or would like to predict, a classification technique or regression technique may be used when creating a model.

Classification techniques are used for discrete predictions. An example of this may be to predict whether a tumour is cancerous or benign or whether an image is a cat or dog. Overall, this is used for when your data can be split into any number of different categories or labels. Algorithms for these types of models include: support vector machines, bagged decision trees, Naïve Bayes and neural networks.

Regression techniques are used to predict continuous data. For example, this is applied to temperature and forecast and the demand for electricity in a specific area. Some algorithms used in regression are: linear model, non-linear model, stepwise regression.

Unsupervised learning

In unsupervised learning a model is able to find patterns in large amounts of data that a human would not be able to see. These patterns are then used to create conclusions to provide solutions.

The most common method of implementing unsupervised learning is by using clustering. This is when the data provided to the model is split into much smaller groups called clusters, each entity within a cluster contains more similar attributes to entities within its own cluster when compared to an entity within another cluster. This is often used in retail companies where the company would like to group their customers based on purchasing habits. Some of the most used clustering algorithms include: k-means, hierarchical clustering, hidden Markov models.

My project

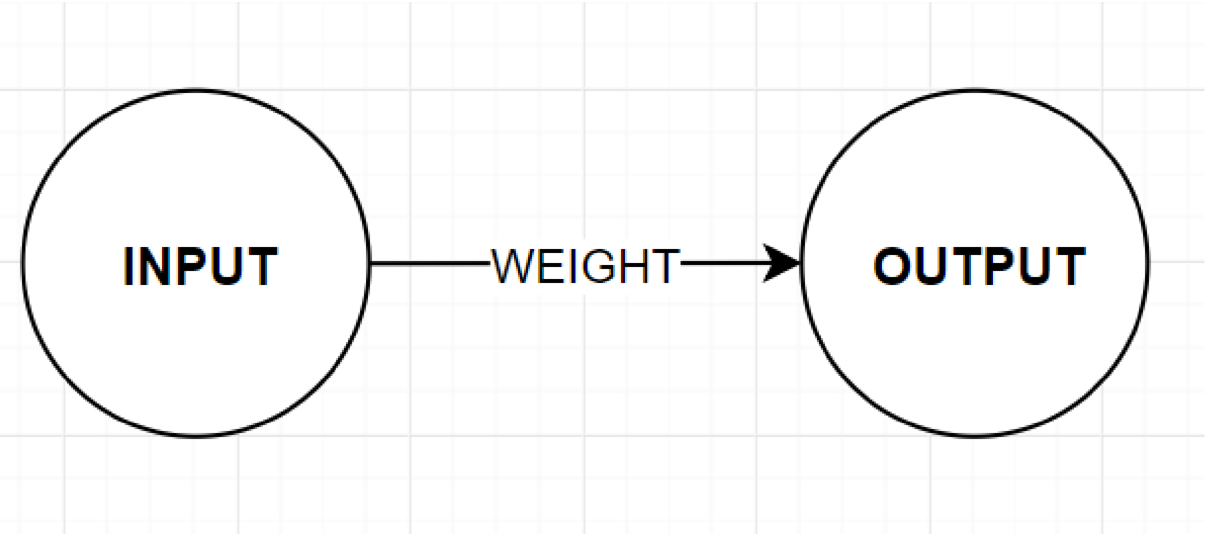
Due to time constraints I will only be able to investigate a single machine learning technique in detail. I chose to look at neural networks, this is because neural networks are largely used in image recognition which is an area that I am interested in. The reason for my interest in the field of image recognition is because a friend of mine recently told me that their younger brother was having difficulties in learning how to write the letters. The Montessori method of teaching is becoming more popular around the world. In this method of learning children are made to draw a letter correctly multiple times to assist in establishing the motor functions for them to be able to write letters. I wanted to find out if it would be possible to use machine learning to automate teaching a child to correctly write each letter of the alphabet in this way. This would be implemented by creating an application with a neural network in the backend which is able to recognise the child's drawing. For my project I will create a tool to discover whether neural networks are a viable method of assisting children is learning how to write by being able to reliably recognise handwritten characters.

Neural Networks

What is a neural network?

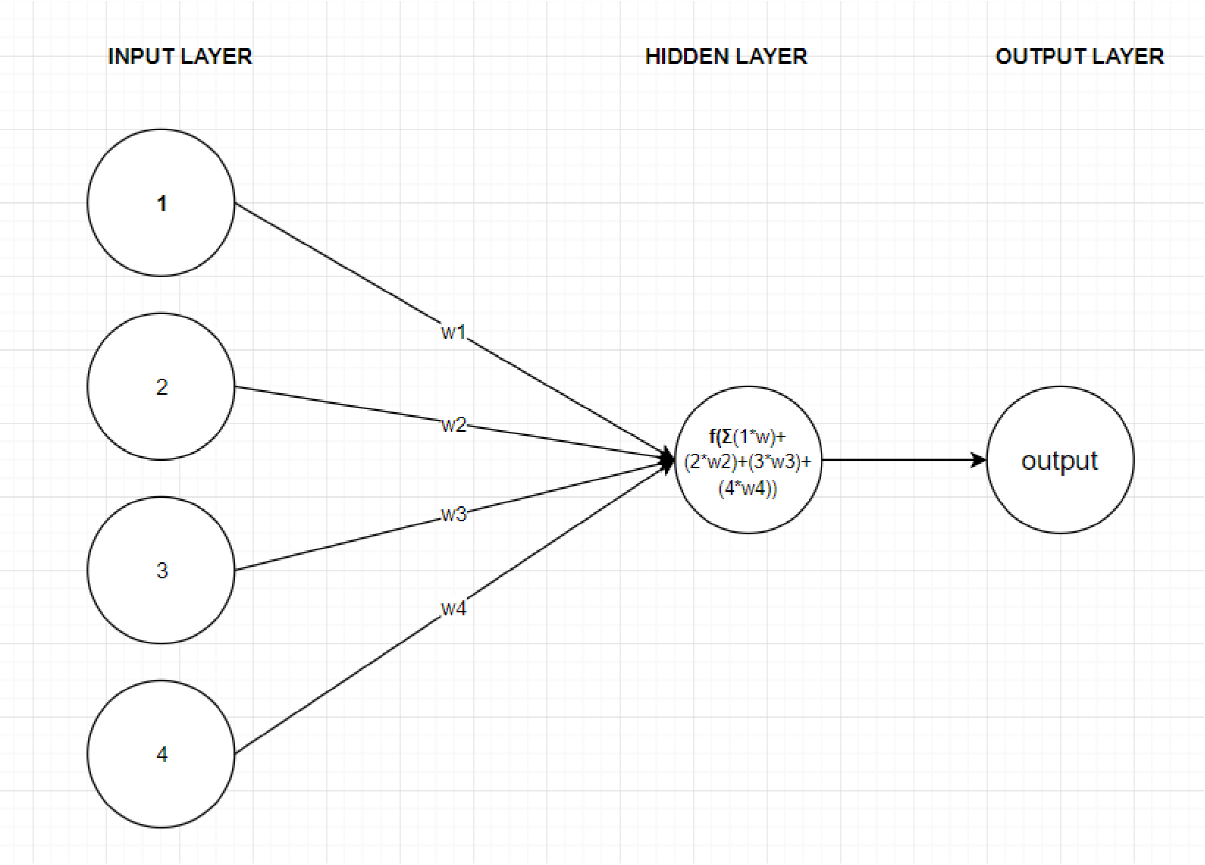
Or artificial neural networks are a common application of artificial intelligence where the implemented model mimics the human brain. The concept behind this method is to abstract the methods of the human brain to create a framework that is able to learn in a similar fashion to the human brain. In the brain there is a network of neurons, each neuron is connected to another by a synapse. Data is sent through our brains by sending electrical impulses along these neurons and synapses. The collection of over 100 billion neurons is what allows us as humans to learn. An artificial neural network replicates this by creating a graph of nodes each connected by weights. The electrical impulses are replaced by mathematical operation and as a result an artificial neural network is able learn.

How do they work



Above is an example of the simplest possible neural network – an input neuron is connected to an output neuron by a weight. In this network the value of the output will be the (input * weight), the output value can be altered by altering the value of the weight. However, this type of network is not useful as there is very little complexity, therefore a network will require a much larger number of neurons and weights.

A network can be described in terms of its “layers”. A layer is a collection of parallel neurons. In addition to this there are three types of layers, an input layer, hidden layer and an output layer. Every network must have an input layer and an output layer but may have one or more hidden layers (a hidden layer is a collection of parallel neurons between the input and output layers). The number of hidden layers and neurons in each layer can be decided by the creator of the model.

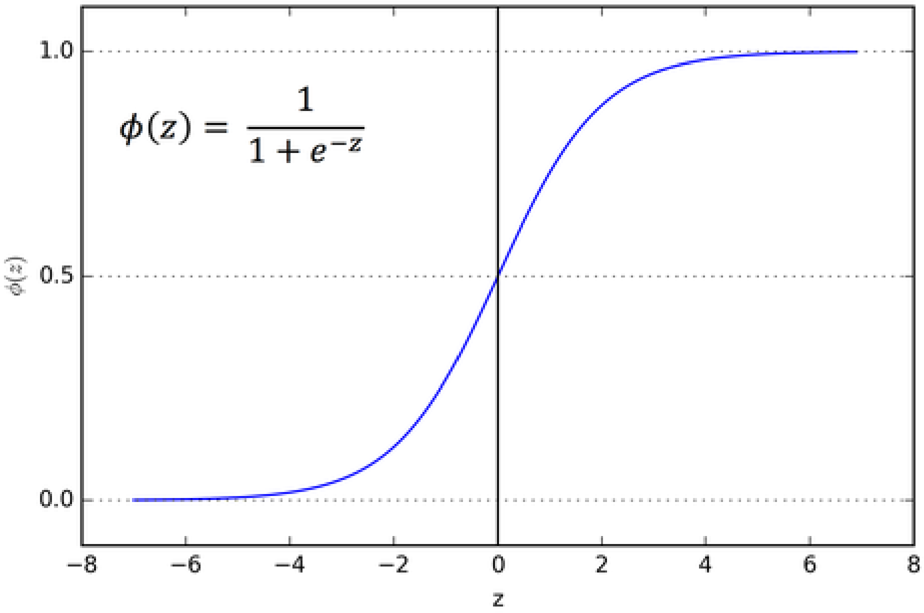


Above is an example of a perceptron. This is a neural network with a single hidden node (neuron) and a single output node – this means it is a binary classifier because it can only return one value. In this network there are four input nodes assigned with the values 1, 2, 3, 4. The way that this input is passed through the network is by summed weights. This means that each input node value is multiplied by the weight connecting it to the next node. In this case, the value of the hidden node is the sum of all the connected nodes multiplied by the weights connecting them. In addition to this, a function is passed over this value. This is known as the activation function. A perceptron (like the one above) is only able to solve linear problems due to the simplicity of the network and the single neuron in the hidden layer. A network with a larger number of hidden neurons and hidden layers is able to solve non-linear problems as a result of the increased complexity of the network.

Activation functions

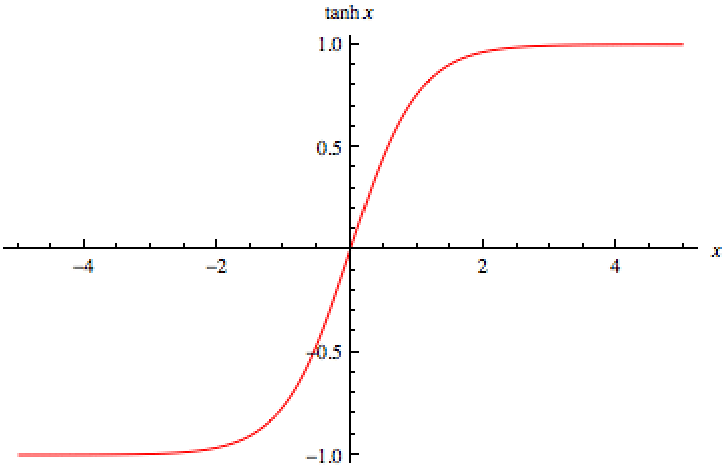
An activation function is a mathematical function that is passed over the value of each hidden and output nodes of a network. These functions are important because they decide how much impact each node will have on the network. In addition to this the activation function used must be a non linear function, this is because it is required to maintain non linearity within the network. Three of the most used activation functions are sigmoid, tanh and relu.

Sigmoid



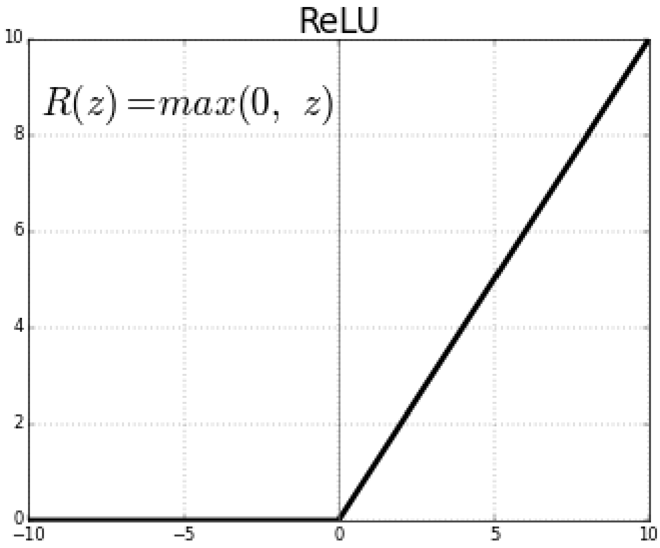
This function “squashes” a value to between 1 and 0. With larger numbers being closer to one and smaller numbers closer to 0. This function is useful because it provides a value between 0 and 1 which can represent probability. However due to its small return value it can take a long time for a network to learn using this function and can also result in the vanishing gradient problem where a network is unable to learn due to the gradient of the function being too small.

Tanh



Using tanh as an activation function returns a value between 1 and -1 – similar to the sigmoid function. Using the tanh function will allow the network to train faster because the gradient of the function will often be larger than sigmoid.

Relu



The rectified linear unit (relu) function return 0 if the input value is less than 0 and will the return the input if the input is greater than 0. This will allow the network to train extremely fast due to the large gradient. However, it may "kill" nodes due to it returning 0 for values less than 0 causing any multiplication by the node to be 0.

I will be implementing all three of these functions and will be able to analyse the differences between them.

Training a neural network

Initially all the weights in a network are set to a random number and then there are three steps to training a neural network.

Forward propagation

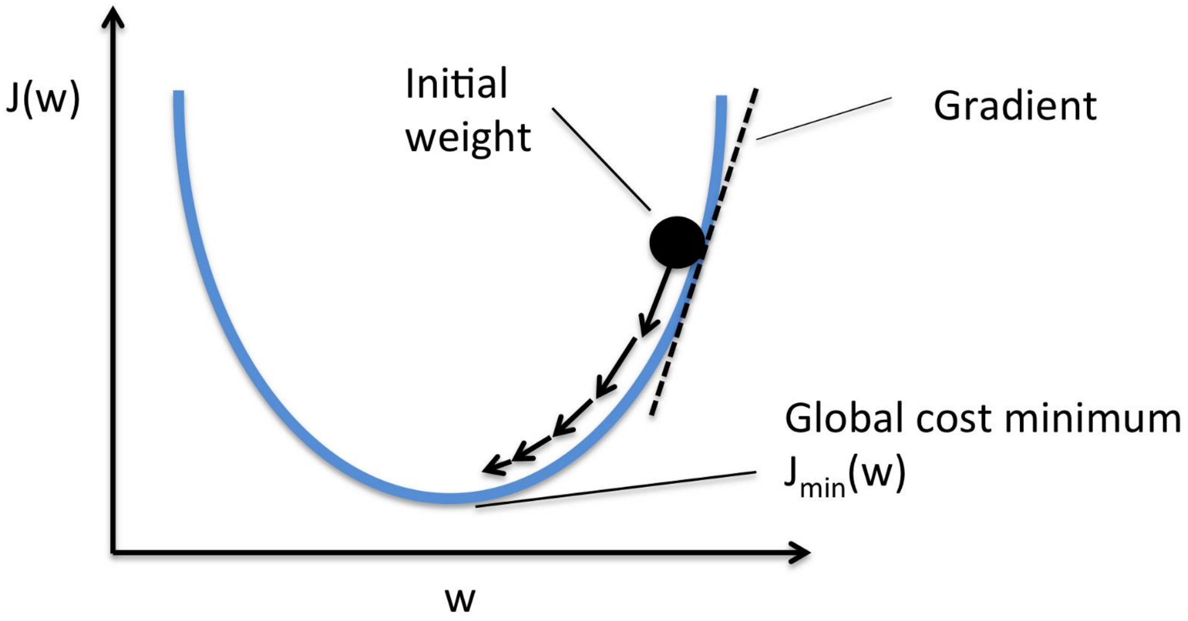
Using supervised learning, forward propagation means to simply means to pass a data sample through the network using the current initialised weights.

Cost calculation

This is a calculation to determine how far the current output of the network is from the desired output. A simple calculation can be: desired output – actual output.

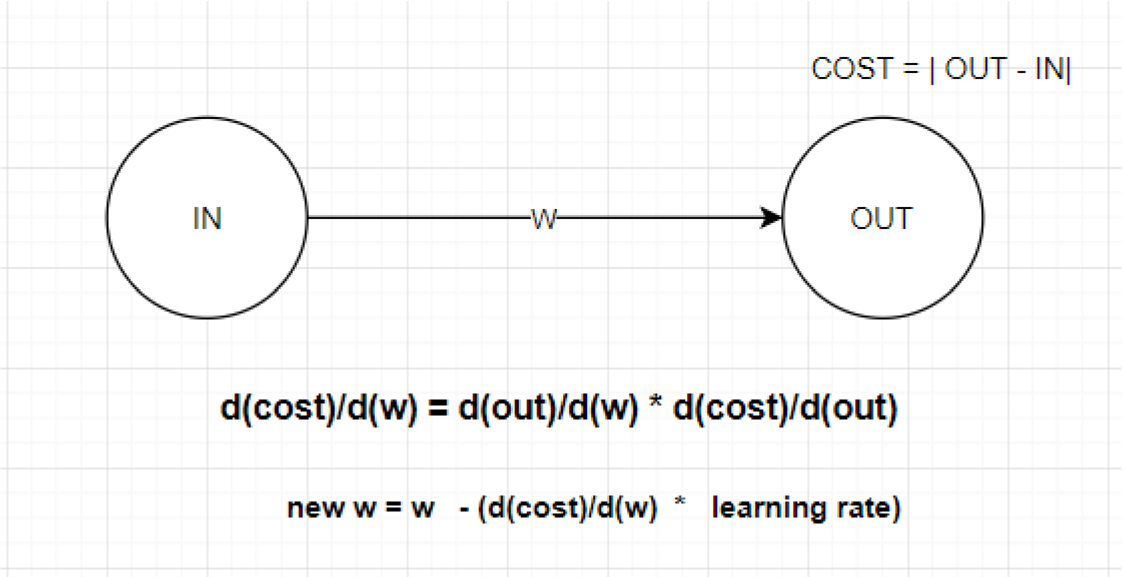
Back propagation

This is the most crucial part of training a network. This is how the network is able to learn and improve its accuracy. The objective of back propagation is to minimise the cost by tuning the weights responsible. This is done by using a method known as gradient descent.

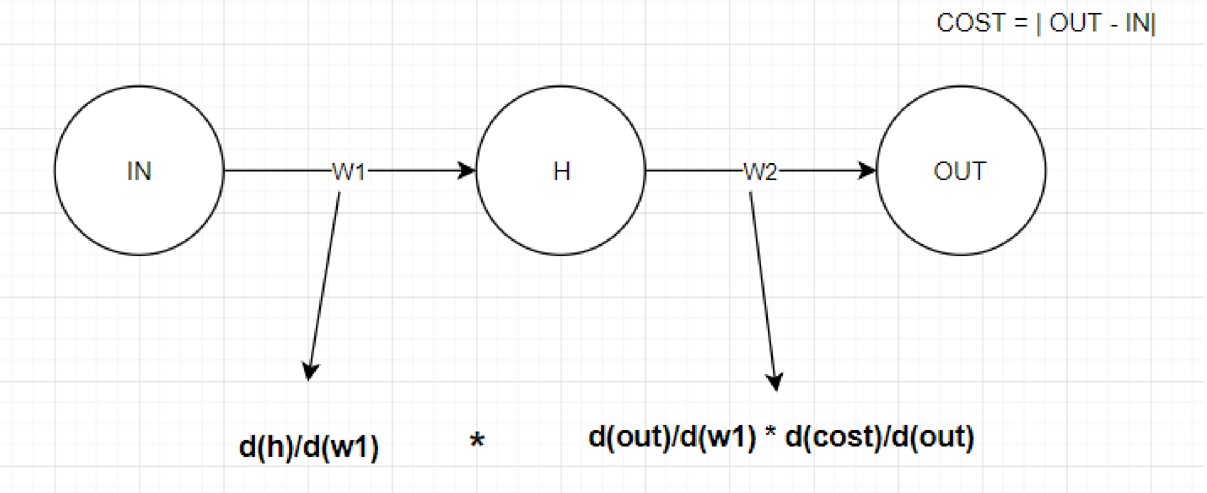


In the graph above a graph is drawn of the cost the weight is responsible for against the value of the weight. The idea is to initialise the weight value so that the cost of the weight is at its lowest possible value. This is done by finding the gradient and then by reducing the weight gradient until the gradient is at the most minimum point. To find how much of the error the specific weight is responsible for you can use the chain rule and differentiation by applying the back-propagation formula.

Once a sample has been passed through the network and an error or cost has been calculated you can apply the back-propagation formula to determine how responsible each weight is for the error.


COST = | OUT - IN|
IN
W
OUT
d(cost)/d(w) = d(out)/d(w) * d(cost)/d(out)
new w = w - (d(cost)/d(w) * learning rate)

Above is a very simple network with a single weight (w). To update the value of the weight to minimise the cost of the network, you must find the current gradient of the cost against the weight. Once that is calculated using the chain rule, the weight value is decreased (to decrease the gradient) by the learning rate multiplied the calculated gradient.

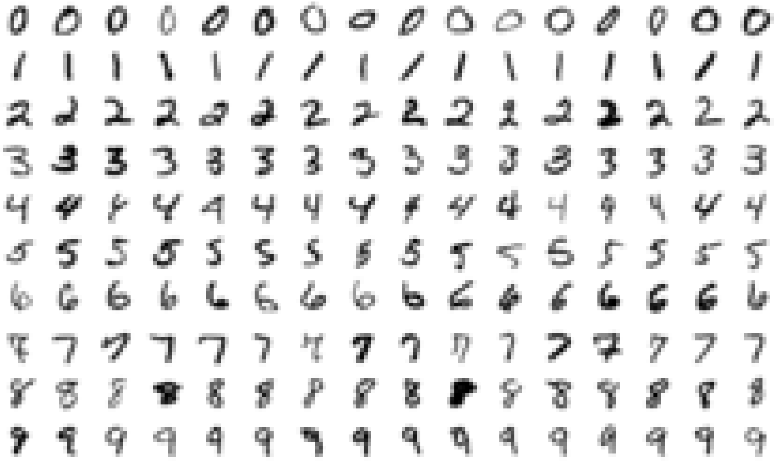


This diagram shows the gradients of the cost against each weight in a multiple layer network. For w1 the gradient is simply the cost of the weight ahead multiplied by derivative of the hidden node with respect to w1.

To train a network, this process of back propagation is repeated for a large number of samples known as a data set. Each time the weights re-update the network becomes more accurate. You may iterate over a sample more than once; 1 iteration is known as an epoch.

Neural networks with image recognition

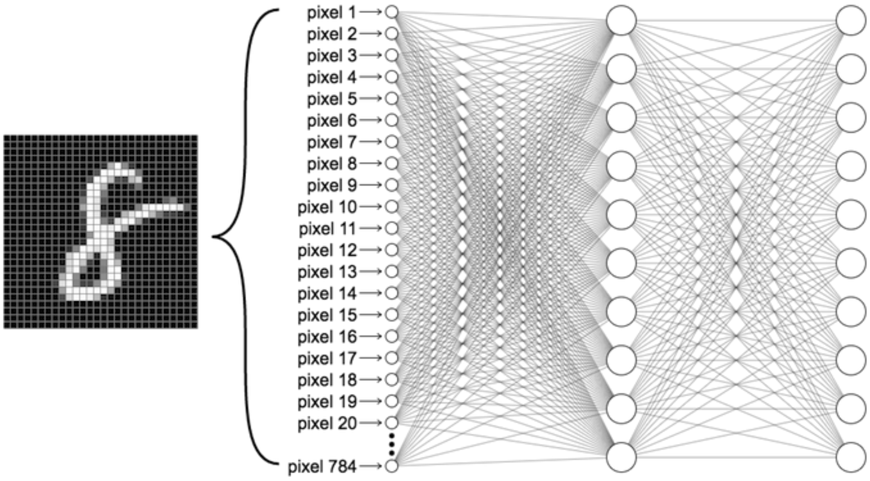
After researching how a neural network fundamentally works, I wanted to research how they are used for image recognition. While doing this I discovered the MNIST handwritten digits dataset. This dataset contains 70,000 images of handwritten digits (0-9), 60,000 used for training a network and 10,000 used for testing the network.



Each image within the dataset is normalised so that they are a 28pixel * 28pixel image, with each pixel being assigned a number between 0-255 to represent the grey level of each pixel.

This dataset can be acquired from this source: http://yann.lecun.com/exdb/mnist/. The data is split into four different files training images, training labels, testing images and testing labels. Each file is compressed using GNU zip compression algorithm resulting in the files having a ".gz" extension. There are modules in python which would allow me to directly read the files into python variables which would be extremely convenient. In addition, the source contains detailed documentation on the various algorithms applied to each sample to data to normalise the data. This information will be essential for my program.

I found that a neural network built for this data set must have a total of 784 input nodes (one for each pixel value) any number of hidden nodes and layers, and 10 output nodes – one for each digit. By training the network with the 60,000 images the network is able to predict an unlabelled digit that is inputted into the network, this is one of the most common forms of image recognition.



This image shows an example of what the neural network architecture may look like.

My investigation

After researching how a neural network works it was clear to me that there are certain factors of a network that will affect the prediction accuracy of the model and the time taken to train the network. These factors are the following:

-The learning rate of the network

-The number of hidden layers/hidden nodes

-The activation function used in the network

-The number of epochs

-The size of the training data (the number of training samples)

I wanted to explore in detail the affects that each of these factors had on the final accuracy of a neural network and the total time taken for the network to be trained.

To do this I decided to create a tool that would allow me to quickly create and train neural networks while allowing me to also easily change the factors that I described above. This tool would also store the features of each trained neural network and allow me to visualise results for each network in a manner whereby I could draw conclusions. For example, the learning rate, hidden layers/nodes, activation function, number of epochs, the size of the training data, the time taken to train and accuracy of the network would all be recorded and I would be able to produce a graph of which ever two variables I chose. I would only use the MNIST dataset as a control variable so that all the other variables will be fairly comparable i.e the same 10,000 images from the MNIST dataset will be used to test every network.

To create this tool, I researched some options that would be suitable. I decided to use python to create the neural network class of my algorithm, this is because python contains a module known as NumPy. This module is perfect for neural networks because it contains many matrix methods which are essential for applying neural network mathematics – some of these examples include the "dot" function which returns the dot product of two matrices is essentially how two matrices are multiplied. For the interface of my program I researched using PyQt and tkinter because they are also python modules, this would make it easier to integrate all aspects of my program. However, as I did not have much time to create my program, I chose to utilise my previous knowledge of web development to create a server where the interface will be a simple website and have python running as the backend. For this, I chose to use flask, which is a minimal web framework. My reason for choosing flask is because it is a simple and lightweight framework which runs on python, since the majority of my program will be implemented using python, flask would provide a seamless integration of a webpage onto my program. In addition, because of flask's lightweight nature it is easy to quickly create a web application with python code running in the backend.

Objectives

My program should:

- Allow a user to input:
    - Number of hidden layers
    - Number of hidden nodes per layer
    - Activation function to use
    - Number of epochs
    - Number of training samples (between 0 – 60,000)

^A html file will be created to provide an interface for the user to input values into fields. These values will then be sent to the python server.

- Create and a train a neural network using the user provided attributes

^The user values will be used in python to create an instance of the neural network class and the train method will be called to train the network.

- Output the accuracy and time taken to train network

^Once the network is trained the evaluate method will be called and the accuracy of the network with the time taken will be passed from the server onto a html file for the user to see.

- Store the neural networks attributes and the resulting accuracy and time to train in a text file

^This should be done on the python server immediately and automatically after outputting the html file with the networks results.

- Allow the user to select two different attributes

^A html file should be outputted containing two sets of variable options (one for the x axis) and one for the y axis) the user should be able to select one of these options from each set and this selection will be sent to the server.

- Use stored information to produce a scatter graph of the selected attributes

^Using python the server should call a function which searches through all the stored neural network data and produces a list of the selected variables. These should be translated to points on a scatter graph. The graph should be outputted back onto a html file for the user to analyse.

- Allow users to select one of the stored networks

^Display each of the networks and their attributes from the text file onto a html page where the user is able to select a single network from a list of all the networks. This choice should be sent to the server.

- Allow user to input their own image to test selected network

^Using a blank html file, the program should allow the user to hand draw an image. This image pixel data should be sent to the server where it is correctly formatted and called as a parameter for the neural network run method (for the selected network from the previous point) and then the output of this method is passed from the server back to the html file and to the user

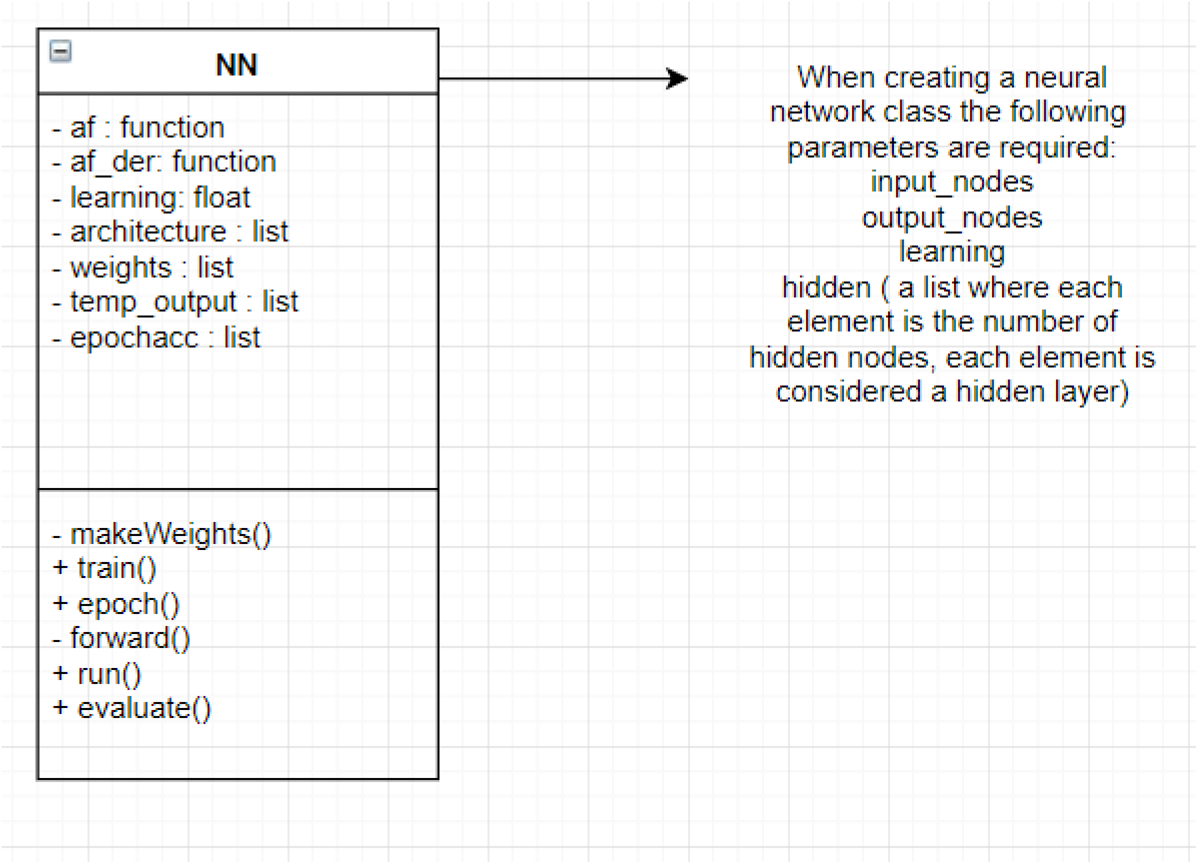
- Ultimately the user should be able to come to a conclusion about what the optimum variables are to create a neural network which maximises accuracy. Whether a neural network is suitable in assisting children in learning to write.

- The resulting tool should be able to be used to optimise a neural network for any problem that involves the use of supervised learning.

- Using the results produced by this neural network tool I will create an image recognition tool that utilises the optimal neural network attributes, to maximise the accuracy of the network.

Main class diagram

This is the class diagram for the neural network class:



When creating a neural network class the following parameters are required:
input_nodes
output_nodes
learning
hidden (a list where each element is the number of hidden nodes, each element is considered a hidden layer)

In this class the “af” variable stands for activation function. It is created with the object and is assigned a function (for example it could be assigned sigmoid which would be a predefined function outside the neural network class). The “af_der” is the same as “af” but is assigned the derivative function of the activation function. The learning rate is assigned to the “learning” variable as a float value (because it will be used in calculations). The “architecture” variable is a list of the number of nodes, where each index is the layer number (the first index is the input layer and the value of the index is the number of nodes in the layer). The “weights” variable is where all the weights between each layer will be stored. The “temp_output” variable is responsible for storing the value calculated at each layer when a sample of data is passed through the network. Lastly the “epochacc” standing for epoch accuracy will store the accuracy of the network after each epoch. All of these variables are private because they are only used within the class by the different methods and are not intended to be used outside the class.

I will now provide the pseudo code for each of the methods within the class;

Constructor

```
Define Constructor (input, output, learning, hidden, af):
    af = af
    af_der = af_der
    learning = leanring
    architecture  = []
    architecture APPEND input
    FOR layer IN hidden: architecture APPEND layer
    architecture APPEND out
    weights = []
    temp_output = []
    epochacc = []
    makeWeights()
```

This constructor method runs every time a neural network object is created. The attributes are passed as parameters and are appropriately applied to the network variables. The makeWeights method is also called to create initial weights for the network.

-makeWeights()

```
Define makeWeights()
    FOR index IN architecture:
        temporary_matrix = MATRIX(architecture[index], architecture[index+1])
        temporary_matrix = RANDOM(temporary_matrix)
        weights APPEND temporary matrix
```

In this method the architecture variable is used to create an initial matrix with random values as the weights between each layer. The rows and columns of the matrix are decided by the index value of the architecture variable. This is a private method because it is only used by the constructor and should not be used anywhere else.

+train()

```
Define train(inputvector, targetvector)
    temp_output = []
    forward(input)
    temp_output INSERT 0 inputvector
    error = target – temp_output[-1]
    FOR i IN RANGE(LEN(temp_ouput),-1):
        delta = learning * error * af_der(temp_output[i])
        delta = DOT(delta, temp_output[i-1])
        weights[i] += delta
        error = DOT(weights[i], error)
```

This method is responsible for the learning of the network. It incorporates the gradient descent and back propagation algorithm to adjust the weights of the network. This is a public method because it will be called from outside the class itself and will take an input vector and a target vector.

+epoch

```
Define epoch(epochs, data, onehotdata, test, testlabels):
    FOR epo IN RANGE(epochs):
        FOR sample IN RANGE(data):
            train(data[sample], onehotdata[sample])
        corrects, wrongs = evaluate(test, testlabels)
        accuracy = corrects/corrects+wrongs
        epochacc APPEND accuracy
    RETURN epochacc
```

This method is to be used when training the network with multiple epochs. The parameters data, onehotdata(which is the essentially the target vector for each of the data samples), test, and testlabels is used to train the network with the data and then test it with the test data and produce an accuracy for each iteration of the data. Just like the train method this is a public method because it will be called outside of the class.

-forward()

```
Define forward(input, weights)
    IF LEN(weights) == 0:
        RETURN
    ELSE:
        input = af(DOT(weights[0], input))
        temp_output APPEND input
        forward(input, weights[1:])
    RETURN temp_output[-1]
```

This is the method by which an input vector is forward propagated through the network. This is done recursively with the base case being when the length of weights is 0 meaning there are no more calculations to be done. This method is private because it is called within run() which is the method that is called outside the class.

+run()

```
Define run(input)
    RETURN forward(input, weights)
```

This method simply abstracts the forward method but is public because it is only called with an input parameter making it more appropriate for use outside the class as the weights of the network are not required as a parameter.

+evaluate ()

```
Define evaluate(data, labels)
    right = 0
    wrong = 0
    FOR i IN RANGE(LEN(data)):
        prediction = MAXINDEX(run(data[i]))
        IF prediction == label[i]:
            right += 1
        ELSE:
            wrong += 1
    RETURN right, wrong
```

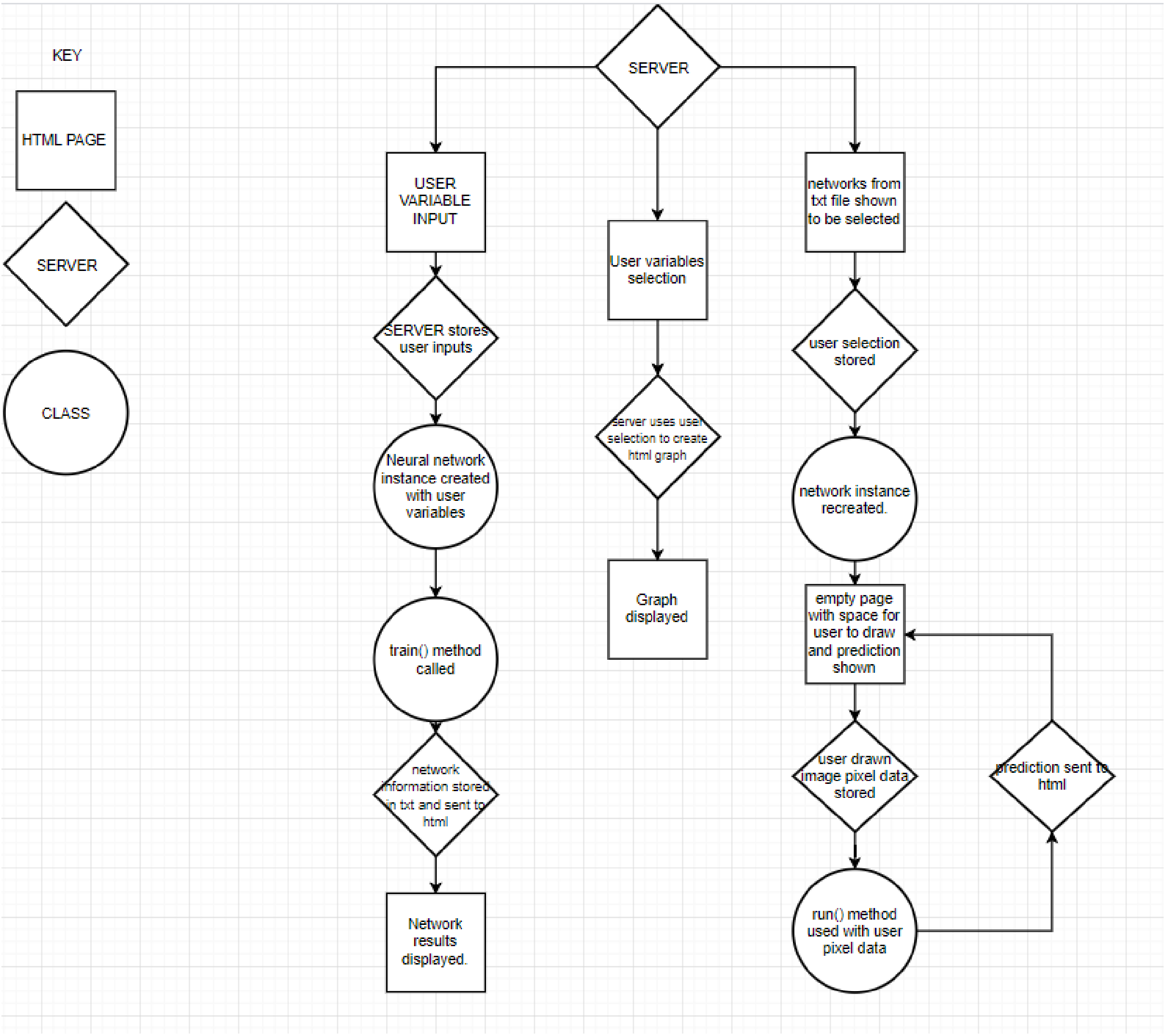
This method is used to calculate the number of correct predictions the network will make when provided with a number of data samples and their corresponding labels. This is a public method as it will be called outside the class methods to determine the accuracy of the network.

IPSO

Inputs	Processes	Storage	Outputs
User -hidden layers -hidden nodes -learning rate -activation function epochs -number of training samples	Create a neural network object with inputted attributes and train with selected number of samples.	Store neural network details, accuracy and time into txt file.	The accuracy of the network and the time taken to train the network.
User -image	Format image to MNIST format Run through selected network		The prediction provided by the network
User -Variable 1 -Variable 2 (selection from neural network attributes)	Read neural network data from text file and produce a scatter graph of the two variables.		Display the graph on a web page
System Image pixel data (a 784-element list)	Train the network or run the network depending on user selection		

All user inputs will be onto a webpage where there will be appropriate fields for the user to enter their inputs. These inputs will be sent to the flask module on python (using post requests) once the user inputs are on python, all processes will be don’t using the neural network class. The information provided by the network class will then be stored onto a text file using a python script. Outputting the data will be handled by flask, this will be done by passing the relevant variables onto webpages in a readable format for the user to read.

Web – server diagram:


KEY
HTML PAGE
SERVER
CLASS
SERVER
USER VARIABLE INPUT
SERVER stores user inputs
Neural network instance created with user variables
train() method called
network information stored in txt and sent to html
Network results displayed.
User variables selection
server uses user selection to create html graph
Graph displayed
networks from txt file shown to be selected
user selection stored
network instance recreated.
empty page with space for user to draw and prediction shown
user drawn image pixel data stored
run() method used with user pixel data
prediction sent to html

Documented design

Introduction

I created a web-based tool that allowed me to quickly select neural network attributes (such as activation function) and then create and train a neural network with the selected attributes, each network was trained using the MNIST handwritten digits dataset. The program then returned and stored the time taken to train the network along with the accuracy of the network. The program also allowed me to test each stored neural network with my own image input. The last requirement of the program was to allow me to select two neural network attributes and then plot a scatter graph of the two variables for example I would be able to plot a graph of learning rate against accuracy. This allowed me to find the features required for a neural network to have a maximum accuracy, and whether this neural network will be suitable to be implemented in an application designed to teach children to write. The tool is also usable by other developers attempting to use neural networks for a similar process.

Packages

For this task I will be using python to implement the main neural network class and algorithms, this will be done using the numpy library, this will be essential because the majority of my algorithms will involve the use of matrices and matrix operations which will be provided by the numpy library.

I will also be using the scipy stats library, this is to generate a suitable random sample of numbers to create the initial weights for the networks.

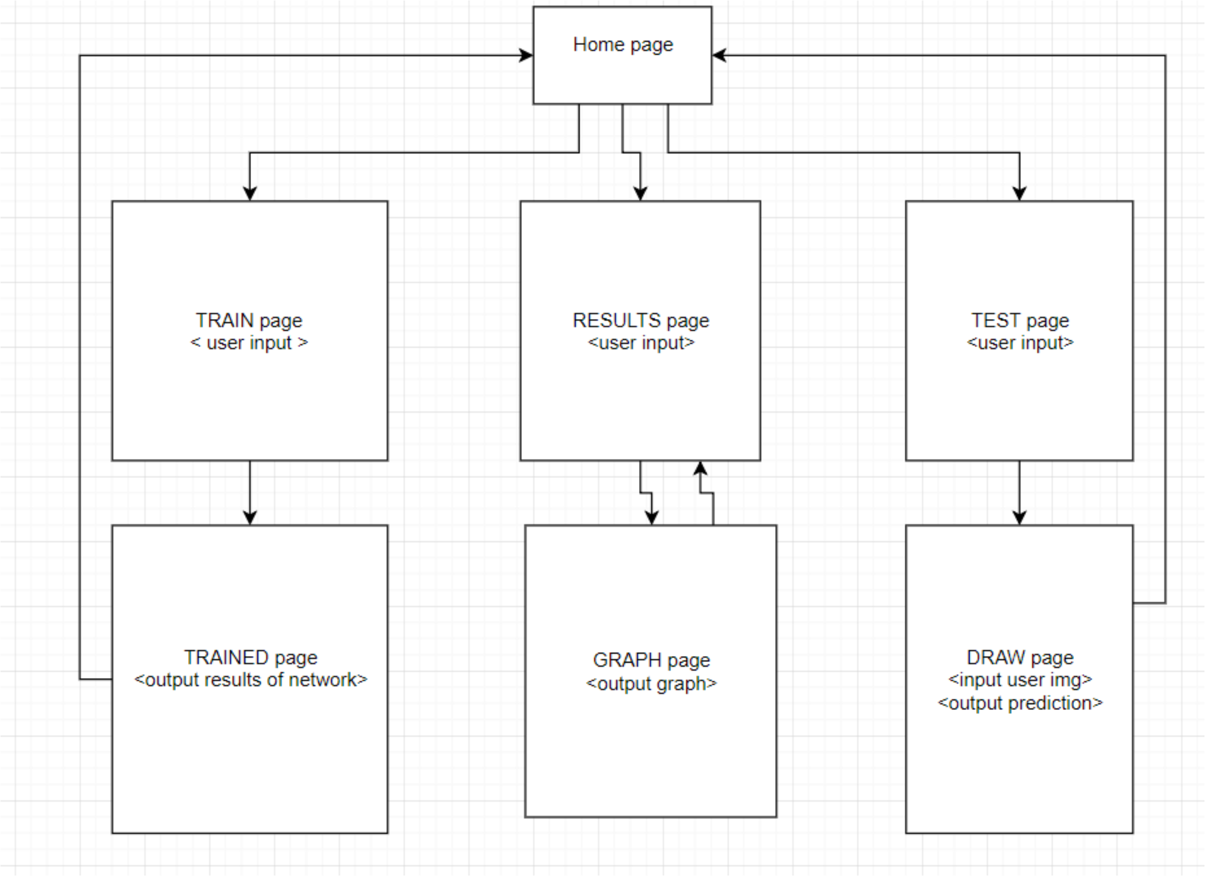
To plot the graphs, I will be using the plotly library because it provides options to output graph data as html code which will be useful in outputting the graph to a html file.

Lastly the pickle library will be used to store networks and the MNIST dataset as binary objects to allow fast access to data.

Server

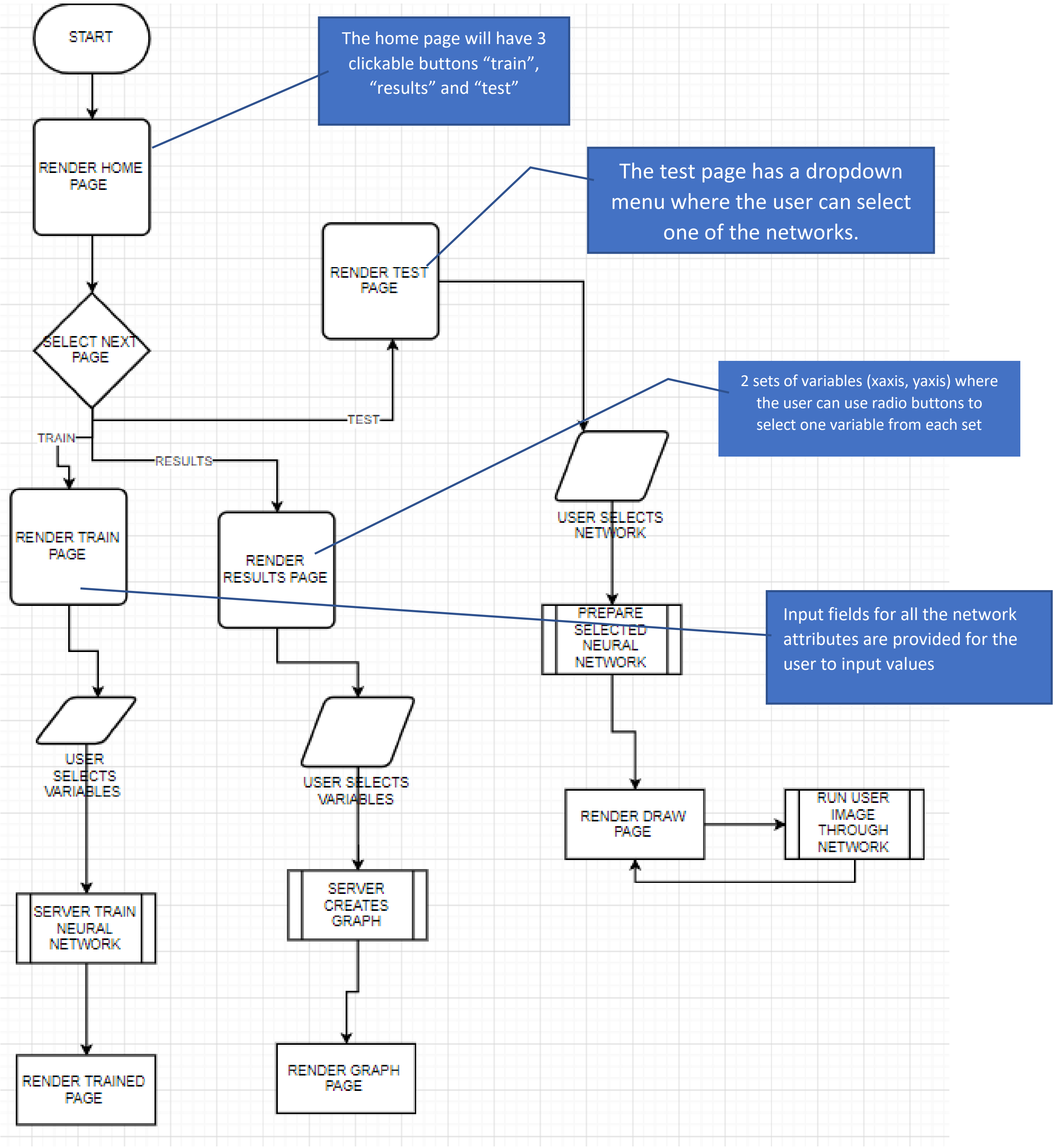
For the server I will be using the flask web framework. This is a python library which will make it easy to integrate and output data from python onto a web page for easy viewing.

Website diagram



This diagram shows all the different webpages that will be on the server. Each arrow shows how each page is linked to another through the use of html buttons. The train page will allow the user to input the neural network parameters and the trained page will output the time taken to train the network as well as the accuracy. The results page will contain a list of variables to select from, the graph page will then output a scatter graph of the two variables. Lastly the test page will allow the user to select a network from a list of all previous networks and the draw page will then allow the user to draw an image which will then be run through the selected network and a prediction will be outputted.

Systems flowchart



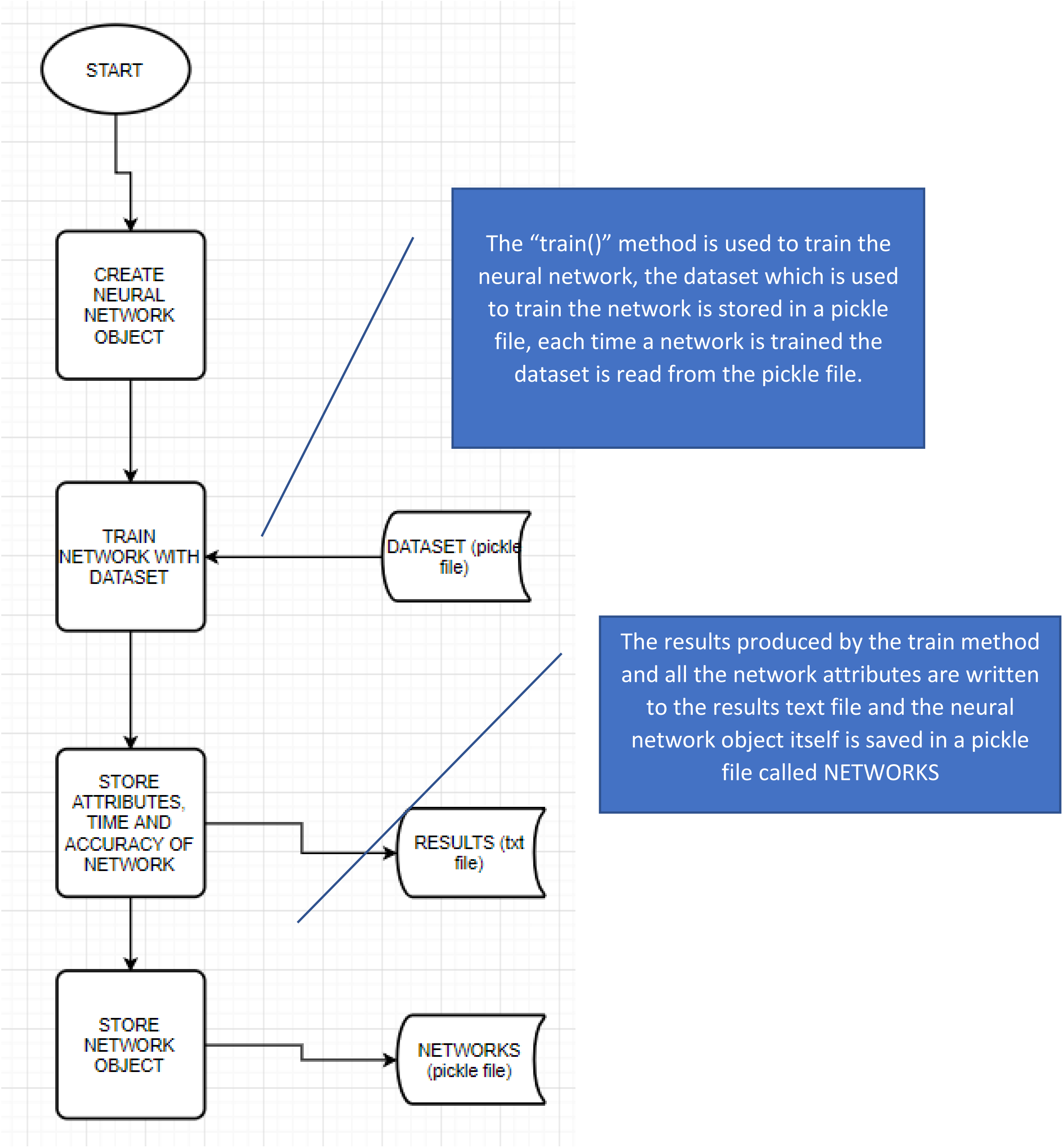
The home page will have 3 clickable buttons “train”, “results” and “test”

The test page has a dropdown menu where the user can select one of the networks.

2 sets of variables (xaxis, yaxis) where the user can use radio buttons to select one variable from each set

Input fields for all the network attributes are provided for the user to input values

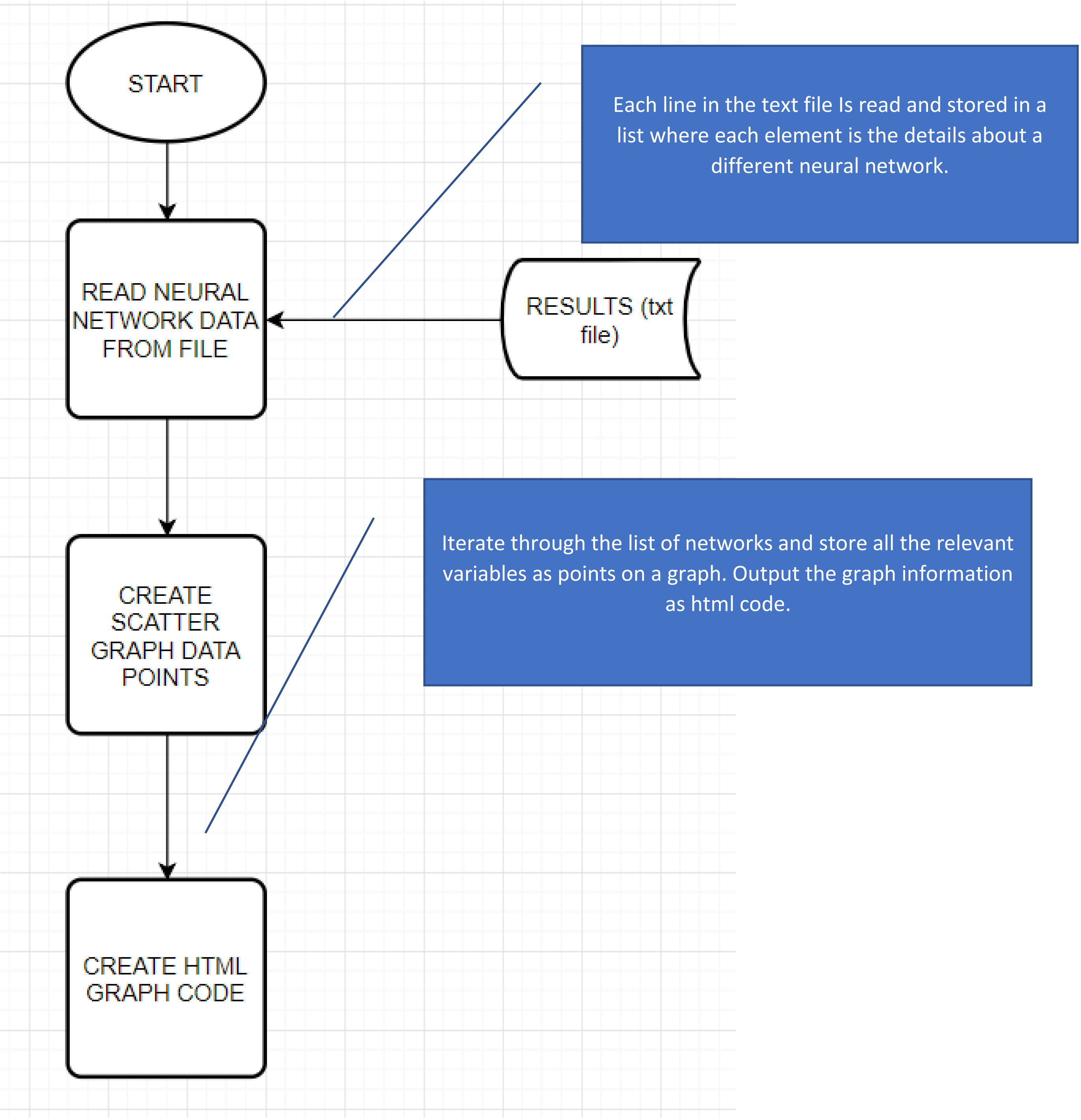
Trian neural network flowchart


START
CREATE NEURAL NETWORK OBJECT
TRAIN NETWORK WITH DATASET
DATASET (pickle file)
STORE ATTRIBUTES, TIME AND ACCURACY OF NETWORK
RESULTS (txt file)
STORE NETWORK OBJECT
NETWORKS (pickle file)

The “train()” method is used to train the neural network, the dataset which is used to train the network is stored in a pickle file, each time a network is trained the dataset is read from the pickle file.

The results produced by the train method and all the network attributes are written to the results text file and the neural network object itself is saved in a pickle file called NETWORKS

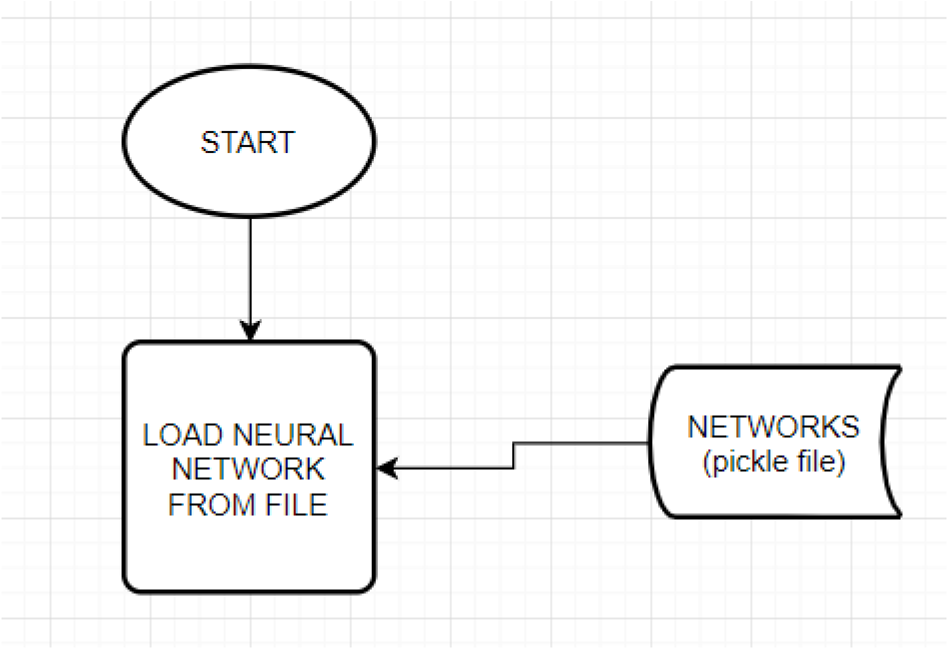
Create graph flowchart


START
READ NEURAL NETWORK DATA FROM FILE
RESULTS (txt file)
CREATE SCATTER GRAPH DATA POINTS
CREATE HTML GRAPH CODE

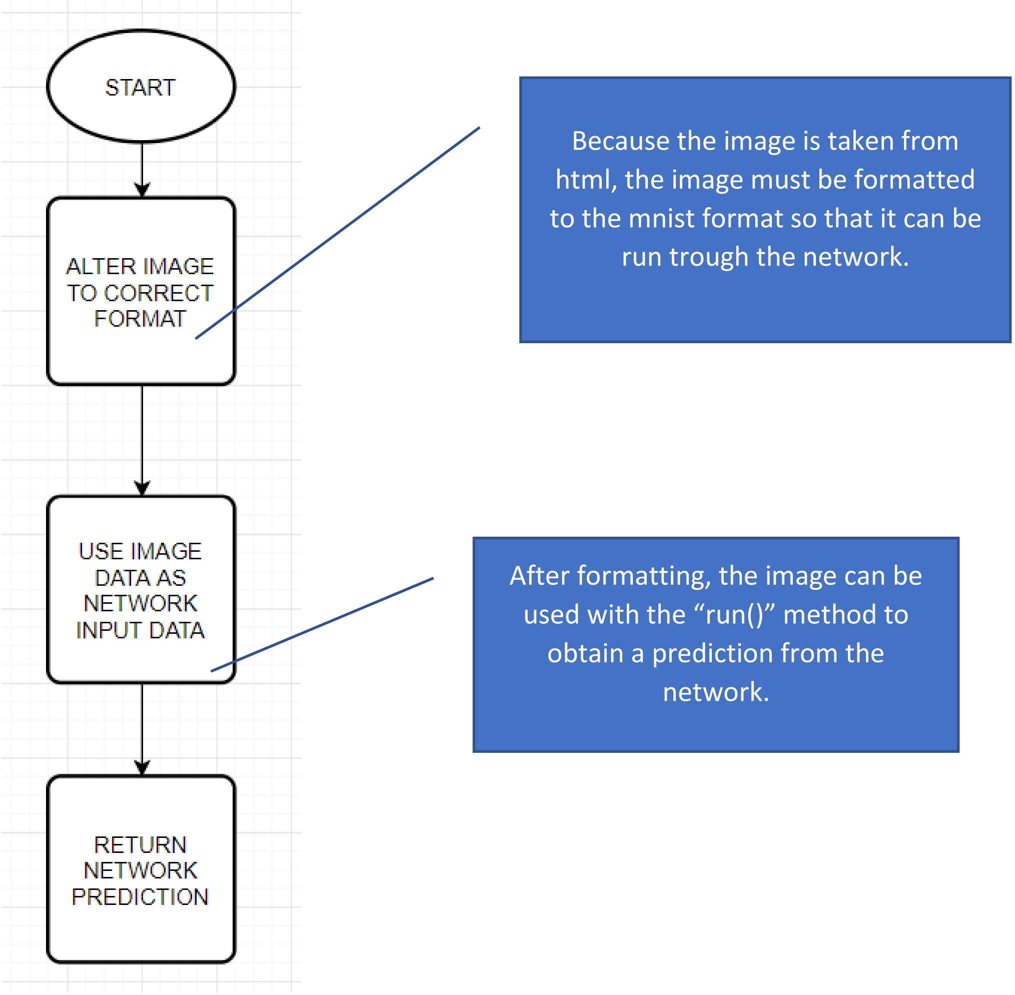
Each line in the text file Is read and stored in a list where each element is the details about a different neural network.

Iterate through the list of networks and store all the relevant variables as points on a graph. Output the graph information as html code.

Prepare neural network flowchart



Run image through network flowchart

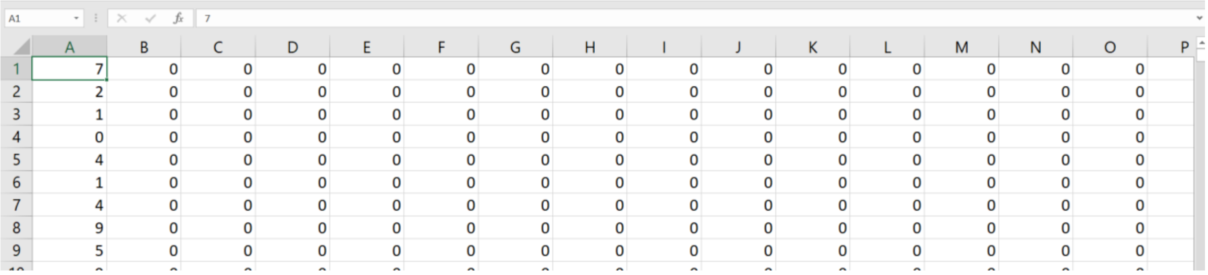

START
ALTER IMAGE TO CORRECT FORMAT
USE IMAGE DATA AS NETWORK INPUT DATA
RETURN NETWORK PREDICTION

Because the image is taken from html, the image must be formatted to the mnist format so that it can be run trough the network.

After formatting, the image can be used with the “run()” method to obtain a prediction from the network.

Obtaining the dataset

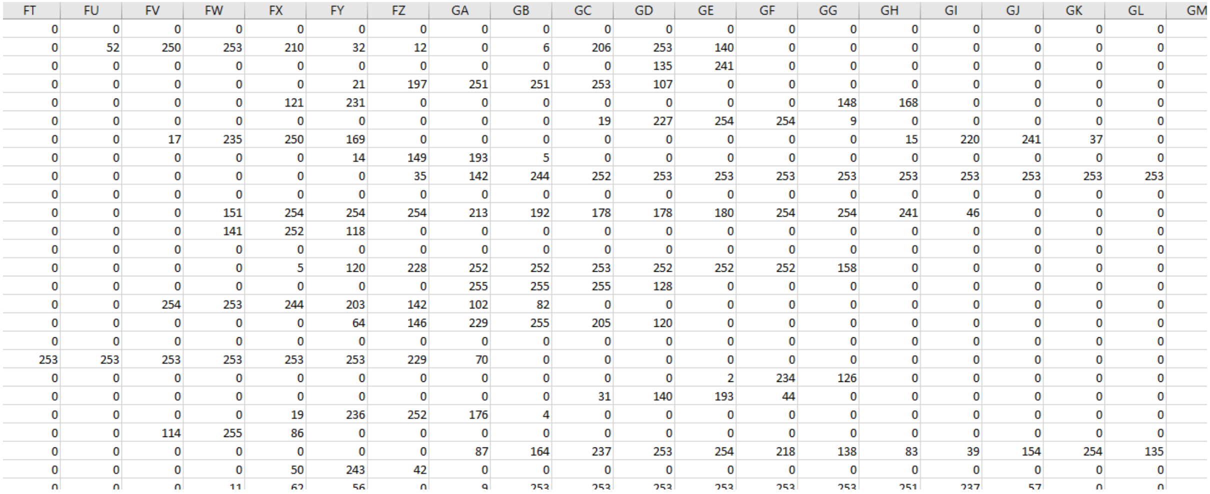
Before beginning my program, I had to obtain the dataset that would be used for training and testing my neural networks. The original dataset and all relevant information can be found here: http://yann.lecun.com/exdb/mnist/index.html, however for my program I decided to download the dataset in csv format from this source: https://www.kaggle.com/oddrationale/mnist-in-csv. I now had 2 csv files, one containing the 60,000 train images and another containing the 10,000 test images. The format of the files can be seen by opening the files using Microsoft excel.



A1 | fx | 7

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
9	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

The first cell of each line contains the label for the image (the first image is a number 7) and is then followed by 784 values each cell is a greyscale pixel value going from the top of the image downward, some of the values can be seen here:



FT	FU	FV	FW	FX	FY	FZ	GA	GB	GC	GD	GE	GF	GG	GH	GI	GJ	GK	GL	GM
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	52	250	253	210	32	12	0	6	206	253	140	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	135	241	0	0	0	0	0	0	0	
0	0	0	0	0	21	197	251	251	253	107	0	0	0	0	0	0	0	0	
0	0	0	0	121	231	0	0	0	0	0	0	0	148	168	0	0	0	0	
0	0	0	0	0	0	0	0	0	19	227	254	254	9	0	0	0	0	0	
0	0	17	235	250	169	0	0	0	0	0	0	0	0	15	220	241	37	0	
0	0	0	0	0	14	149	193	5	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	35	142	244	252	253	253	253	253	253	253	253	253	253	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	151	254	254	254	213	192	178	178	180	254	254	241	46	0	0	0	
0	0	0	141	252	118	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	5	120	228	252	252	253	252	252	252	158	0	0	0	0	0	
0	0	0	0	0	0	0	255	255	255	128	0	0	0	0	0	0	0	0	
0	0	254	253	244	203	142	102	82	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	64	146	229	255	205	120	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
253	253	253	253	253	253	229	70	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	2	234	126	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	31	140	193	44	0	0	0	0	0	0	
0	0	0	0	19	236	252	176	4	0	0	0	0	0	0	0	0	0	0	
0	0	114	255	86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	87	164	237	253	254	218	138	83	39	154	254	135	
0	0	0	0	50	243	42	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	11	62	56	0	9	253	253	253	253	253	253	251	237	57	0	0	

The larger pixel values are generally surrounded by smaller values this is because an antialiasing method is used on each image to smooth out edges.

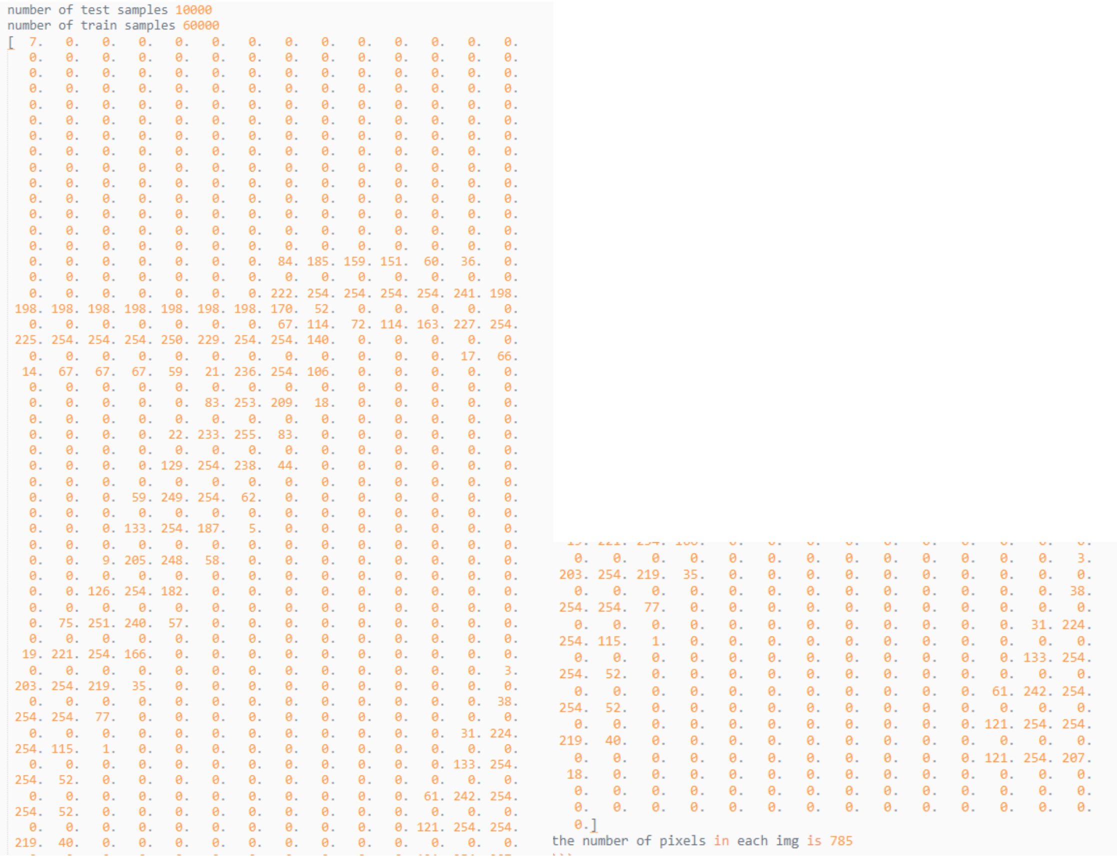
Reading the Data

Next my objective was to format the image data into a suitable form and to then store this data onto a pickle file. This is because the pickle library stores data as a binary stream, this makes it more efficient for python to read that data – making all the processing faster.

The numpy library contains a loadtext function that allowed me to directly read from the csv file into a 2-dimensional numpy array where each element in the array is a line from the csv file.

```
import numpy as np

tedata = np.loadtxt("mnist_test.csv", delimiter=",")

trdata = np.loadtxt("mnist_train.csv", delimiter=",")

print(f"number of test samples {len(tedata)}")

print(f"number of train samples {len(trdata)}")

print(tedata[0])

print(f"the number of pixels in each img is {len(tedata[0])}")
```

In this script I am reading the csv files into python variables using the loadtxt function from numpy. I have used print statements to show me the number of test samples and the number of train samples. As well as all the pixel values for the first data sample and the number of pixels in the image.



```
number of test samples 10000
number of train samples 60000
[  7.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.  84. 185. 159. 151.  60.  36.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0. 222. 254. 254. 254. 254. 241. 198.
 198. 198. 198. 198. 198. 198. 198. 170.  52.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.  67. 114.  72. 114. 163. 227. 254.
 225. 254. 254. 254. 250. 229. 254. 254. 140.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  17.  66.
  14.  67.  67.  67.  59.  21. 236. 254. 106.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.  83. 253. 209.  18.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.  22. 233. 255.  83.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0. 129. 254. 238.  44.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.  59. 249. 254.  62.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0. 133. 254. 187.   5.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   9. 205. 248.  58.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0. 126. 254. 182.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.  75. 251. 240.  57.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  19. 221. 254. 166.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   3.
 203. 254. 219.  35.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  38.
 254. 254.  77.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  31. 224.
 254. 115.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 133. 254.
 254.  52.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  61. 242. 254.
 254.  52.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 121. 254. 254.
 219.  40.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 121. 254. 207.
  18.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.]
the number of pixels in each img is 785
```

The following is produced, there are 785 “pixels” because the first value is the label and is

not part of the image. As you can see the first image from the test data was a number 7 which is correct and the number itself can vaguely be seen in the array. Everything was read correctly so I could move on.

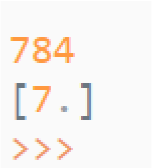
Formatting the data

The next step is to format the data so that the labels and images are in their own arrays and the image data is a smaller number that will work better when being used in gradient descent.

```
teimgs = np.asfarray(tedata[:, 1:])/255

trimgs = np.asfarray(trdata[:, 1:])/255

trlabels = np.asfarray(trdata[:, :1])

telabels = np.asfarray(tedata[:, :1])

print(len(teimgs[0]))

print(telabels[0])
```

In this script I have changed the previously mentioned variables into numpy float arrays. I have used numpys slicing to remove the first value of each element in the arrays (teimgs and trimgs) because these first element of each array is the label for the data. I have then divided each of the values in these 2 arrays by 255 so that each pixel value is between 0 and 1 (this will make it easier when training the network). Next I have used the first element each array within trdata and tedata to produce a list of labels for the unlabelled data previously mentioned. Lastly I used print statements to show that number of pixels in an image array and the label for that image.

This is the result:



```
784
[7.]
>>>
```

There are now 784 pixels as there should be and the label array has been created correctly.

Vectors

Next was to create the one hot vectors, these will be arrays from 0-9 where each index is the label. For example, the one hot vector for the number 7 would be [0,0,0,0,0,0,0,1,0,0] the 7th index is a 1 indicating it is a 7. These are used as target vectors when training the neural network.

```
lr = np.arange(10)

trlabelso = (lr==trlabels).astype(np.float)

telabelso = (lr==telabels).astype(np.float)

trlabelso[trlabelso==0] = 0.01

trlabelso[trlabelso==1] = 0.99

telabelso[telabelso==0] = 0.01

telabelso[telabelso==1] = 0.99
```

To create these target vectors, I have created a numpy list from 0-9 using the arrange function. Next for I use a Boolean expression to convert the 0-9 list into 0s and 1, the 1 being when the Boolean expression is true. This results in a list of vectors where the index of the 1 is the label, for example a list: [0,1,0,0,0,0,0,0,0,0] will represent the label for 1. Lastly I convert any 1s and 0s into 0.01 or 0.99, this is to make sure gradient descent functions correctly.

Lastly all these arrays are dumped into a pickle file:

```
with open("numbers.pkl", "bw") as file:

data = (trimgs, teimgs, trlabels, telabels, trlabelso, telabelso)

pickle.dump(data, file)
```

Here is a table explaining each of the variables:

Variable name	Explanation
Trimgs	This is 2d array containing all the pixel data for the 60,000 training images
Teimgs	This is a 2d array containing all the pixel data for the 10,000 test images
Trlabels	This is an array containing the labels for the training images
Telabels	This is an array containing the labels for the test images
Trlabelso	This is a 2d array containing the target vectors for the training images
telabelso	This is a 2d array containing the target vectors for the test images.

The dataset is now ready to be used.

The neural network class

NN			
Attributes:			
Name	**Type**	**Typical value**	**Description**
Self.af	Function	Sigmoid, relu, tanh	Standing for activation function, is passed through as a class parameter and is then assigned a function depending on what is passed through.
Self.af_der	Function	(Sig, rel, tanh)_der	Stands for activation function derivative and is assigned a function depending on what the value of “self.af” is.
Self.learning	Float	<10	This is passed as a class parameter and is the learning rate of the neural network. A float.
Self.architecture	List	-	This is assigned as an empty list to represent the structure (the number of nodes per layer) of the neural network. Immediately after being created the parameters input_nodes, output_nodes and hidden are appended to self.architecture.
Slef.weights	List	-	An empty list intended to store the values of the weights between the layers.
Self.temp_output	List	-	An empty list intended to store the value at each layer after a forward pass.
Self.epochacc	List	-	An empty list created to store the accuracy after each epoch.
Methods			
Name	**Parameters**	**Return value**	**Description**
makeWeights		-	Creates the initial weights for the matrix.
Train	List, list	-	Trains the network with a given input and target output.
Epoch	Int, list, list, list, list	list	Trains the network n number of times for a given dataset, returns the accuracy of after each iteration.
Forward	List, list	list	Passes a given data sample through the network and returns the output.

Run	list	List	Calls the forward method
Evaluate	List, list	Integer, integer	Calculates the number of correct and incorrect predictions for a given dataset returns the number of correct and incorrect predictions.

CONSTRUCTOR

```
def __init__(self, input_nodes, output_nodes, learning, hidden, af):
    if af == "sigmoid":
            self.af = sigmoid
            self.af_der = sig_der
    elif af == "relu":
            self.af = relu
            self.af_der = rel_der
    elif af == "tanh":
            self.af = tanh
            self.af_der = tanh_der
    self.learning = learning
    self.architecture = []
    self.architecture.append(input_nodes)
    for k in hidden:
            self.architecture.append(k)
    self.architecture.append(output_nodes)
    self.weights = []
    self.temp_output = []
    self.epochacc = []
    self.makeWeights()
```

This is called whenever the neural network class is called and is how the neural network object is initialised. The parameters are the number of input, output nodes, learning rate, hidden layers (a list of integers where each element is the number of hidden nodes and the length of the list is the number of hidden layers) and the activation function. Firstly, the objects activation function is initialised using the previously defined functions. The architecture of the object is a list where each element is the number of nodes in that layer. The makeWeights function is also called here because each network is automatically assigned random weight values between all the nodes, these weights will be stored in the weights variables.

CREATING THE INITIAL WEIGHTS

```
def makeWeights(self):
        for i in range(len(self.architecture)-1):
            limit = 1/np.sqrt(self.architecture[i])
            w = truncated_normal(mean=0, sd=1, low=-limit, upp=limit)
            temp_weight_matrix = w.rvs((self.architecture[i+1], self.architecture[i]))
            self.weights.append(temp_weight_matrix)
```

This is called whenever the neural network class is called and is how the neural network object is initialised. The parameters are the number of input, output nodes, learning rate, hidden layers (a list of integers where each element is the number of hidden nodes and the length of the list is the number of hidden layers) and the activation function. Firstly, the objects activation function is initialised using the previously defined functions. The architecture of the object is a list where each element is the number of nodes in that layer. The makeWeights function is also called here because each network is automatically assigned random weight values between all the nodes, these weights will be stored in the weights variables

FORWARD PROPAGATE

```
def forward(self, input_vector, weights):
    if len(weights) == 0:
        return
    else:
        if len(weights) == 1 and (self.af == relu or self.af == tanh):
            input_vector = sigmoid(np.dot(weights[0], input_vector))
        else:
            input_vector = self.af(np.dot(weights[0], input_vector))
        self.temp_output.append(input_vector)
        self.forward(input_vector, weights[1:])
  return self.temp_output[-1]
```

This method within the neural network class takes an input vector and the weights of the network to pass the input vector through the network. This is done recursively. The base case is if the weights variable (which is a list) has a length of 0, meaning that there are no more calculations to be done, if this case is met, the function will simply return and unwind. Otherwise, the new input vector is the dot product of the current input vector and the weight matrix ahead of it, the activation function is applied on this value. There is a check whether the networks activation function is tanh or relu because the sigmoid function must always be applied on the output layer regardless of the activation function. Next the input vector is appended onto the temporary out list to store all the values of each layer (this is useful for when it needs to be trained). Once this is done the forward function is called again, with the new input vector and the sliced weights list. Lastly when the function completely unwinds the last element is returned, this is the output layer.

```
def run(self, input_vector):
    return self.forward(input_vector, self.weights)
```

This method is an abstraction of the previous forward method, it only requires an input vector and will return the output layer.

TRAINING ALGORITHM (GRADIENT DESCENT)

```
def train(self, input_vector, target_vector):
    input_vector = np.array(input_vector, ndmin=2).T
    target_vector = np.array(target_vector, ndmin=2).T
    self.temp_output = []
    self.forward(input_vector, self.weights)
    self.temp_output.insert(0, input_vector)
    error = target_vector - self.temp_output[-1]
    for i in range(len(self.temp_output)-1,0,-1):
        if i == len(self.temp_output)-1 and (self.af == relu or self.af == tanh):
            delta = self.learning * error * sig_der(self.temp_output[i])
        else:
            delta = self.learning * error * self.af_der(self.temp_output[i])
        delta = np.dot(delta, self.temp_output[i-1].T)
        self.weights[i-1] += delta
        error = np.dot(self.weights[i-1].T, error)
```

This is the method that applies the gradient descent formula to train the network, it requires an input vector and a target vector. Firstly because both the input vector and target vector are passed as 2d arrays, they are converted into vertical numpy arrays using the np.array function and the transpose function (.T). The temp output is cleared because the input vector is to be passed through the network using the forward method. The error is calculated by subtracting the output layer from the target vector. Next a for loop is used to iterate through the temp output list and weights in from end to front. There is a check to see if the current iteration is on the output layer, and whether the networks activation function is relu or tanh, this is because the sigmoid function is always applied to the output layer (regardless of the networks activation function) and because of this the sigmoid derivative must be used in the calculation. The delta (meaning change) is assigned the gradient of the error with respect to the current layer multiplied by the learning rate. This value is then added to the respective weights of the layer. After this the new error is the dot product of the old error and the weights.

EVALUATE NETWORK ACCURACY

```
    def evaluate(self, data, labels):
        right =0
        wrong =0
        for i in range(len(data)):
            prediction = argmax(self.run(data[i]))
            if prediction == labels[i]:
                right += 1

            else:
                wrong += 1

        return right,wrong
```

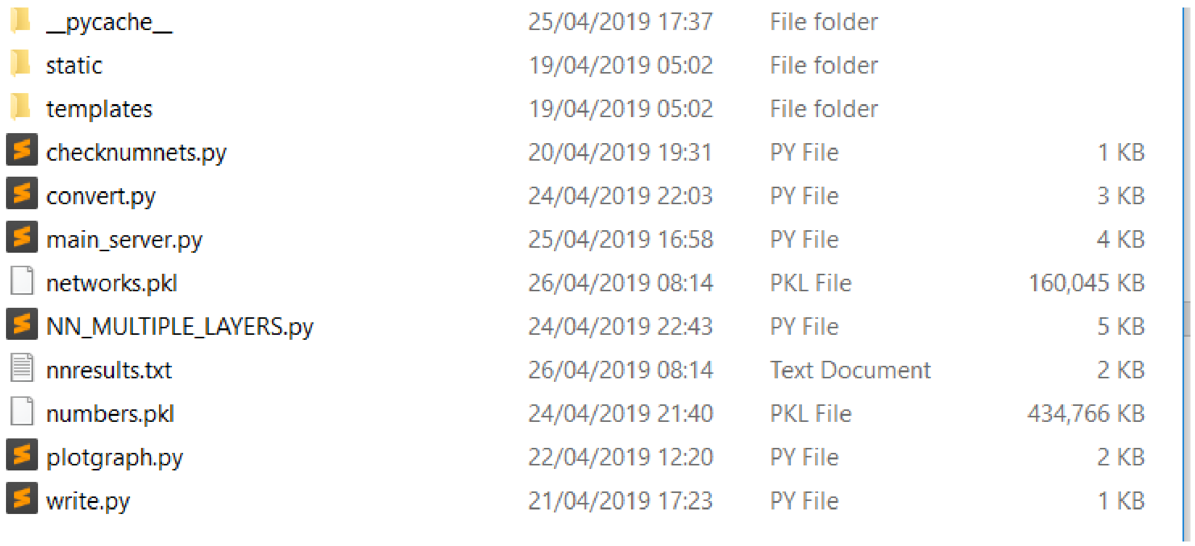
This method is designed to calculate the accuracy of the network. It takes a list of data samples and their respective labels as parameters. For each item in the dataset the item is passed through the network using the run method, the prediction is found using argmax (this returns the index of the largest value in the output layer) if correct the correct variable is incremented or if not the wrong variable is incremented and the correct and wrongs variables are returned.

MULTIPLE EPOCHS

```
def epoch(self, epochs, data, onehotdata, test, testlabels):
    for i in range(epochs):
        for x in range(len(data)):
            self.train(data[x], onehotdata[x])
        corrects, wrongs = self.evaluate(test, testlabels)
        testacc = corrects/(corrects+wrongs)
        singleacc = testacc
        self.epochacc.append(singleacc)
    return self.epochacc
```

This method requires a number of epochs parameter, a list of data, that data’s one hot vectors, test data and its respective labels. A nested for loop is used to train the network the specified number of times, each time the neural network has been trained the accuracy is calculated and appended onto the epochacc list. Once complete, a list of the accuracy of the network after each epoch is returned.

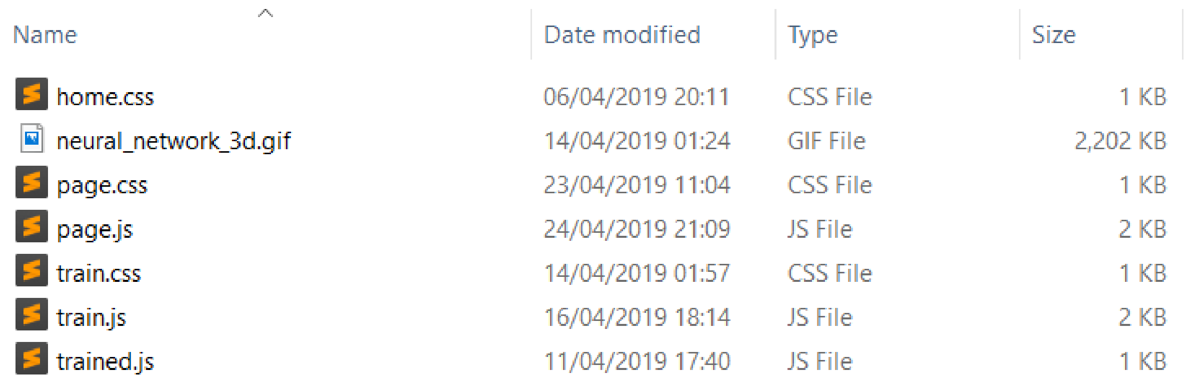
File structure



__pycache__	25/04/2019 17:37	File folder	
static	19/04/2019 05:02	File folder	
templates	19/04/2019 05:02	File folder	
checknumnets.py	20/04/2019 19:31	PY File	1 KB
convert.py	24/04/2019 22:03	PY File	3 KB
main_server.py	25/04/2019 16:58	PY File	4 KB
networks.pkl	26/04/2019 08:14	PKL File	160,045 KB
NN_MULTIPLE_LAYERS.py	24/04/2019 22:43	PY File	5 KB
nnresults.txt	26/04/2019 08:14	Text Document	2 KB
numbers.pkl	24/04/2019 21:40	PKL File	434,766 KB
plotgraph.py	22/04/2019 12:20	PY File	2 KB
write.py	21/04/2019 17:23	PY File	1 KB

This is the root directory of the project. All python files, stored neural network objects are stored in the “networks.pkl” pickle file. “nnresults.txt” is the text file where all the network information is stored. Lastly “numbers.pkl” is the pickle file in which the mnist handwritten digits’ dataset is stored.

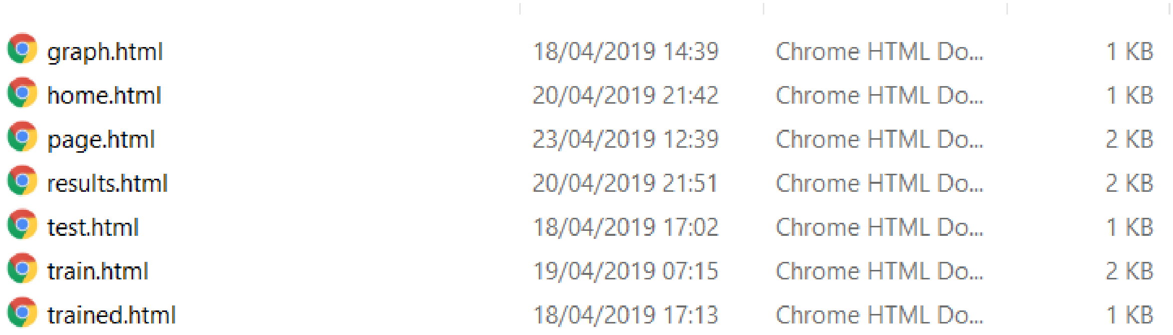
Static



Name	Date modified	Type	Size
home.css	06/04/2019 20:11	CSS File	1 KB
neural_network_3d.gif	14/04/2019 01:24	GIF File	2,202 KB
page.css	23/04/2019 11:04	CSS File	1 KB
page.js	24/04/2019 21:09	JS File	2 KB
train.css	14/04/2019 01:57	CSS File	1 KB
train.js	16/04/2019 18:14	JS File	2 KB
trained.js	11/04/2019 17:40	JS File	1 KB

The static folder is where all the different stylesheets and javascript files are stored. These files each relate to a html file, the css files control the visual aspects of the webpage and the javascript files control the functions of the webpage. There is also a neural network animation gif, this will be shown as a loading screen when the user trains a network – because the training time could be long in some instances and the animation will show that the webpage is still functioning.

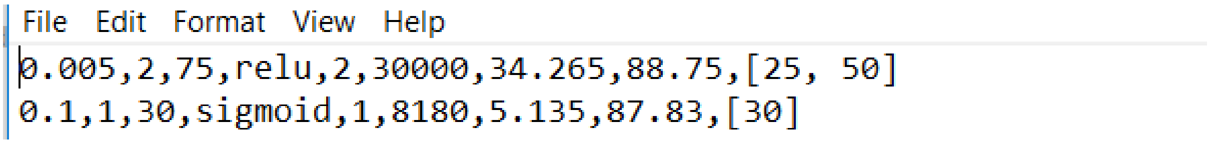
Templates



graph.html	18/04/2019 14:39	Chrome HTML Do...	1 KB
home.html	20/04/2019 21:42	Chrome HTML Do...	1 KB
page.html	23/04/2019 12:39	Chrome HTML Do...	2 KB
results.html	20/04/2019 21:51	Chrome HTML Do...	2 KB
test.html	18/04/2019 17:02	Chrome HTML Do...	1 KB
train.html	19/04/2019 07:15	Chrome HTML Do...	2 KB
trained.html	18/04/2019 17:13	Chrome HTML Do...	1 KB

Within the templates folder every html file is stored. As seen in the images each webpages stylesheet and javascript file are named the same, this is to make it easier to link them to each other.

Text structure



```
0.005,2,75,relu,2,30000,34.265,88.75,[25, 50]
0.1,1,30,sigmoid,1,8180,5.135,87.83,[30]
```

The format is as follows:

Learning rate, number of hidden layers, total number of hidden nodes, activation function, epochs, training samples, time taken to train, hidden layer structure.

This is how the neural network information is stored within the text file. When reading the text file, it can be read as one large string. Each line network is spirited by “\n” and then each attribute can also be separated by the “,”.

Algorithms

I will now go through each of the files seen in the project structure image and will explain the purpose of each one.

Python Files

NN_MULTIPLE_LAYERS – this file is responsible for the neural network class and how the class is implemented into the server.

Creating the activation functions

```
def sigmoid(x):

        return 1/(1 + np.e**-x)

def sig_der(x): # sigmoid dirivative

        return x * (1.0 -x)

def relu(x):

        return x * (x>0)

def rel_der(x):

         return 1 * (x > 0)

def tanh(x):

        return np.tanh(x)

def tanh_der(x):

        return 1 - (tanh(x)**2)
```

These are each of the mathematical activation functions and their respective derivatives ready to be used by the neural network class.

Implementing usage of the neural network class

```
def main(input_nodes, output_nodes, learning, hidden, af, epochs, trainrange):
    totalnodes = sum(hidden)
    with open("numbers.pkl", "br") as fh:
        data = pickle.load(fh)
    trimgs = data[0]
    teimgs = data[1]
    trlabels = data[2]
    telabels = data[3]
    trlabelso = data[4]
    telabelso = data[5]
    trimgs, trlabelso = trimgs[:trainrange], trlabelso[:trainrange]
    layers = len(hidden)
    ann = NN(input_nodes, output_nodes, learning, hidden, af)
    start = time.clock()
    if epochs == 1:
        for i in range(len(trimgs)): ann.train(trimgs[i], trlabelso[i])
        corrects, wrongs = ann.evaluate(teimgs, telabels)
        acc = corrects/(corrects+wrongs)
    elif epochs > 1:
        acc = ann.epoch(epochs, trimgs, trlabelso, teimgs, telabels)
    end_time = round((time.clock() - start),3)
    final_values = [learning, layers, totalnodes, af, epochs, trainrange, end_time, acc, hidden]
    with open("networks.pkl", "ab") as d:
        pickle.dump(ann, d)
    return final_values
```

This function is separate to the class, it is used by the server to receive all the relevant information. The function takes all the required parameters to create the neural network as well as the number of epochs and “trainrange”, this is the number of training samples to be used to train the network.

A list slice operation is used to alter the training image list to only contain the specified number of samples.

Once the function is complete a list containing all the relevant attributes and results of the network is created. The neural network object is pickle dumped to the networks file and the list of final values is returned.

```
def runimg(img, option):
    from convert import web2arr
    with open("networks.pkl", "rb") as f:
        for i in range(option+1):
            ann = pickle.load(f)

    x = np.argmax(ann.run(web2arr(img)))
    return x
```

This is also in the neural network python file; this is used to run the user image data through a selected network. The option parameter relates to the index of the stored network. Each network is read from the pickle file until the correct one is read, the user image data is then ran through the network and the prediction is obtained. The argmax function is used because the class run function returns an array of values, the index of the largest value is the prediction and argmax returns the index of the largest value.

Write- This python file is responsible for writing the list (containing all the network details) to the text file, in a format that can easily be read by python.

```
def writetotxt(filename, data):
    with open(filename, "a")as file:
        for x in data:
            if x == data[-2]:
                if type(x) == list:
                        x=x[-1]
              x =x *100
              x=str(x)+","
            elif x == data[-1]:
               x = str(x)
            else: x = str(x) + ","
            file.write(x)
        file.write("\n")
```

This function Is called once the final values list is returned. The neural network results txt file is opened using the append option. Next a for loop is used to iterate over each element in the data. The first check is if the current value is the 2nd last value (the accuracy), if it is a list, only take the final value in that list. The reason for this is because if the network is trained using more than one epoch then a list of accuracies is returned (for each epoch) however only the last value of the list is kept because this is the final accuracy of the network. It is then multiplied by 100 (to make it a percentage). All values are written as strings with a "," ahead of them with the exception of the last element. This is to maintain a format in which the file can be read correctly.

Plotgraph- This is the file where the graph object is created and returned as html code.

```
def divplot(xvar, yvar):
    form = ["learning rate", "number of layers", "number of hidden nodes", "epochs", "samples", "time", "accuracy", "structure"]
    with open("nnresults.txt")as file:
        txtdata = file.readlines()
    txtdata = [i.strip("\n").split(",") for i in txtdata]
    relu =[]
    tanh = []
    sigmoid =[]
    for i in range(len(txtdata)):
        if "relu" in txtdata[i]:
            del txtdata[i][3]
            relu.append(txtdata[i])
        if "tanh" in txtdata[i]:
            del txtdata[i][3]
            tanh.append(txtdata[i])
        if "sigmoid" in txtdata[i]:
            del txtdata[i][3]
            sigmoid.append(txtdata[i])
    relux, reluy = [float(i[xvar]) for i in relu], [float(i[yvar]) for i in relu]
    relu_sorted = sorted(zip(relux, reluy))
    relux, reluy = map(list,zip(*relu_sorted))
    tanhx, tanhy =[float(i[xvar]) for i in tanh], [float(i[yvar]) for i in tanh]
    tanh_sorted = sorted(zip(tanhx, tanhy))
    tanhx, tanhy = map(list,zip(*tanh_sorted))
    sigmoidx, sigmoidy = [float(i[xvar]) for i in sigmoid], [float(i[yvar]) for i in sigmoid]
    sigmoid_sorted = sorted(zip(sigmoidx, sigmoidy))
    sigmoidx, sigmoidy = map(list,zip(*sigmoid_sorted))
```

Once the lists are created, each index in one list corresponds to the same index in the other list. Because of this the zip function is used to create a single list of tupples where each tupple is a point on the graph. This list is sorted in ascending order based on the 1st element in the tuple(which is the x value). The list of tuples is then split again. The zipping of the lists is required to keep the correct values mapped to each other.

```
relutrace = go.Scatter(
    x=relux,
    y=reluy,
    name="RELU"
)
tanhtrace = go.Scatter(
    x=tanhx,
    y=tanhy,
    name="TANH"
)
sigmoidtrace = go.Scatter(
    x=sigmoidx,
    y=sigmoidy,
    name="SIGMOID"
)
data = [relutrace, tanhtrace, sigmoidtrace]
layout = go.Layout(
    title = go.layout.Title(text=f"{form[yvar]} against {form[xvar]}"),

    xaxis=go.layout.XAxis(
        title=go.layout.xaxis.Title(text=f"{form[xvar]}")
    ),
    yaxis=go.layout.YAxis(
        title=go.layout.yaxis.Title(text=f"{form[yvar]}")
    )
)
fig = go.Figure(data=data, layout=layout)
x = py.offline.plot(fig, include_plotlyjs=False, output_type="div")
return x
```

In this function the plotly module is used to construct a scatter graph with three different points and lines for the three different activation functions. The function takes parameters for x and y axis variables, these parameters work as index values. The text file containing all the neural network variables is read and formatted so that it is a 3dimensional array. Each activation function is given its own list which contains all the network data for the networks trained using the activation function. Using a for loop and the sort function, two new lists for each activation is created, one for the x axis and one for the y axis. From the initial list each element is selected and from that the specified element is chosen (using the parameters passed into the function). At this stage there are 6 one dimensional lists consisting of graph data for 3 different traces. Each trace is labelled and plotted onto the graph object. The function then outputs a div html tag to be used by the server to display the graph on a webpage.

Convert-This is where the user image (from the webpage) is formatted correctly to be accurately passed through the network.

```
def web2arr(img):
    img = np.asfarray(img)
    img = img.reshape((300, 300))
    img = Image.fromarray(img)
    img = img.resize((28,28))
    temp = np.asarray(img)
    while np.sum(temp[0]) == 0:
     temp = temp[1:]
    while np.sum(temp[:,0]) ==0:
     temp = np.delete(temp,0,1)
    while np.sum(temp[-1]) == 0:
     temp = temp[:-1]
    while np.sum(temp[:, -1]) == 0:
     temp = np.delete(temp, -1,1)
    rows,cols = temp.shape
    if rows > cols:
     factor = 20.0/rows
     rows = 20
     cols = int(round(cols*factor))
     temp = cv2.resize(temp, (cols,rows))
    else:
     factor = 20.0/cols
     cols = 20
     rows = int(round(rows*factor))
     temp = cv2.resize(temp, (cols,rows))
    colsPadding = (int(math.ceil((28-cols)/2.0)),int(math.floor((28-cols)/2.0)))
    rowsPadding = (int(math.ceil((28-rows)/2.0)),int(math.floor((28-rows)/2.0)))
    temp = np.lib.pad(temp,(rowsPadding,colsPadding),'constant')

    shiftx,shifty = getBestShift(temp)

    shifted = shift(temp,shiftx,shifty)

    temp = shifted

    fac = 255*0.99+1

    temp = np.asfarray(temp)/fac

    temp = temp.reshape(784)

    return temp
```

This function is required to convert the list posted by the webpage into the standard mnist format, so that the network can produce an accurate prediction. From the mnist website:

“The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.”

This means that the user image must be converted into a 20 * 20 pixel box, and then centre fitted onto a 28*28 pixel field.

After researching on how to recreate the mnist format I found an article which explained and demonstrated how any image could be formatted to the mnist format. https://medium.com/@o.kroeger/tensorflow-mnist-and-your-own-handwritten-digits-4d1cd32bbab4

I used parts of this algorithm and my own algorithm to be able to convert the canvas image from the webpage into the correct format.

The function does this by first converting the single image array into a 300 * 300 matrix. 300 is used because this is the canvas size on the webpage. Next using the Image library from PIL and image object is created from the numpy array, this is so that the “resize” function can be used to change the image from a 300*300 image to a 28*28 image, once this is done the a new numpy array is created using the new 28*28 matrix.

After this the empty space or white pixels are removed from the image using for loops to check each pixel around the edges.

Next the image is given padding (white pixels) so that the size of the image is 28*28.

The image is then shifted to the centre of the box. Lastly each value in the array is divided by the same factor as the training data and the image is converted into a 784-list ready to be passed into the network as an input vector.

User Interface

My program is a web-based application, because of this I have had to setup a server which connects the web element of the project to the python side. Flask is a micro web framework for python and it was perfect for this task.

First, I will explain the web files and their purposes.

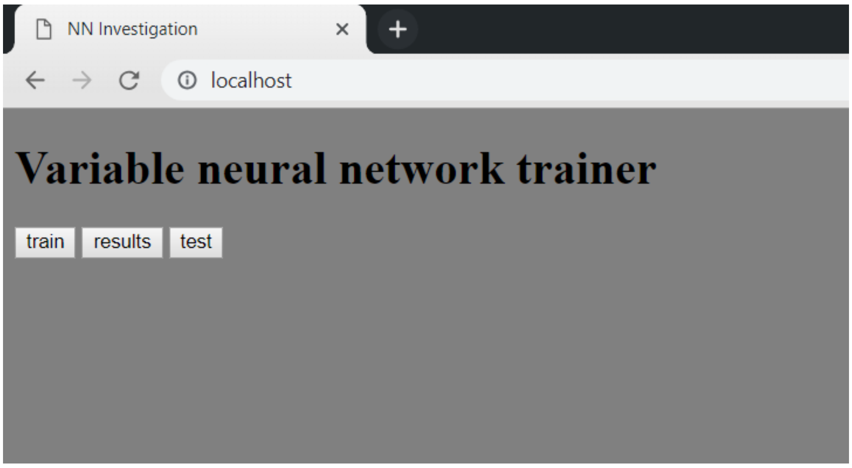
Home.html

```
<!DOCTYPE html>

<html>

<head>

    <title>NN Investigation</title>

    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='home.css') }}">

</head>

<body>

    <h1>Variable neural network trainer</h1>

    <button onclick="location.href='train'">train</button>

    <button onclick="location.href='results'">results</button>

    <button onclick="location.href='test'">test</button>

</body>

</html>
```

When the button is clicked a request is sent to the server eg for the test button the "/test" request is sent.

This is the home page of the application; I have simply created three clickable buttons where each one links to its respective page. The "home.css" stylesheet is also linked using one of flasks features where the directory of the filename is found within the project folder. The stylesheet for this page simply sets the background colour to grey.

This is what the page looks like.



Variable neural network trainer

Train.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Train a NN</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='train.css') }}">
    <script type="text/javascript" src="https://code.jquery.com/jquery-3.3.1.js"></script>
</head>
<body>                          1
    <div id="animation"><img src="{{ url_for('static', filename='neural_network_3d.gif') }}" id="img"></div>
                                                    2
    <form id="layerform" class="onload">
        <input type="number" id="#oflayers" max="5" min="1" required>
        <label>number of hidden layers</label><button type="submit" id="selectlayers">select</button>
    </form>
    <form action="{{ url_for('data') }}" id="nn" method="post" class="onload" onsubmit="hide()">
        <br>
                                          3
        <div id="layers"></div>
        <label>learning rate  </label><input type="number" name="learningrate" step="0.001" required><br>
        <label>Activation Function</label><br>
        <input type="radio" name="activation" value="sigmoid" checked>sigmoid<br>
        <input type="radio" name="activation" value="relu">relu<br>
        <input type="radio" name="activation" value="tanh">tanh<br>
        <label>epochs  </label><input type="number" name="epochs" required><br>
        <label>training data</label><input type="range" min="0" max="60000" id="data_slider"
name="trainrange"><p id="currentrange"></p><br>
        <input type="submit" value="start training">
    </form>
</body>
<script type="text/javascript" src="{{ url_for('static', filename='train.js') }}"></script>
</html>
```

This is the page where the user will enter all the attributes for the network that they would like train.

1.This div tag is initially hidden and is shown using javascript once the user has entered their network attributes to show that the network is being trained.

2. This is a form that allows the user to enter the number of hidden layers they would like, using that input the second form (where all the other attributes are entered) will show input fields for each of the users layers, for example if the user enters 3 hidden layers, 3 input fields will appear for the user to enter the number of hidden nodes in each of these layers.

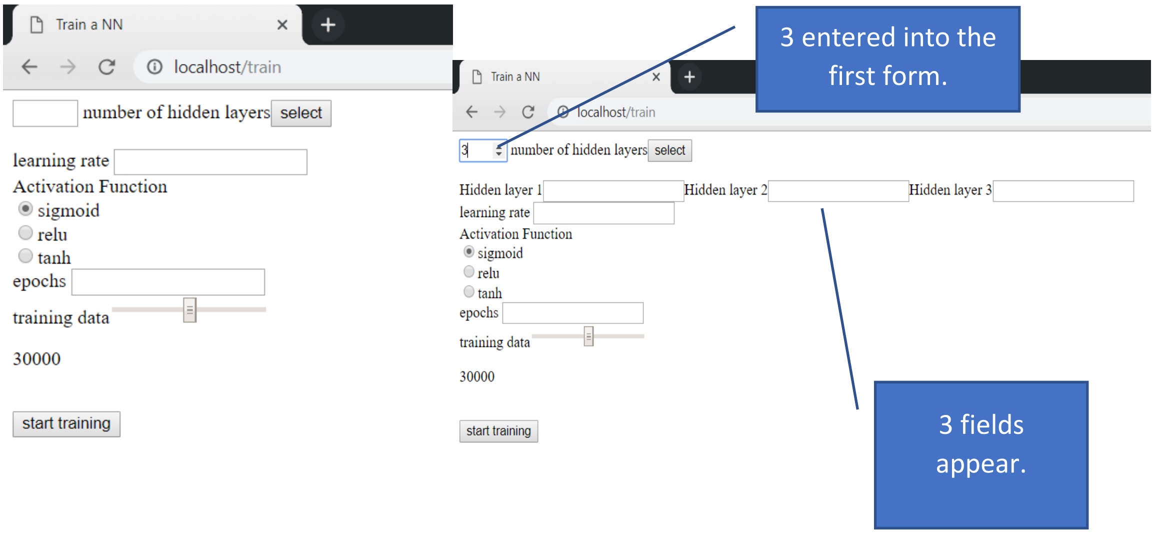
3. This form is where the user enters all of their network information for the activation function, I have used radio buttons (with sigmoid being checked by default) as there are only 3 choices. For the training data I have chosen to use a slider, this allows the user to easily select a value between 0 and 60,000. This form uses the post method to send the form to "/data" this will be explained later.

Train.js

```
document.getElementById('selectlayers').addEventListener("click", getlayers);
$("#layerform").submit(function(e) {
  e.preventDefault();
});
function getlayers(){
    var x = document.getElementById("#oflayers").value;
    var div = document.getElementById("layers");
    div.innerHTML = "";
    for (i=0; i<x; i++) {
        if (x > 5) {
            return;
        }
        var temp = document.createElement("input");
        temp.type = "number";
        temp.name = "hiddenlayers";
        temp.required = true;
        var label = document.createElement("label");
        label.innerHTML = 'Hidden layer ' + (i+1);
        div.appendChild(label);
        div.appendChild(temp);
    };
}
var slider = document.getElementById("data_slider");
var current = document.getElementById("currentrange");
current.innerHTML = slider.value;
slider.oninput = function(){
    current.innerHTML = this.value;
}
function hide(){
    var tohide = document.getElementsByClassName("onload");
    var toshow = document.getElementById("animation");
    for (i=0, len = tohide.length; i<len; i++){
        tohide[i].style.display = "none";

    }

    toshow.style.display ="block";

    toshow.style.background = "black";

    document.body.style.background = "black";

}
```

This javascript file is responsible for any interaction on the train html webpage. The getlayers function here gets the user value from the first form and then creates that number of input fields in the second form. The hide function hides all the webpage content and shows the neural network animation.

This is what the different variations of the train page look like.



number of hidden layers select

learning rate

Activation Function

● sigmoid

○ relu

○ tanh

epochs

training data

30000

start training

3 number of hidden layers select

Hidden layer 1 Hidden layer 2 Hidden layer 3

learning rate

Activation Function

● sigmoid

○ relu

○ tanh

epochs

training data

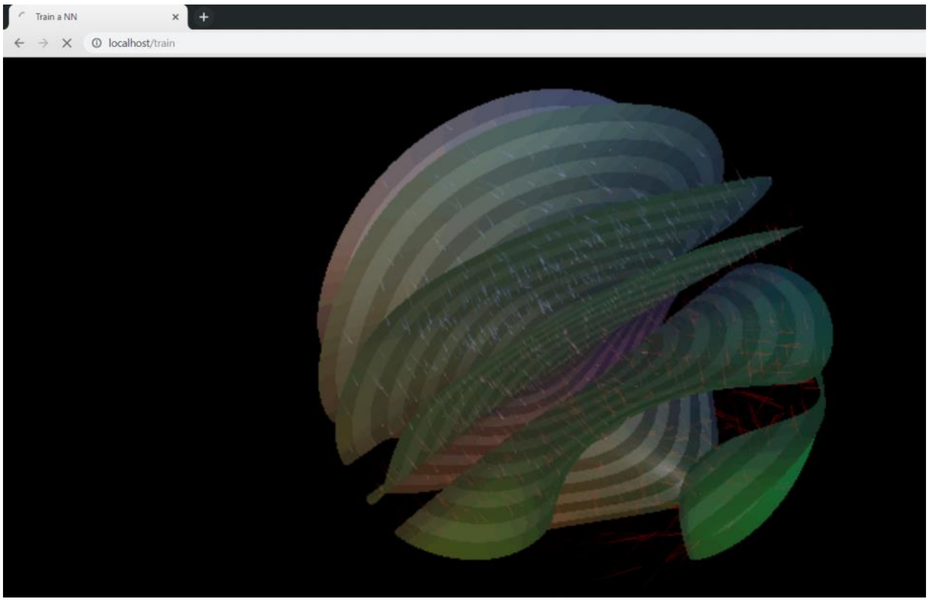
30000

start training

3 entered into the first form.

3 fields appear.

Once “start training” has been clicked.

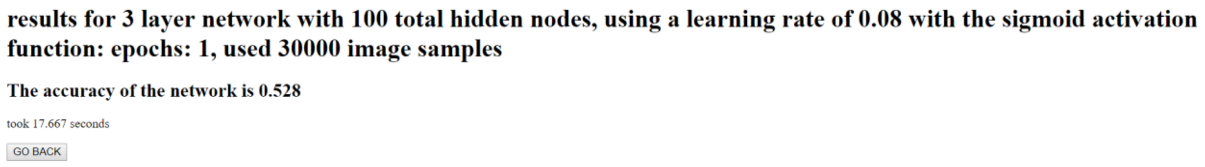


Trained.html

```
<!DOCTYPE html>
<html>
<head>
    <title>TRAINING RESULTS</title>
</head>
<body>
    <h1>results for {{ layers }} layer network with {{ totalnodes }} total hidden nodes, using a learning rate of {{ lr }} with the {{ activation }} activation function: epochs: {{ epochs }}, used {{ trainrange }} image samples</h1>

    {% if epochs > 1 %}
    <ul>
        {% for i in range(epochs) %}
        <li>ACCURACY AFTER EPOCH {{i+1}} is {{ epochacc[i] }} </li>
        {% endfor %}
    </ul>
    {% else %}
    <h2>The accuracy of the network is {{ epochacc }}</h2>
    {% endif %}

    <p>took {{ time }} seconds</p>
    <button onclick="location.href='/'">GO BACK</button>
</body>
<script type="text/javascript" src="{{ url_for('static', filename='trained.js') }}"></script>
</html>
```

This trained page will appear after the network is finished training. The "{{" brackets indicate a flask function, in this case I have passed variables from python to this html page and have used for loops and if statements to alter the page depending on the variable values, for example if the epochs are greater than 1 the page will show each epoch accuracy.

This is for 1 epoch.

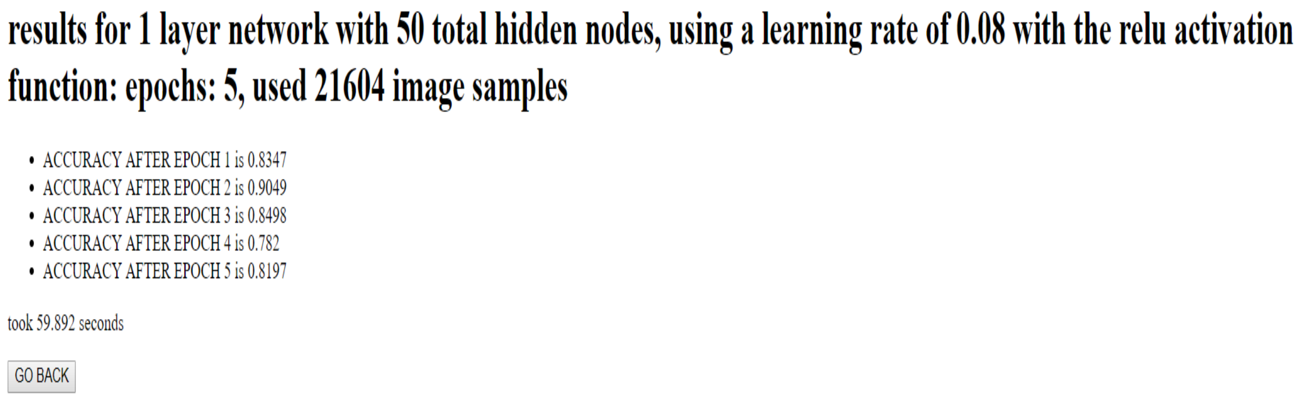


results for 3 layer network with 100 total hidden nodes, using a learning rate of 0.08 with the sigmoid activation function: epochs: 1, used 30000 image samples

The accuracy of the network is 0.528

took 17.667 seconds

This is with 5 epochs; each epoch accuracy is shown.



results for 1 layer network with 50 total hidden nodes, using a learning rate of 0.08 with the relu activation function: epochs: 5, used 21604 image samples

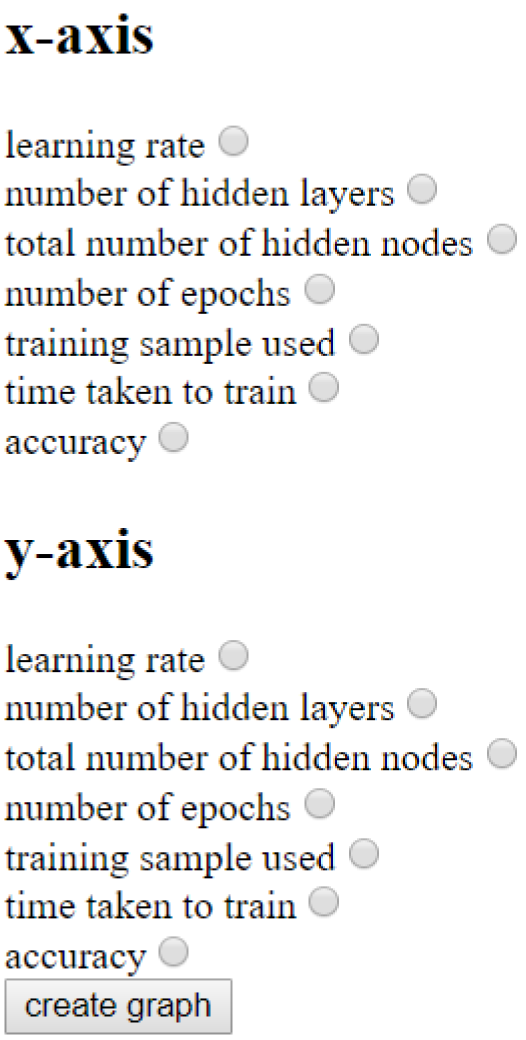
- ACCURACY AFTER EPOCH 1 is 0.8347
- ACCURACY AFTER EPOCH 2 is 0.9049
- ACCURACY AFTER EPOCH 3 is 0.8498
- ACCURACY AFTER EPOCH 4 is 0.782
- ACCURACY AFTER EPOCH 5 is 0.8197

took 59.892 seconds

Results.html

```
<!DOCTYPE html>
<html>
<head>
    <title>variable selector</title>
</head>
<body>
    <form action="{{ url_for('graph') }}" method="post">
        <h2>x-axis</h2>
        <label>learning rate</label><input type="radio" name="x" value="0" required><br>
        <label>number of hidden layers</label><input type="radio" name="x" value="1" required><br>
        <label>total number of hidden nodes</label><input type="radio" name="x" value="2" required><br>
        <label>number of epochs</label><input type="radio" name="x" value="3" required><br>
        <label>training sample used</label><input type="radio" name="x" value="4" required><br>
        <label>time taken to train</label><input type="radio" name="x" value="5" required><br>
        <label></label>accuracy<input type="radio" name="x" value="6" required><br>
        <h2>y-axis</h2>
        <label>learning rate</label><input type="radio" name="y" value="0" required><br>
        <label>number of hidden layers</label><input type="radio" name="y" value="1" required><br>
        <label>total number of hidden nodes</label><input type="radio" name="y" value="2" required><br>
        <label>number of epochs</label><input type="radio" name="y" value="3" required><br>
        <label>training sample used</label><input type="radio" name="y" value="4" required><br>
        <label>time taken to train</label><input type="radio" name="y" value="5"> required<br>
        <label></label>accuracy<input type="radio" name="y" value="6" required><br>
        <input type="submit" value="create graph">
    </form>
</body>
</html>
```

This page is where the user selects two variables to plot a graph, I have done this by having 2 sets of radio buttons where a user must select 1 button from each of the sets, this is contained in a form which uses the post method to “/graph” URL. The form value will contain the index of the user’s selection for each of the sets.



x-axis

learning rate ○
number of hidden layers ○
total number of hidden nodes ○
number of epochs ○
training sample used ○
time taken to train ○
accuracy ○

y-axis

learning rate ○
number of hidden layers ○
total number of hidden nodes ○
number of epochs ○
training sample used ○
time taken to train ○
accuracy ○

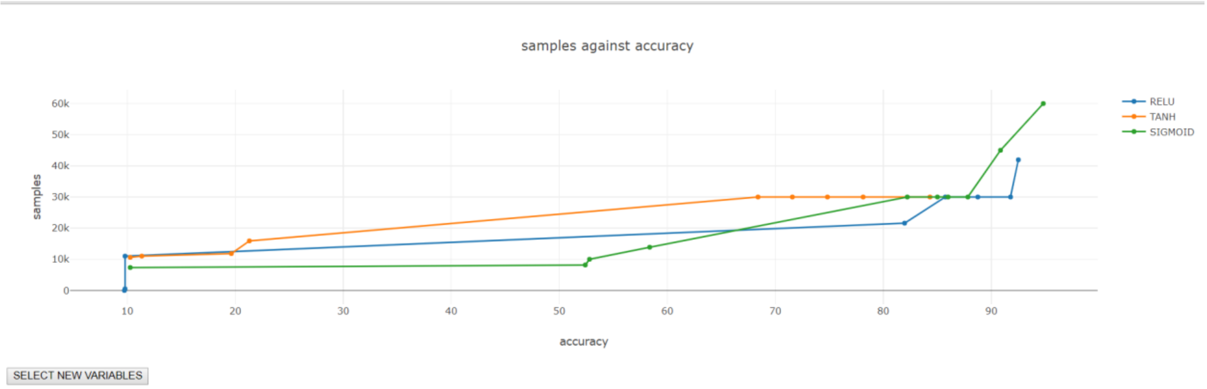
This is what the resulting page looks like.

Graph.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Graph</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
    {{ div | safe}}

    <button onclick="location.href='results'">SELECT NEW VARIABLES</button>
</body>
</html>
```

This page simply displays the html code for the graph which will be passed through to the page by python ({{div}}), the safe parameter means that flask will load the contents of the div variable. I have also linked the plotly javascript url, this is to allow the graph to be interactive.

Here is an example of one of the graphs (accuracy on x axis and number of samples on y axis)

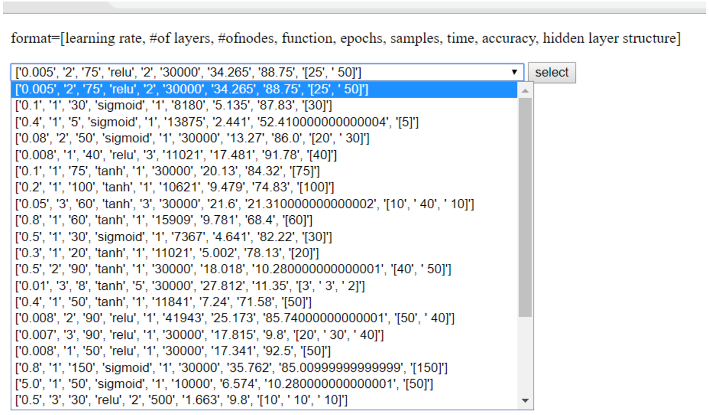


Test.html

```
<!DOCTYPE html>

<html>

<head>

    <title>NN testing</title>

    <script type="text/javascript" src="https://code.jquery.com/jquery-3.3.1.js"></script>

</head>

<body>

    <p>format=[learning rate, #of layers, #ofnodes, function, epochs, samples, time, accuracy, hidden layer
structure]</p>

    <form action="{{ url_for('draw') }}" method="post">

        <select name="network">

            {% for i in range(numnetworks) %}

            <option name="network" value="{{ i }}">{{ networks[i] }}</option>

            {% endfor %}

        </select>

    <input type="submit" value="select">

    </form>

</body>

</html>
```

This page is where the user will select one of the saved networks. To do this I have created a dropdown selection and have added options based on the passed variables. This will iterate through “numnetworks” which is a list of all saved networks (read from the text file) and add an option tag for each of the networks. This is within a form so the users selection index is sent to “/draw” using the post method.

Every network stored in the text file is shown here:



format=[learning rate, #of layers, #ofnodes, function, epochs, samples, time, accuracy, hidden layer structure]

```
['0.005', '2', '75', 'relu', '2', '30000', '34.265', '88.75', '[25', ' 50]']
['0.005', '2', '75', 'relu', '2', '30000', '34.265', '88.75', '[25', ' 50]']
['0.1', '1', '30', 'sigmoid', '1', '8180', '5.135', '87.83', '[30]']
['0.4', '1', '5', 'sigmoid', '1', '13875', '2.441', '52.410000000000004', '[5]']
['0.08', '2', '50', 'sigmoid', '1', '30000', '13.27', '86.0', '[20', ' 30]']
['0.008', '1', '40', 'relu', '3', '11021', '17.481', '91.78', '[40]']
['0.1', '1', '75', 'tanh', '1', '30000', '20.13', '84.32', '[75]']
['0.2', '1', '100', 'tanh', '1', '10621', '9.479', '74.83', '[100]']
['0.05', '3', '60', 'tanh', '3', '30000', '21.6', '21.310000000000002', '[10', ' 40', ' 10]']
['0.8', '1', '60', 'tanh', '1', '15909', '9.781', '68.4', '[60]']
['0.5', '1', '30', 'sigmoid', '1', '7367', '4.641', '82.22', '[30]']
['0.3', '1', '20', 'tanh', '1', '11021', '5.002', '78.13', '[20]']
['0.5', '2', '90', 'tanh', '1', '30000', '18.018', '10.280000000000001', '[40', ' 50]']
['0.01', '3', '8', 'tanh', '5', '30000', '27.812', '11.35', '[3', ' 3', ' 2]']
['0.4', '1', '50', 'tanh', '1', '11841', '7.24', '71.58', '[50]']
['0.008', '2', '90', 'relu', '1', '41943', '25.173', '85.74000000000001', '[50', ' 40]']
['0.007', '3', '90', 'relu', '1', '30000', '17.815', '9.8', '[20', ' 30', ' 40]']
['0.008', '1', '50', 'relu', '1', '30000', '17.341', '92.5', '[50]']
['0.8', '1', '150', 'sigmoid', '1', '30000', '35.762', '85.00999999999999', '[150]']
['5.0', '1', '50', 'sigmoid', '1', '10000', '6.574', '10.280000000000001', '[50]']
['0.5', '3', '30', 'relu', '2', '500', '1.663', '9.8', '[10', ' 10', ' 10]']
```

Page.html

```
<!DOCTYPE html>

<html>

<head>

    <title>TEST PAGE</title>

    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='page.css') }}">

</head>

<body scroll="no">

    <div id="drawscreen">

        <div id="bar"><h1 id="answer"></h1></div>

        <div id="info">

            <p>learning rate: {{ data[0] }}</p><br>

            <p>hidden layers: {{ data[1] }}</p><br>

            <p>hidden nodes: {{ data[2] }}</p><br>

            <p>hidden layers structure: {{ data[8:] }}</p><br>

            <p>activation function: {{ data[3] }}</p><br>

            <p>epochs: {{ data[4] }}</p><br>

            <p>training samples: {{ data[5] }}</p><br>

            <p>time taken: {{ data[6] }}</p><br>

            <p>accuray: {{ data[7] }}</p><br>

        </div>

        <div id="back">

            <canvas id="draw" style="border: 1px solid black;"></canvas>

            <h2>USE SPACE TO CLEAR</h2>

            <h2>USE ENTER TO RECOGNISE</h2>

            <button onclick="location.href='/'">Go home</button>

        </div>

    </div>

</body>

<script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script type="text/javascript" src="{{ url_for('static', filename='page.js') }}"></script>
```

Display the network details

This is the page where the user is able to test the network with their own input. To do this I have used the canvas tag, the user will be able to hand draw an image on this canvas and then use the enter key to pass the image through the network. All of this is controlled by the javascript.

Page.js

```
var canvas = document.getElementById("draw");
var barh  = document.getElementById("bar").offsetHeight;
canvas.width = 300;
canvas.height = 300;
const context = canvas.getContext("2d");
context.lineWidth = 18;
var down = false;
canvas.addEventListener("mousemove", draw);
canvas.addEventListener("mousedown", function() {
    down = true;
    context.beginPath();
    context.moveTo(xpos, ypos);
    canvas.addEventListener("mousemove", draw)
});
```

This controls how the user draws on the canvas. As the user clicks the mouse and moves it a new path is created in the canvas.

```
canvas.addEventListener("mouseup", function() {down = false});
function draw(e) {
    xpos = (e.clientX - canvas.offsetLeft);
    ypos = (e.clientY - canvas.offsetTop);
    if(down == true){
        context.lineTo(xpos, ypos);
        context.stroke();}}
function ge(){
    var imgdata = context.getImageData(0, 0, canvas.width, canvas.h
    return imgdata.data;}
```

This function gets the pixel data for the canvas.

```
function load(){
    document.getElementById("answer").innerHTML = "loading";
}
document.body,onkeyup = function(e){
    if(e.keyCode == 13){
```

1

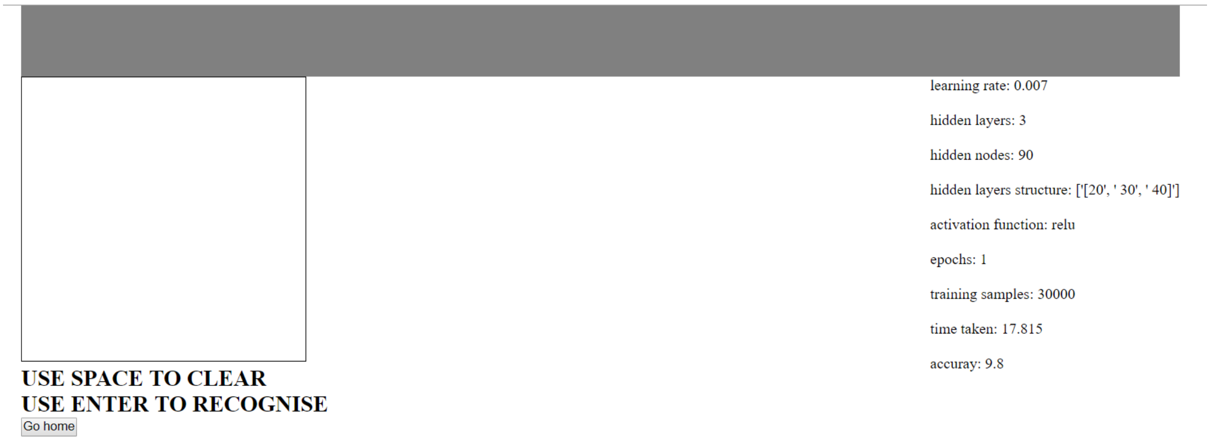
```
        load();
        var imgd = ge();
        var i =3;
        var newd = [];
        for (; i<imgd.length;){
            newd.push(imgd[i]);
            i = i+4;}
        $.post("/givenumber", {"img":newd}, function(data){
            document.getElementById("answer").innerHTML = "PREDICTION " + data.number
        });}}
document.body.onkeyup = function(e){
    if(e.keyCode == 32){
        context.clearRect(0, 0, canvas.width, canvas.height);
    }}
var down = false;
```

This writes “loading” to the answer tag until the prediction is written

When the space bar is pressed the canvas is cleared.

1 This function is run when the enter key is pressed. Firstly, the canvas pixel data is obtained and stored. Each pixel has 4 different values Red green blue and alpha, for this task I am only interested in the alpha value for each pixel. To get the alpha pixel value for each pixel the canvas data array is iterated through and every 4th value is appended onto a new list. Next a post request is sent to the server, this request contains the new pixel value array. The returned value from the server is written to the answer tag where the user will be able to see.

This is what the page looks like:



learning rate: 0.007

hidden layers: 3

hidden nodes: 90

hidden layers structure: ['[20', ' 30', ' 40]']

activation function: relu

epochs: 1

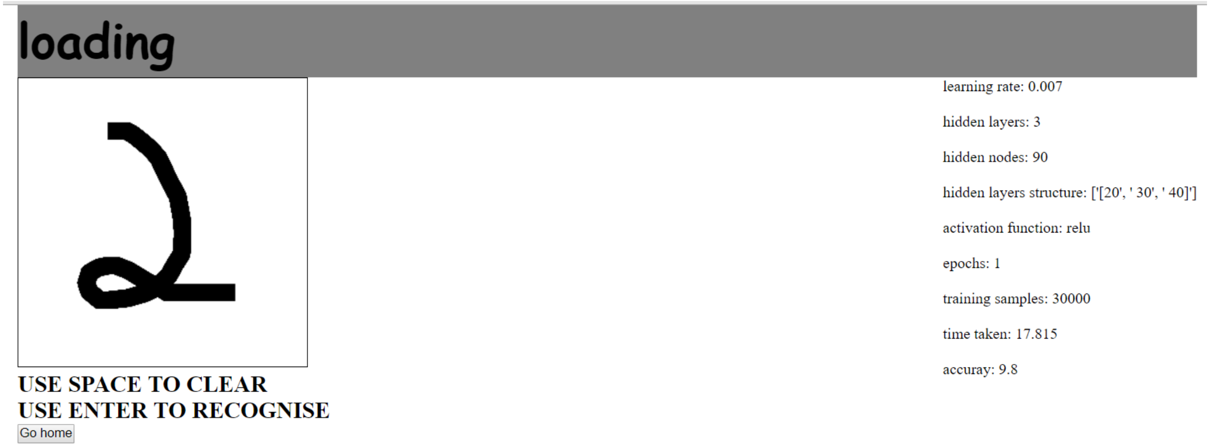
training samples: 30000

time taken: 17.815

accuray: 9.8

USE SPACE TO CLEAR
USE ENTER TO RECOGNISE

After something is drawn in the canvas and the enter key is pressed:



loading

learning rate: 0.007

hidden layers: 3

hidden nodes: 90

hidden layers structure: ['[20', ' 30', ' 40]']

activation function: relu

epochs: 1

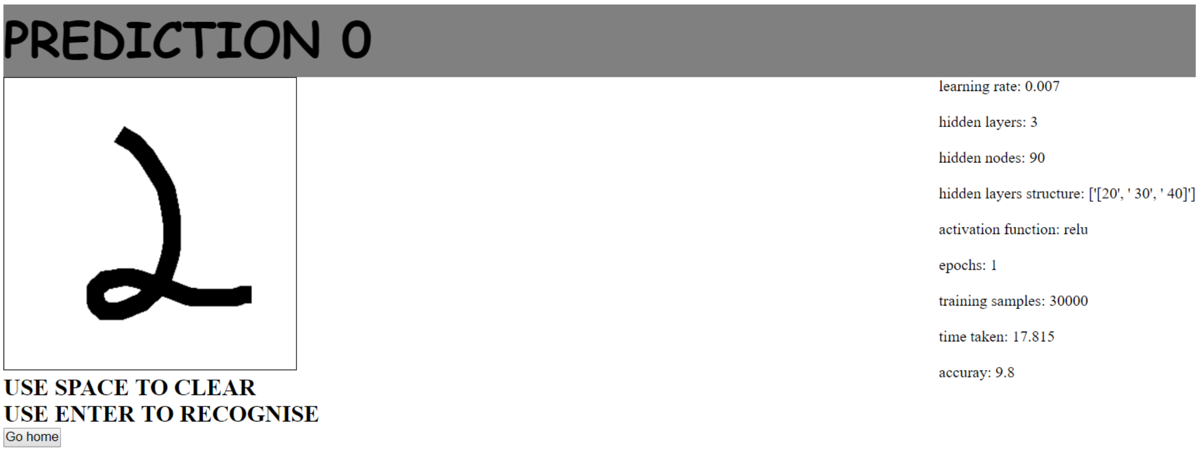
training samples: 30000

time taken: 17.815

accuray: 9.8

USE SPACE TO CLEAR
USE ENTER TO RECOGNISE

Once its finished loading:



PREDICTION 0

learning rate: 0.007

hidden layers: 3

hidden nodes: 90

hidden layers structure: ['[20', ' 30', ' 40]']

activation function: relu

epochs: 1

training samples: 30000

time taken: 17.815

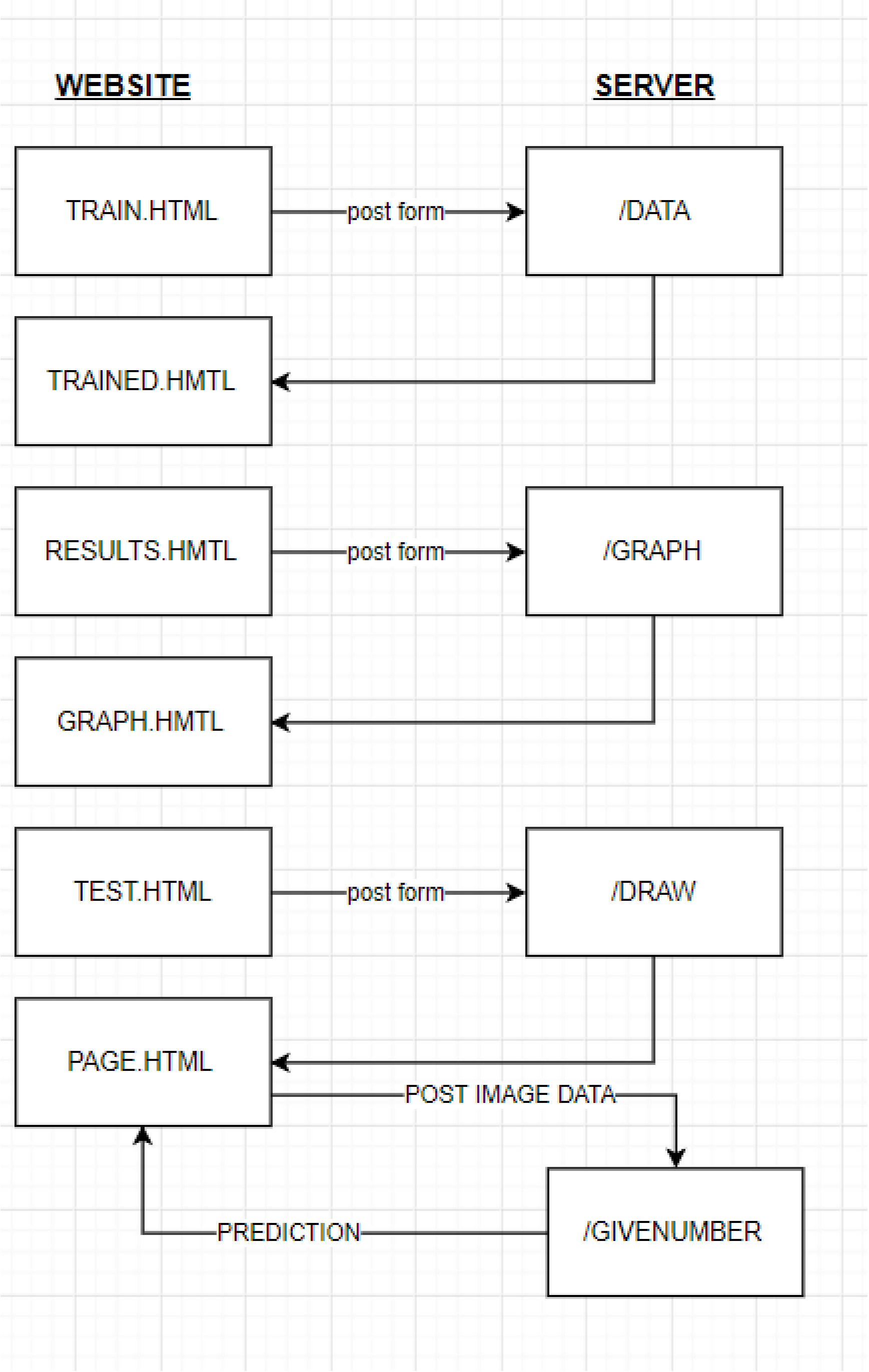
accuray: 9.8

USE SPACE TO CLEAR
USE ENTER TO RECOGNISE

(the prediction is wrong because this network has an accuracy of 9%)

Server

Here is a diagram illustrating server requests and how the server responds to these requests



This all takes place on the “**main_server”** python file.

```
from flask import Flask, render_template, url_for, request, redirect, jsonify
from NN_MULTIPLE_LAYERS import main
from NN_MULTIPLE_LAYERS import runimg
from plotgraph import divplot
from write import writetotxt
import numpy as np
server = Flask(__name__)
@server.route("/")
def index():
    return render_template("home.html")

@server.route("/train")
def train():
    return render_template("train.html")

@server.route("/data", methods =["POST", "GET"])
def data():
    formdata = request.form
    innodes = 784
    onodes = 10
    hlayers = formdata.getlist("hiddenlayers")
    hlayers = [int(i) for i in hlayers]
    lr = float(formdata.getlist("learningrate")[0])
    activation = formdata.getlist("activation")[0]
    epochs = int(formdata.getlist("epochs")[0])
    trainrange = int(formdata.getlist("trainrange")[0])
    final_values = main(innodes, onodes, lr, hlayers, activation, epochs, trainrange)
    writetotxt("nnresults.txt", final_values)
    return render_template("trained.html", layers=len(hlayers), totalnodes=final_values[2], lr=lr, activation=activation, epochs=epochs, epochacc=final_values[7], trainrange=trainrange, time=final_values[6])
@server.route("/results")
def results():
```

Here I have imported all the different python functions so that they can be used by the server.

The server route decorator dictates what happens when a request is made for the url. “/” signals the index or home page and in this case the index function is ran which simply returns the home.html webpage.

In the data function, the form data is stored into separate variables and the main function is called to train the network. The network results are written to the text file and are also passed through to the trained html page.

```
    return render_template("results.html")


@server.route("/graph", methods=["POST"])
def graph():
    graphdata = request.form
    xvar = int(graphdata.getlist("x")[0])
    yvar = int(graphdata.getlist("y")[0])
    x = divplot(xvar, yvar)
    return render_template("graph.html", div=x)
```

Here the form data is stored into a variable. This data is an index value, and because the network results format is maintained throughout the program it is easy to select the correct variables and plot the graph

```
@server.route("/test")
def test():
    with open("nnresults.txt", "r")as file:
        txtdata = file.readlines()
    txtdata = [i.strip("\n").split(",") for i in txtdata]
    return render_template("test.html", numnetworks=len(txtdata),networks=txtdata)
```

Here the text file is read and split so that it is in multidimensional array where each element is a different network. This array is then passed to the test html webpage

```
@server.route("/draw", methods=["POST"])
def draw():
    global option
    option = int(request.form.getlist("network")[0])
    with open("nnresults.txt", "r")as file:
        txtdata = file.readlines()
    txtdata = [i.strip("\n").split(",") for i in txtdata]
    return render_template("page.html", data=txtdata[option])
```

This function receives the form request from the test page and creates the draw page with the network details shown.

```
@server.route('/givennumber', methods=['POST', "GET"])
def givennumber():
    news = request.form.getlist("img[]")
    n = str(runimg(news, option))
    return jsonify(number=n)
if __name__ == "__main__":
    server.run(debug=True, host="0.0.0.0", port=80)
```

Here the neural network is loaded from the pickle file using the global option variable(the index is the same so the correct network is loaded). The user image data is ran through the network and the prediction is ran returned to the webpage in the json format

An Improvement

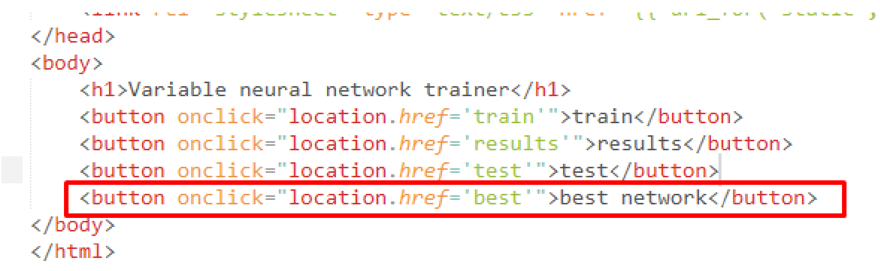
After completing the previously explained code, I realised an extremely useful feature for this program would be to have an option where the user can immediately view the attributes of the neural network with the greatest accuracy. I did this by creating the following html file:

```
<!DOCTYPE html>

<html>

<head>

    <title>best network</title>

</head>

<body>

    <h2>The best network is</h2>

    <p>accuray: {{ data[7] }}</p><br>

    <p>learning rate: {{ data[0] }}</p><br>

    <p>hidden layers: {{ data[1] }}</p><br>

    <p>hidden nodes: {{ data[2] }}</p><br>

    <p>hidden layers structure: {{ data[8:] }}</p><br>

    <p>activation function: {{ data[3] }}</p><br>

    <p>epochs: {{ data[4] }}</p><br>

    <p>training samples: {{ data[5] }}</p><br>

    <p>time taken: {{ data[6] }}</p><br>

</body>

</html>
```

Network information is passed as an array named “data”.

This simply displays each attribute of the network on a separate line.

I also created a new button on the homepage which would link to a new route.



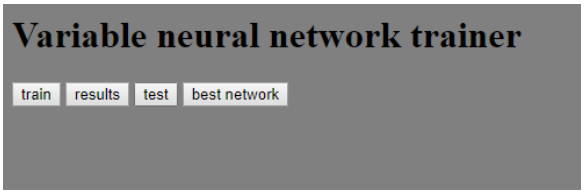
```
</head>
<body>
    <h1>Variable neural network trainer</h1>
    <button onclick="location.href='train'">train</button>
    <button onclick="location.href='results'">results</button>
    <button onclick="location.href='test'">test</button>
    <button onclick="location.href='best'">best network</button>
</body>
</html>
```

The new button labelled “best network” links to the route “/best”. As a result, a new server route was created in the main server fie.

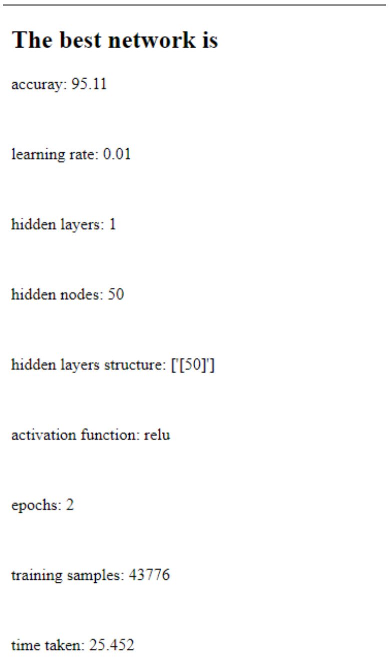
```
@server.route('/best')

def best():

    with open("nnresults.txt", "r")as file:

        txtdata = file.readlines()

    txtdata = [i.strip("\n").split(",") for i in txtdata]

    biggest =0

    for i in txtdata:

        if float(i[7]) > biggest:

            biggest = float(i[7])

            best = i

    return render_template("best.html", data=best)
```

This shows the function that runs when the webpage requests "/best". The algorithm reads the network text file and then iterates through each of the network's accuracies, updating the "biggest" variable with the network with the greatest accuracy. Lastly, the best.html file is rendered and the network with the greatest accuracy is passed through to be displayed.

This is the view of the new homepage:


Variable neural network trainer

This is the best.html page once the best network button is clicked:


The best network is

accuray: 95.11

learning rate: 0.01

hidden layers: 1

hidden nodes: 50

hidden layers structure: ['[50]']

activation function: relu

epochs: 2

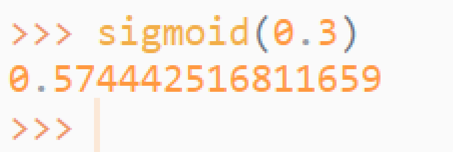
training samples: 43776

time taken: 25.452

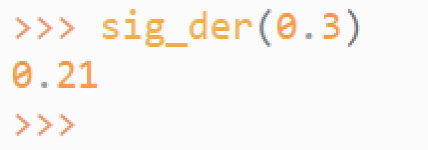
Testing

1. Activation function testing.

- Sigmoid test, I run the value 0.3, the expected outcome is 0.57444 which is obtained from a calculator. The derivative is expected to be 0.21.



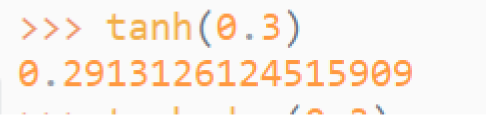
```
>>> sigmoid(0.3)
0.574442516811659
>>>
```



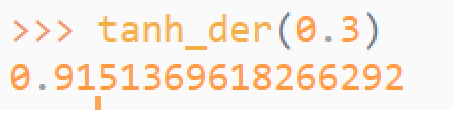
```
>>> sig_der(0.3)
0.21
>>>
```

The correct values were returned.

- Tanh test, again the value 0.3 is used as a test, from a calculator the expected values are 0.29131 and 0.91514 for the derivative.



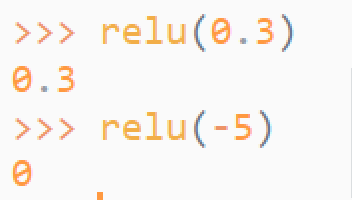
```
>>> tanh(0.3)
0.2913126124515909
```



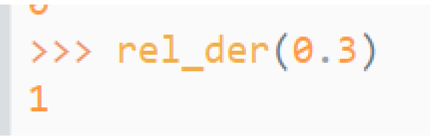
```
>>> tanh_der(0.3)
0.9151369618266292
```

The correct values were returned.

- Relu test, tested using 0.3, expected outcome is 0.3 and 1 for the derivative. I have also tested it with -5 because relu is meant to convert any values less than 0 to 0.



```
>>> relu(0.3)
0.3
>>> relu(-5)
0
```

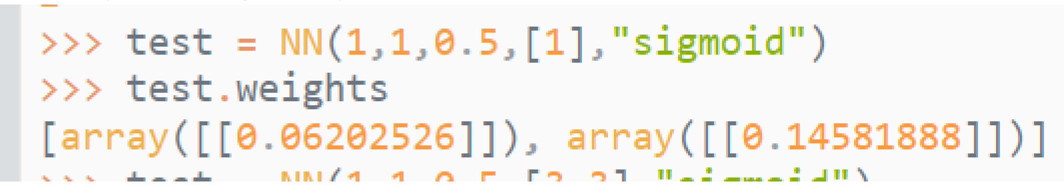


```
>>> rel_der(0.3)
1
```

Everything is working correctly.

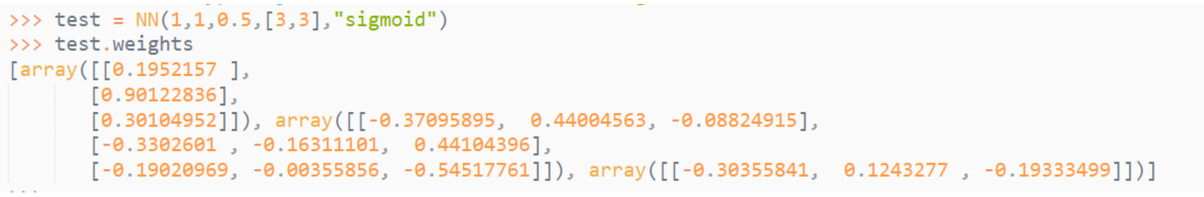
2. Weights initialisation testing,

- To test this, I created a neural network object with 1 hidden layer with 1 node and 1 input and output node. I expect the weights array to have 2 elements.



```
>>> test = NN(1,1,0.5,[1],"sigmoid")
>>> test.weights
[array([[0.06202526]]), array([[0.14581888]])]
```

This is what I expected, I also tested this with a network with 2 hidden layers where each hidden layer has 3 nodes. There is 1 input and 1 output node. I expect and array with one three element array followed by a 3x3 matrix and then another three-element array.

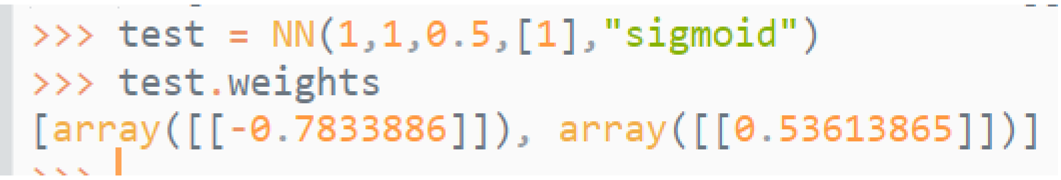


```
>>> test = NN(1,1,0.5,[3,3],"sigmoid")
>>> test.weights
[array([[0.1952157 ],
       [0.90122836],
       [0.30104952]]), array([[-0.37095895,  0.44004563, -0.08824915],
       [-0.3302601 , -0.16311101,  0.44104396],
       [-0.19020969, -0.00355856, -0.54517761]]), array([[-0.30355841,  0.1243277 , -0.19333499]])]
```

This is the correct output.

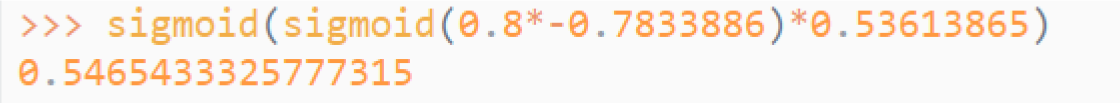
3. Testing the forward propagation.

- To test this, I create a simple network with 1 input, hidden and output node. Next, I print the weights of the network.



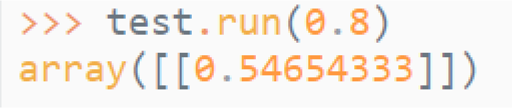
```
>>> test = NN(1,1,0.5,[1],"sigmoid")
>>> test.weights
[array([[-0.7833886]]), array([[0.53613865]])]
>>> 
```

I will test the run function using a value of 0.8. Using the printed weights, I can manually work out the output of the network. The expected outcome will be sigmoid(sigmoid(0.8*-0.7833886)*0.53613865)
I calculated this using my program.



```
>>> sigmoid(sigmoid(0.8*-0.7833886)*0.53613865)
0.5465433325777315
```

Next, I called the run function with the parameter of 0.8.

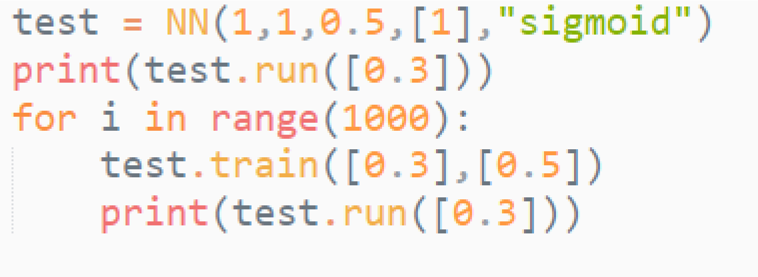


```
>>> test.run(0.8)
array([[0.54654333]])
```

The run function returned the same value as what I calculated, meaning it is working correctly.

4. Testing the back-propagation algorithm.

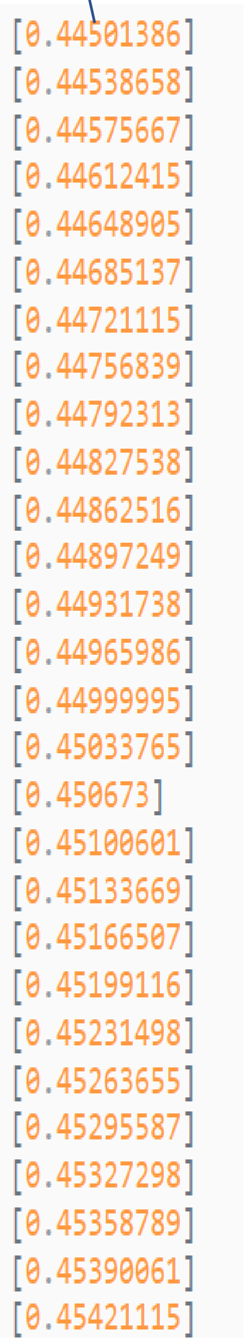
- Using the same network as the previous test, I will train the network to convert all values of 0.3 to 0.5, this will be done calling the train function with 0.3 as the input and 0.5 as the target. I will check how close to 0.3 the output of the network is every train call by calling the run function. This is done by using a for loop:



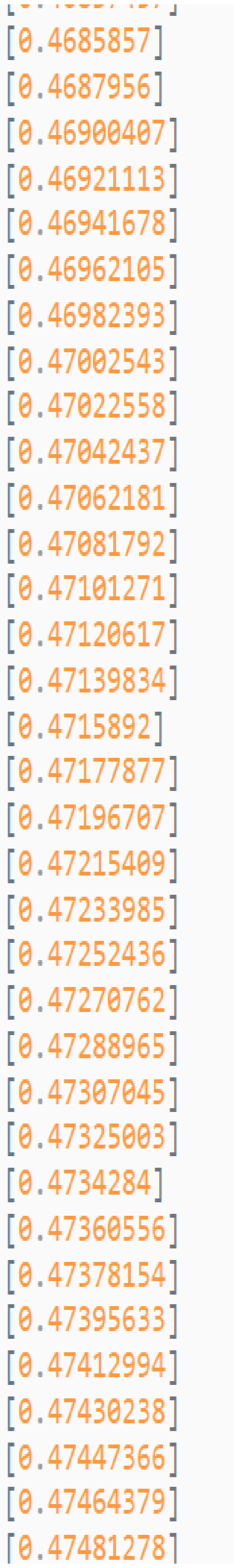
```
test = NN(1,1,0.5,[1],"sigmoid")
print(test.run([0.3]))
for i in range(1000):
    test.train([0.3],[0.5])
    print(test.run([0.3]))
```

This was the output (only show few print statements as 1000 is too large)

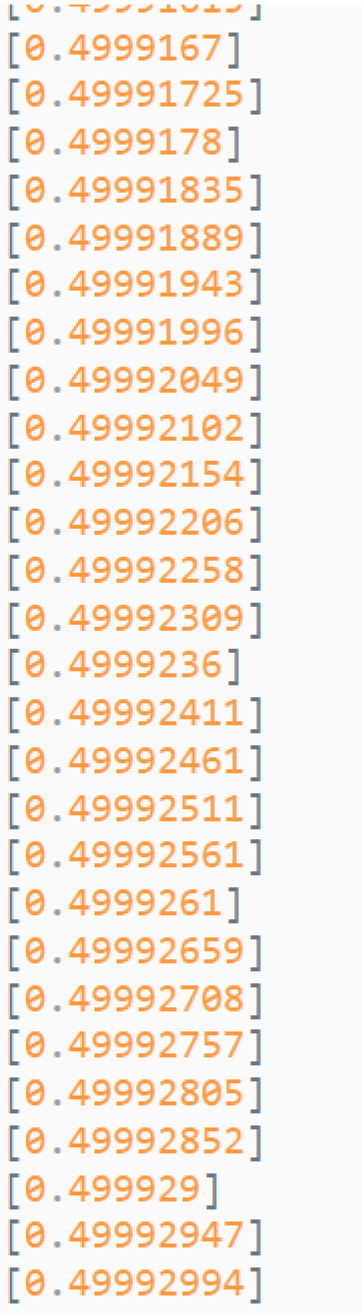
This is the initial value before any training



```
[0.44501386]
[0.44538658]
[0.44575667]
[0.44612415]
[0.44648905]
[0.44685137]
[0.44721115]
[0.44756839]
[0.44792313]
[0.44827538]
[0.44862516]
[0.44897249]
[0.44931738]
[0.44965986]
[0.44999995]
[0.45033765]
[0.450673]
[0.45100601]
[0.45133669]
[0.45166507]
[0.45199116]
[0.45231498]
[0.45263655]
[0.45295587]
[0.45327298]
[0.45358789]
[0.45390061]
[0.45421115]
```



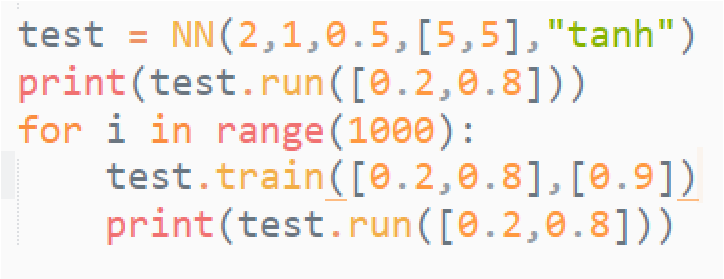
```
[0.4685857]
[0.4687956]
[0.46900407]
[0.46921113]
[0.46941678]
[0.46962105]
[0.46982393]
[0.47002543]
[0.47022558]
[0.47042437]
[0.47062181]
[0.47081792]
[0.47101271]
[0.47120617]
[0.47139834]
[0.4715892]
[0.47177877]
[0.47196707]
[0.47215409]
[0.47233985]
[0.47252436]
[0.47270762]
[0.47288965]
[0.47307045]
[0.47325003]
[0.4734284]
[0.47360556]
[0.47378154]
[0.47395633]
[0.47412994]
[0.47430238]
[0.47447366]
[0.47464379]
[0.47481278]
```



```
[0.4999167]
[0.49991725]
[0.4999178]
[0.49991835]
[0.49991889]
[0.49991943]
[0.49991996]
[0.49992049]
[0.49992102]
[0.49992154]
[0.49992206]
[0.49992258]
[0.49992309]
[0.4999236]
[0.49992411]
[0.49992461]
[0.49992511]
[0.49992561]
[0.4999261]
[0.49992659]
[0.49992708]
[0.49992757]
[0.49992805]
[0.49992852]
[0.499929]
[0.49992947]
[0.49992994]
```

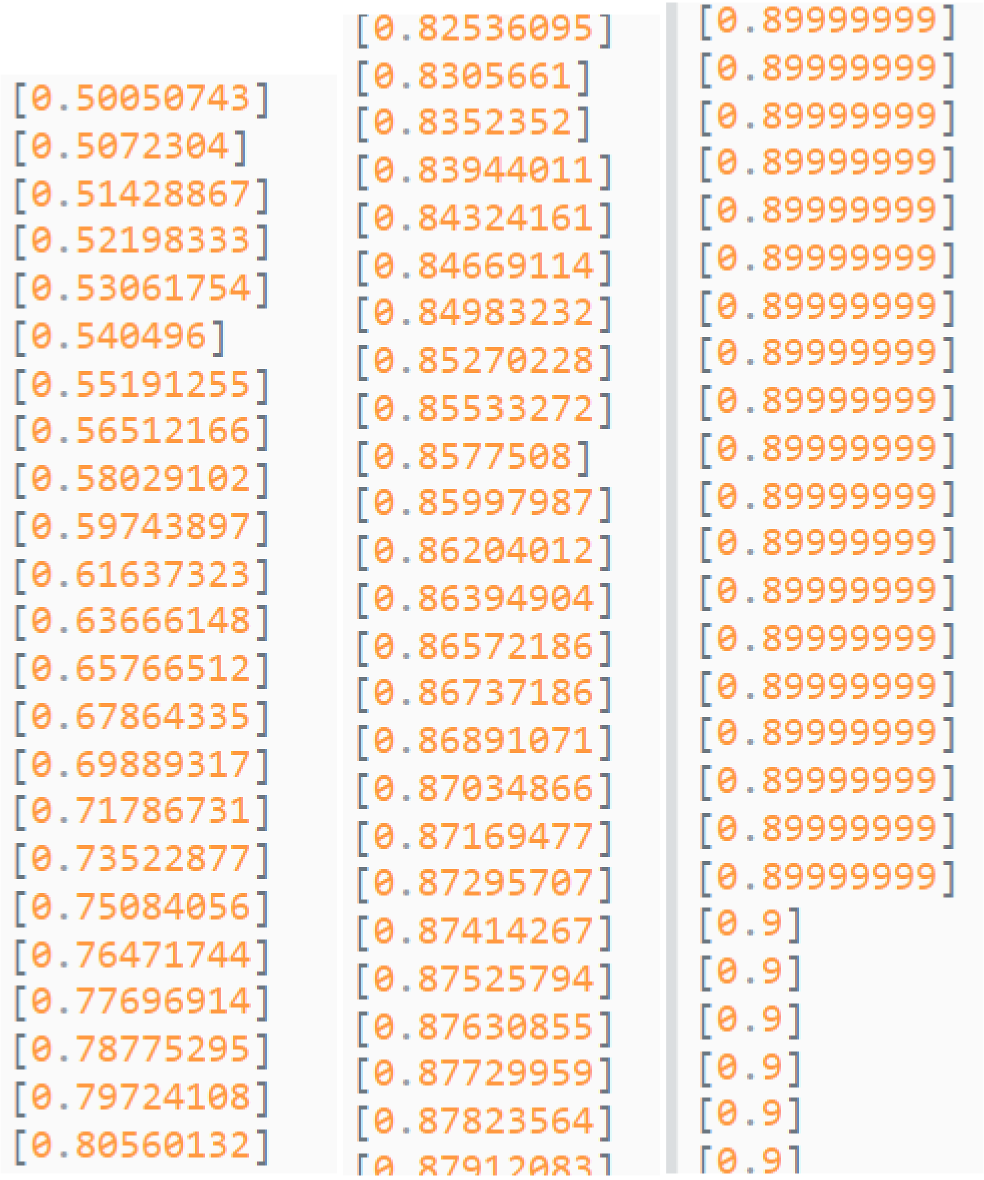
As you can see, the output is converging to 0.5 meaning that the back-propagation algorithm is working.

To make sure that the back-propagation algorithm is woriking perfectly I will test it one more time using the tanh activation function. Also, I will have 3 input vectors, 2 hidden layers (hidden layer 1 with 5 hidden nodes, hidden layer 2 with 5 hidden nodes) and an output layer with a single node. For this network I will try to convert the vector [0.2,0.8] to a value of [0.9]. Using the same type of for loop:



```
test = NN(2,1,0.5,[5,5],"tanh")
print(test.run([0.2,0.8]))
for i in range(1000):
    test.train([0.2,0.8],[0.9])
    print(test.run([0.2,0.8]))
```

This is the ouput (1st print statement being the untrained output)



```
[0.50050743]
[0.5072304]
[0.51428867]
[0.52198333]
[0.53061754]
[0.540496]
[0.55191255]
[0.56512166]
[0.58029102]
[0.59743897]
[0.61637323]
[0.63666148]
[0.65766512]
[0.67864335]
[0.69889317]
[0.71786731]
[0.73522877]
[0.75084056]
[0.76471744]
[0.77696914]
[0.78775295]
[0.79724108]
[0.80560132]
```

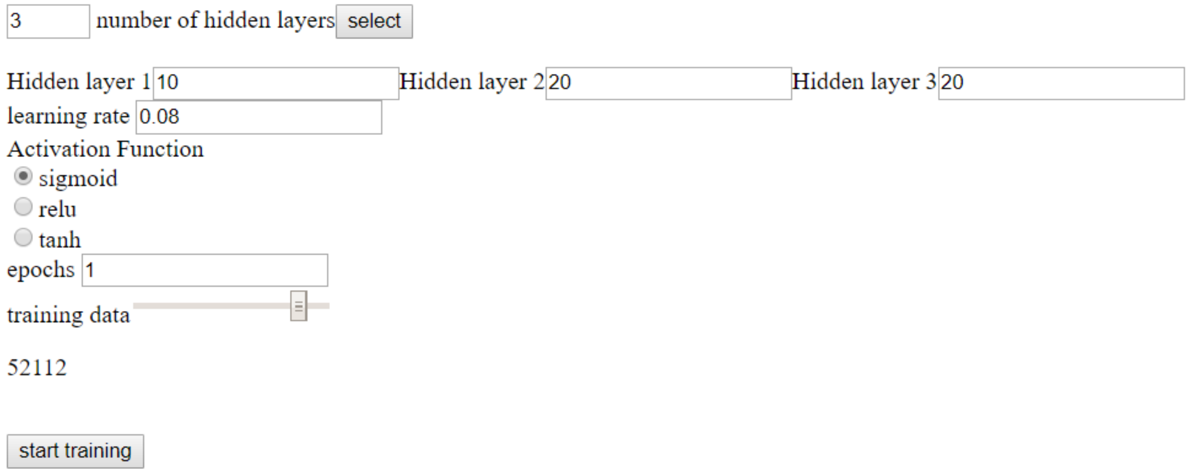
```
[0.82536095]
[0.8305661]
[0.8352352]
[0.83944011]
[0.84324161]
[0.84669114]
[0.84983232]
[0.85270228]
[0.85533272]
[0.8577508]
[0.85997987]
[0.86204012]
[0.86394904]
[0.86572186]
[0.86737186]
[0.86891071]
[0.87034866]
[0.87169477]
[0.87295707]
[0.87414267]
[0.87525794]
[0.87630855]
[0.87729959]
[0.87823564]
[0.87912083]
```

```
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.89999999]
[0.9]
[0.9]
[0.9]
[0.9]
[0.9]
[0.9]
```

This network actually manages to converge to exactly 0.9, and a lot faster than the sigmoid test. This shows that the back-propagation is definitely working correctly.

5. Testing the server is handling parameters correctly.

- For the train page, once the user has entered their values the trained page should be loaded and should provide a description of the network (should be the same as the user input).



3 number of hidden layers select

Hidden layer 1 10 Hidden layer 2 20 Hidden layer 3 20

learning rate 0.08

Activation Function

● sigmoid
○ relu
○ tanh

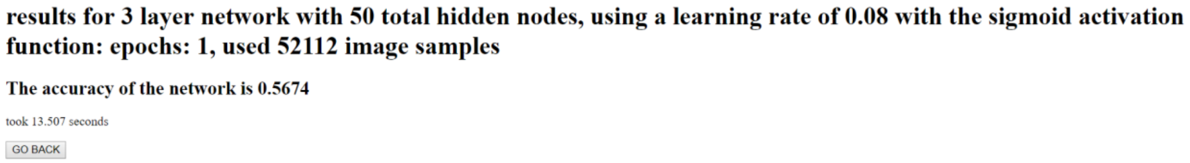
epochs 1

training data

52112

start training

This is the input I used, I expect the resulting page to say "results for 3 layer network with 50 total hidden nodes, using a learning rate of 0.08 with the sigmoid activation function: epochs: 1, used 52112 image samples". This is the page which is produced:



results for 3 layer network with 50 total hidden nodes, using a learning rate of 0.08 with the sigmoid activation function: epochs: 1, used 52112 image samples

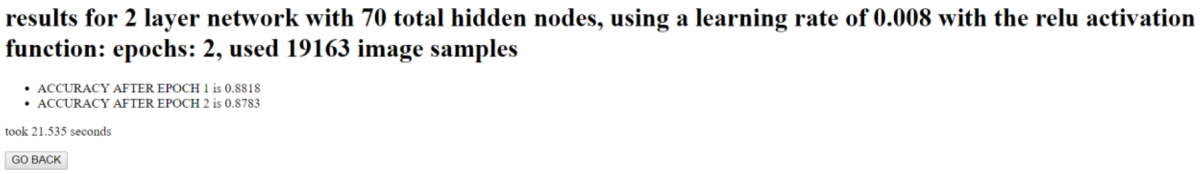
The accuracy of the network is 0.5674

took 13.507 seconds

GO BACK

Everything is as expected meaning the server is training the network correctly and the network parameters are being passed correctly.

I tested this with using the other 2 activation functions and using multiple epochs to test that the accuracy for each epoch is being shown, and that there are no issues when using the different activation functions.

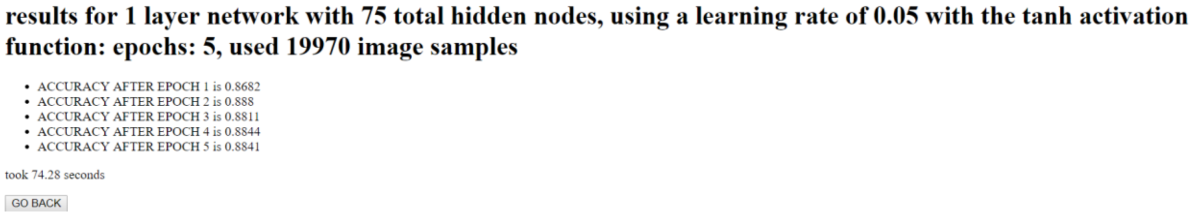


results for 2 layer network with 70 total hidden nodes, using a learning rate of 0.008 with the relu activation function: epochs: 2, used 19163 image samples

- ACCURACY AFTER EPOCH 1 is 0.8818
- ACCURACY AFTER EPOCH 2 is 0.8783

took 21.535 seconds

GO BACK



results for 1 layer network with 75 total hidden nodes, using a learning rate of 0.05 with the tanh activation function: epochs: 5, used 19970 image samples

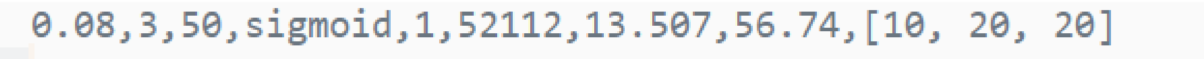
- ACCURACY AFTER EPOCH 1 is 0.8682
- ACCURACY AFTER EPOCH 2 is 0.888
- ACCURACY AFTER EPOCH 3 is 0.8811
- ACCURACY AFTER EPOCH 4 is 0.8844
- ACCURACY AFTER EPOCH 5 is 0.8841

took 74.28 seconds

GO BACK

This shows that all the activation functions are working correctly and the epochs are being displayed. The network is also being trained correctly because there is a realistic accuracy being calculated.

6. Check that the network is written correctly to the text file. From the previous test I expect a line in the "nnresults" text file containing "0.08,3,50,sigmoid,1,52112,13.507,56.74,[10, 20, 20]".
   This is what the text file shows:

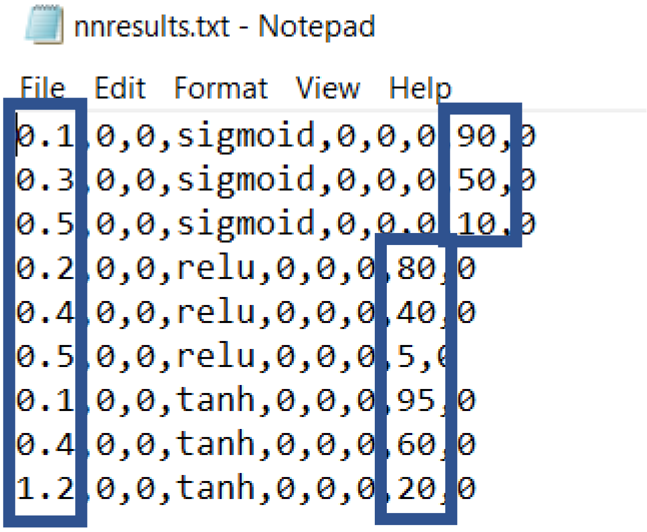


```
0.08,3,50,sigmoid,1,52112,13.507,56.74,[10, 20, 20]
```

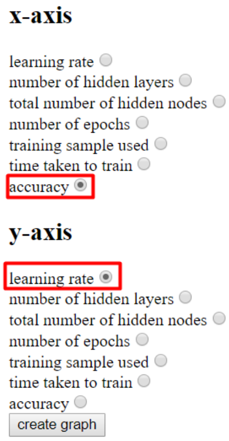
   Everything is written to the file correctly.

7. This test is to make sure that the correct values are chosen from the text file when plotting the graph.
   To do this I have created a dummy text file.
   The format of the neural network text files is:
   learning rate, number of hidden layers, total hidden nodes, activation function, epochs, training samples, time taken, accuracy, hidden layer structure.

   I have copied this format but have set everything to 0 apart from the learning rate, activation function and accuracy. I will test the graph function by plotting accuracy on the x axis, and learning rate on the y axis. From the screen shot, the left highlighted column shows the y co-ordinates (learning rate) and the right highlighted column shows the corresponding x co-ordinates (accuracy). Each of these co-ordinates should be plotted on the graph.



```
0.1,0,0,sigmoid,0,0,0,90,0
0.3,0,0,sigmoid,0,0,0,50,0
0.5,0,0,sigmoid,0,0,0,10,0
0.2,0,0,relu,0,0,0,80,0
0.4,0,0,relu,0,0,0,40,0
0.5,0,0,relu,0,0,0,5,0
0.1,0,0,tanh,0,0,0,95,0
0.4,0,0,tanh,0,0,0,60,0
1.2,0,0,tanh,0,0,0,20,0
```



x-axis

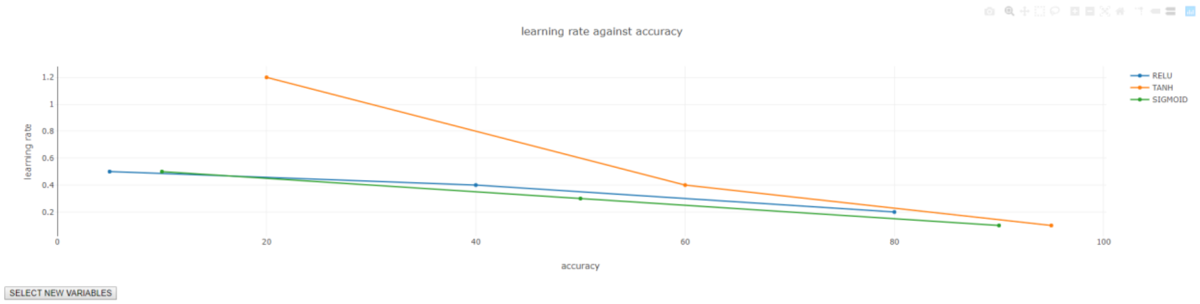
learning rate ○
number of hidden layers ○
total number of hidden nodes ○
number of epochs ○
training sample used ○
time taken to train ○
accuracy ●

y-axis

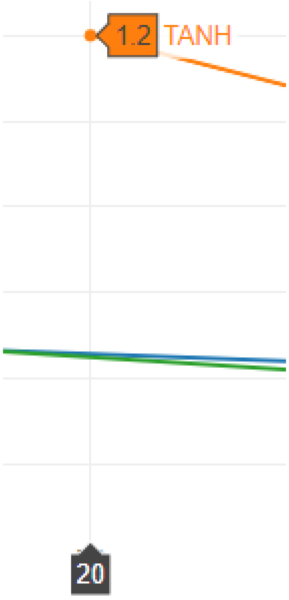
learning rate ●
number of hidden layers ○
total number of hidden nodes ○
number of epochs ○
training sample used ○
time taken to train ○
accuracy ○

   The screenshot on the right shows that I have selected accuracy to be on the x axis and learning rate on the y axis.

This screenshot shows the graph which is produced.

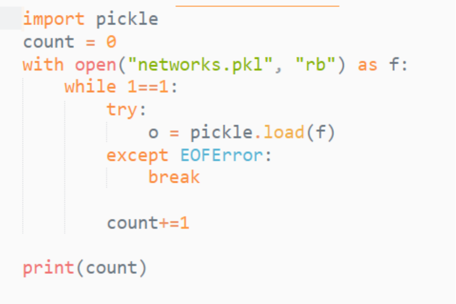


When each point on the trace is hovered over the x and y value is shown, for example when I hover over the final point of the tanh trace:

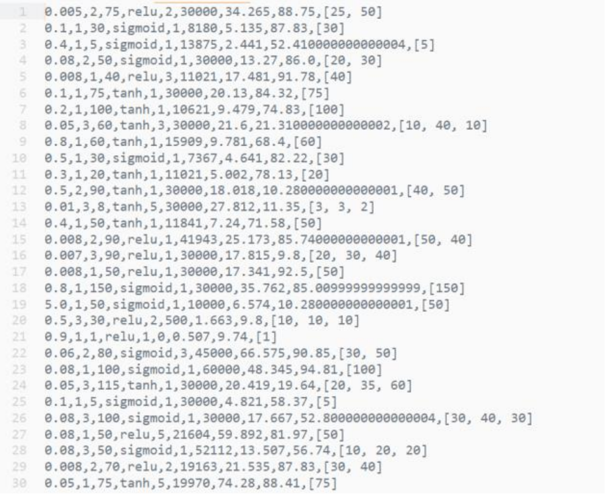


This is the correct values which means that the graph has been plotted correctly.

8. This is to check that the user image is being formatted correctly and that the correct neural network object is loaded from the pickle file.
   Firstly, because when the neural network is loaded from the pickle file it is referenced by its index in the file, this means that the number of networks in the text file and pickle should be the same or the wrong one will be loaded. I created the following script.

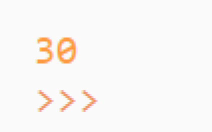


```
import pickle
count = 0
with open("networks.pkl", "rb") as f:
    while 1==1:
        try:
            o = pickle.load(f)
        except EOFError:
            break

        count+=1

print(count)
```



```
0.005,2,75,relu,2,30000,34.265,88.75,[25, 50]
0.1,1,30,sigmoid,1,8180,5.135,87.83,[30]
0.4,1,5,sigmoid,1,13875,2.441,52.410000000000004,[5]
0.08,2,50,sigmoid,1,30000,13.27,86.0,[20, 30]
0.008,1,40,relu,3,11021,17.481,91.78,[40]
0.1,1,75,tanh,1,30000,20.13,84.32,[75]
0.2,1,100,tanh,1,10621,9.479,74.83,[100]
0.05,3,60,tanh,3,30000,21.6,21.310000000000002,[10, 40, 10]
0.8,1,60,tanh,1,15909,9.781,68.4,[60]
0.5,1,30,sigmoid,1,7367,4.641,82.22,[30]
0.3,1,20,tanh,1,11021,5.002,78.13,[20]
0.5,2,90,tanh,1,30000,18.018,10.280000000000001,[40, 50]
0.01,3,8,tanh,5,30000,27.812,11.35,[3, 3, 2]
0.4,1,50,tanh,1,11841,7.24,71.58,[50]
0.008,2,90,relu,1,41943,25.173,85.74000000000001,[50, 40]
0.007,3,90,relu,1,30000,17.815,9.8,[20, 30, 40]
0.008,1,50,relu,1,30000,17.341,92.5,[50]
0.8,1,150,sigmoid,1,30000,35.762,85.00999999999999,[150]
5.0,1,50,sigmoid,1,10000,6.574,10.280000000000001,[50]
0.5,3,30,relu,2,500,1.663,9.8,[10, 10, 10]
0.9,1,1,relu,1,0,0.507,9.74,[1]
0.06,2,80,sigmoid,3,45000,66.575,90.85,[30, 50]
0.08,1,100,sigmoid,1,60000,48.345,94.81,[100]
0.05,3,115,tanh,1,30000,20.419,19.64,[20, 35, 60]
0.1,1,5,sigmoid,1,30000,4.821,58.37,[5]
0.08,3,100,sigmoid,1,30000,17.667,52.800000000000004,[30, 40, 30]
0.08,1,50,relu,5,21604,59.892,81.97,[50]
0.08,3,50,sigmoid,1,52112,13.507,56.74,[10, 20, 20]
0.008,2,70,relu,2,19163,21.535,87.83,[30, 40]
0.05,1,75,tanh,5,19970,74.28,88.41,[75]
```

The screenshot on the right shows that there are 30 networks stored in the text file, when I run the script, I would expect an output of 30.

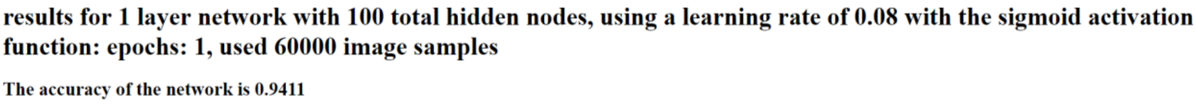


```
30
>>>
```

30 was outputted so the networks are stored correctly within the pickle file.

Next, I have to test whether the user image is being formatted correctly, to do this I created 2 new networks, one with an intentional low accuracy, and the other with a high accuracy.

The high accuracy network is:

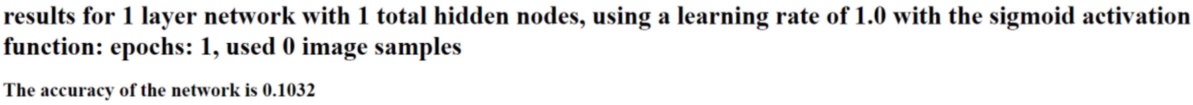


results for 1 layer network with 100 total hidden nodes, using a learning rate of 0.08 with the sigmoid activation function: epochs: 1, used 60000 image samples

The accuracy of the network is 0.9411

Has an accuracy of 0.9411

The low accuracy network is:

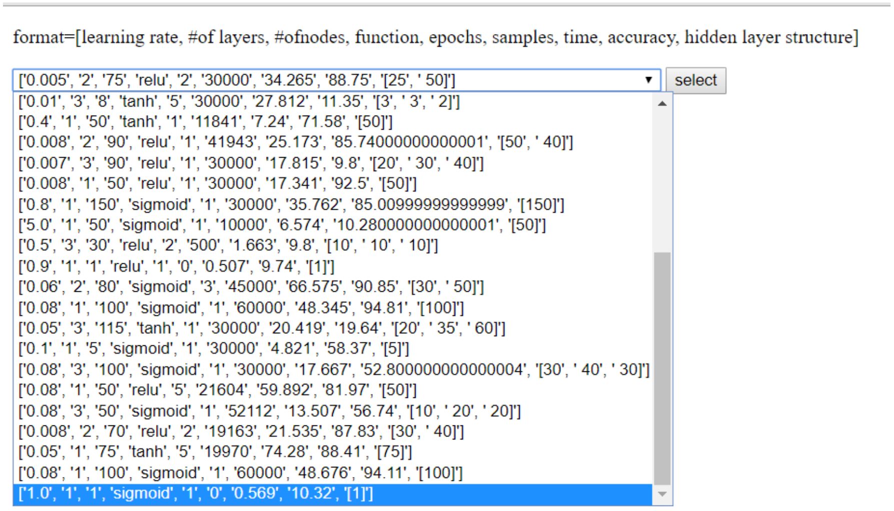


results for 1 layer network with 1 total hidden nodes, using a learning rate of 1.0 with the sigmoid activation function: epochs: 1, used 0 image samples

The accuracy of the network is 0.1032

Has an accuracy of 0.1 (which is 10% and is expected as there are an equal number of each number)

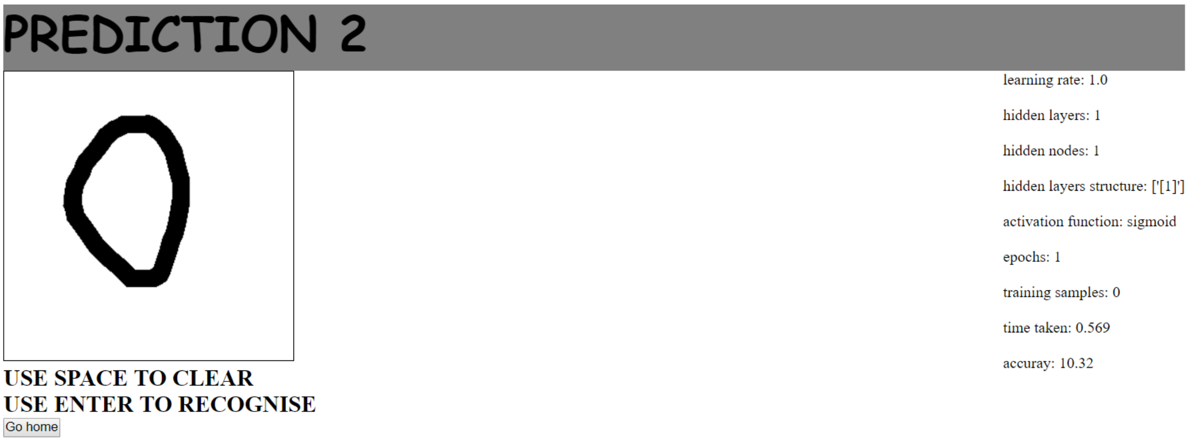
Next on the test page I select the network with the low accuracy.



format=[learning rate, #of layers, #ofnodes, function, epochs, samples, time, accuracy, hidden layer structure]

```
['0.005', '2', '75', 'relu', '2', '30000', '34.265', '88.75', '[25', ' 50]']
['0.01', '3', '8', 'tanh', '5', '30000', '27.812', '11.35', '[3', ' 3', ' 2]']
['0.4', '1', '50', 'tanh', '1', '11841', '7.24', '71.58', '[50]']
['0.008', '2', '90', 'relu', '1', '41943', '25.173', '85.74000000000001', '[50', ' 40]']
['0.007', '3', '90', 'relu', '1', '30000', '17.815', '9.8', '[20', ' 30', ' 40]']
['0.008', '1', '50', 'relu', '1', '30000', '17.341', '92.5', '[50]']
['0.8', '1', '150', 'sigmoid', '1', '30000', '35.762', '85.00999999999999', '[150]']
['5.0', '1', '50', 'sigmoid', '1', '10000', '6.574', '10.280000000000001', '[50]']
['0.5', '3', '30', 'relu', '2', '500', '1.663', '9.8', '[10', ' 10', ' 10]']
['0.9', '1', '1', 'relu', '1', '0', '0.507', '9.74', '[1]']
['0.06', '2', '80', 'sigmoid', '3', '45000', '66.575', '90.85', '[30', ' 50]']
['0.08', '1', '100', 'sigmoid', '1', '60000', '48.345', '94.81', '[100]']
['0.05', '3', '115', 'tanh', '1', '30000', '20.419', '19.64', '[20', ' 35', ' 60]']
['0.1', '1', '5', 'sigmoid', '1', '30000', '4.821', '58.37', '[5]']
['0.08', '3', '100', 'sigmoid', '1', '30000', '17.667', '52.800000000000004', '[30', ' 40', ' 30]']
['0.08', '1', '50', 'relu', '5', '21604', '59.892', '81.97', '[50]']
['0.08', '3', '50', 'sigmoid', '1', '52112', '13.507', '56.74', '[10', ' 20', ' 20]']
['0.008', '2', '70', 'relu', '2', '19163', '21.535', '87.83', '[30', ' 40]']
['0.05', '1', '75', 'tanh', '5', '19970', '74.28', '88.41', '[75]']
['0.08', '1', '100', 'sigmoid', '1', '60000', '48.676', '94.11', '[100]']
['1.0', '1', '1', 'sigmoid', '1', '0', '0.569', '10.32', '[1]']
```

Once on the draw page, I expect all inputs to produce the same prediction because this network has received no training.



PREDICTION 2

learning rate: 1.0

hidden layers: 1

hidden nodes: 1

hidden layers structure: ['[1]']

activation function: sigmoid

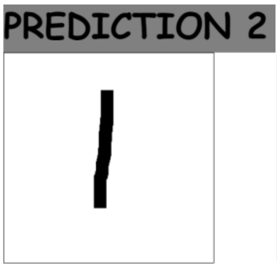
epochs: 1

training samples: 0

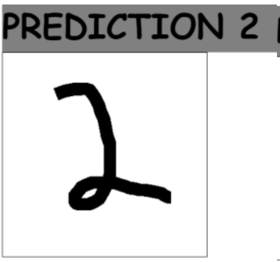
time taken: 0.569

accuray: 10.32

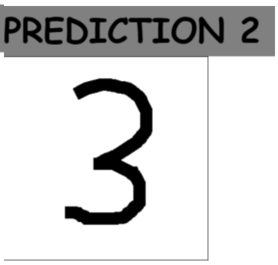
USE SPACE TO CLEAR
USE ENTER TO RECOGNISE



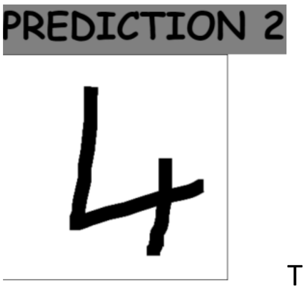
PREDICTION 2



PREDICTION 2



PREDICTION 2

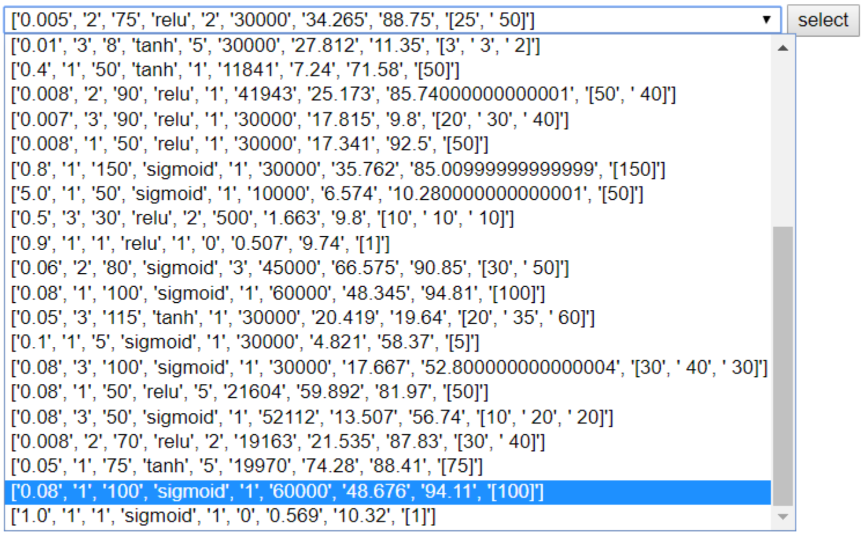


PREDICTION 2

The output of the network is always 2 because it has not been trained.

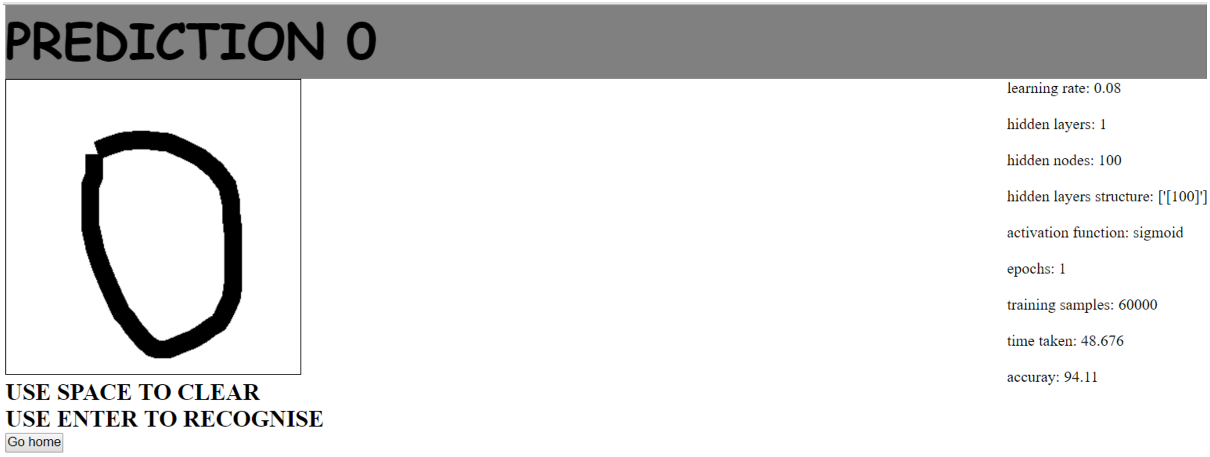
This means that these images are being run through the correct network.

Next, I select the network with the high accuracy.



```
['0.005', '2', '75', 'relu', '2', '30000', '34.265', '88.75', '[25', ' 50]']
['0.01', '3', '8', 'tanh', '5', '30000', '27.812', '11.35', '[3', ' 3', ' 2]']
['0.4', '1', '50', 'tanh', '1', '11841', '7.24', '71.58', '[50]']
['0.008', '2', '90', 'relu', '1', '41943', '25.173', '85.74000000000001', '[50', ' 40]']
['0.007', '3', '90', 'relu', '1', '30000', '17.815', '9.8', '[20', ' 30', ' 40]']
['0.008', '1', '50', 'relu', '1', '30000', '17.341', '92.5', '[50]']
['0.8', '1', '150', 'sigmoid', '1', '30000', '35.762', '85.00999999999999', '[150]']
['5.0', '1', '50', 'sigmoid', '1', '10000', '6.574', '10.280000000000001', '[50]']
['0.5', '3', '30', 'relu', '2', '500', '1.663', '9.8', '[10', ' 10', ' 10]']
['0.9', '1', '1', 'relu', '1', '0', '0.507', '9.74', '[1]']
['0.06', '2', '80', 'sigmoid', '3', '45000', '66.575', '90.85', '[30', ' 50]']
['0.08', '1', '100', 'sigmoid', '1', '60000', '48.345', '94.81', '[100]']
['0.05', '3', '115', 'tanh', '1', '30000', '20.419', '19.64', '[20', ' 35', ' 60]']
['0.1', '1', '5', 'sigmoid', '1', '30000', '4.821', '58.37', '[5]']
['0.08', '3', '100', 'sigmoid', '1', '30000', '17.667', '52.800000000000004', '[30', ' 40', ' 30]']
['0.08', '1', '50', 'relu', '5', '21604', '59.892', '81.97', '[50]']
['0.08', '3', '50', 'sigmoid', '1', '52112', '13.507', '56.74', '[10', ' 20', ' 20]']
['0.008', '2', '70', 'relu', '2', '19163', '21.535', '87.83', '[30', ' 40]']
['0.05', '1', '75', 'tanh', '5', '19970', '74.28', '88.41', '[75]']
['0.08', '1', '100', 'sigmoid', '1', '60000', '48.676', '94.11', '[100]']
['1.0', '1', '1', 'sigmoid', '1', '0', '0.569', '10.32', '[1]']
```

As the accuracy of this network is 94%, I expect it to predict almost every single digit drawn correct.



PREDICTION 0

learning rate: 0.08

hidden layers: 1

hidden nodes: 100

hidden layers structure: ['[100]']

activation function: sigmoid

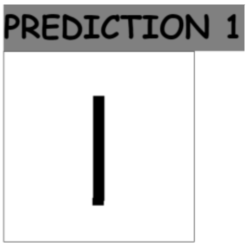
epochs: 1

training samples: 60000

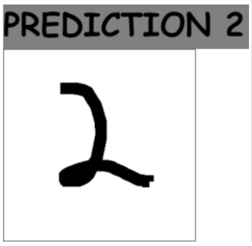
time taken: 48.676

accuray: 94.11

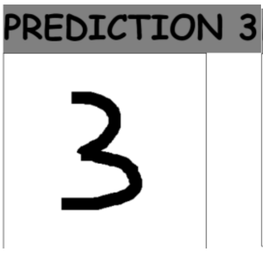
USE SPACE TO CLEAR
USE ENTER TO RECOGNISE



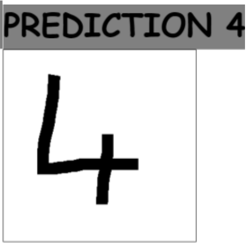
PREDICTION 1



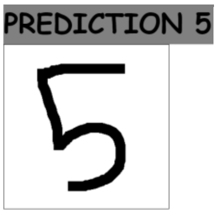
PREDICTION 2

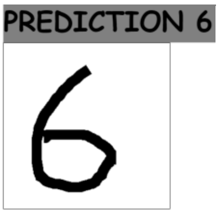


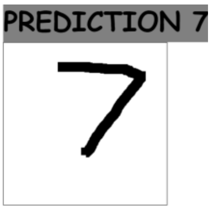
PREDICTION 3

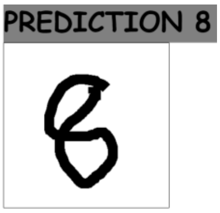


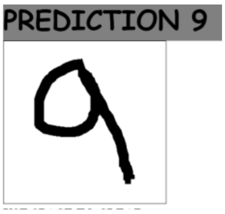
PREDICTION 4


PREDICTION 5


PREDICTION 6

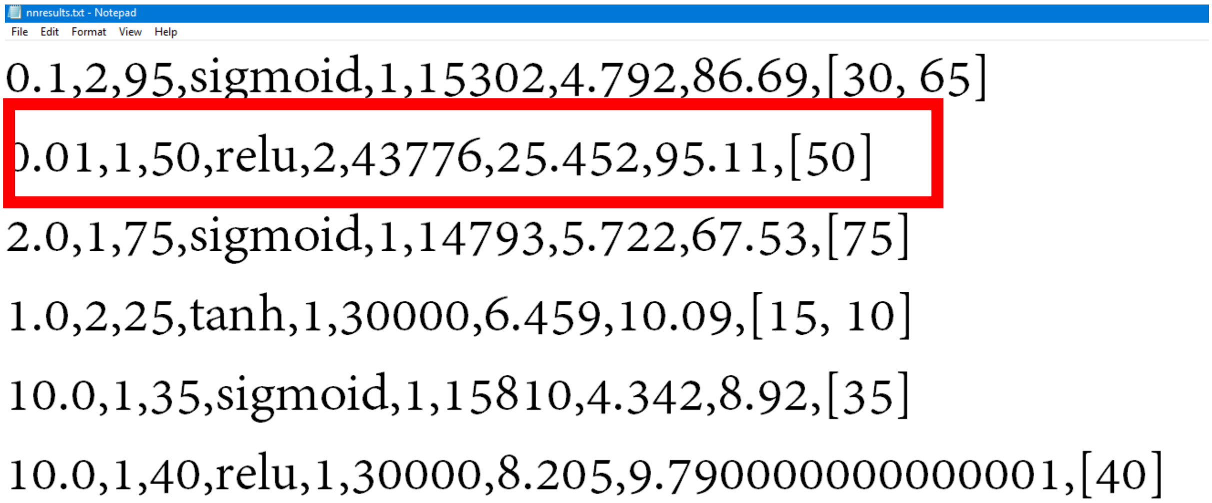

PREDICTION 7


PREDICTION 8


PREDICTION 9

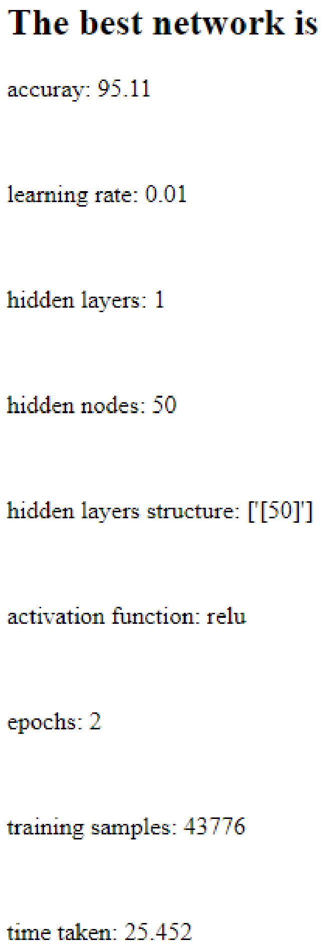
The network was able to predict all of my images, showing that the formatting algorithm is working correctly.

9. The next test was to see whether the best network element of the program was functioning correctly, this was a simple test where I looked at the networks text file and manually search for the network with the highest accuracy.



```
0.1,2,95,sigmoid,1,15302,4.792,86.69,[30, 65]
0.01,1,50,relu,2,43776,25.452,95.11,[50]
2.0,1,75,sigmoid,1,14793,5.722,67.53,[75]
1.0,2,25,tanh,1,30000,6.459,10.09,[15, 10]
10.0,1,35,sigmoid,1,15810,4.342,8.92,[35]
10.0,1,40,relu,1,30000,8.205,9.790000000000001,[40]
```

From this image you can see that the highlighted network with an accuracy of 95% is the greatest in the text file.



The best network is

accuray: 95.11

learning rate: 0.01

hidden layers: 1

hidden nodes: 50

hidden layers structure: ['[50]']

activation function: relu

epochs: 2

training samples: 43776

time taken: 25.452

This is the output once the best network button is clicked, all the displayed information matches up with the text file indicating it is working correctly.

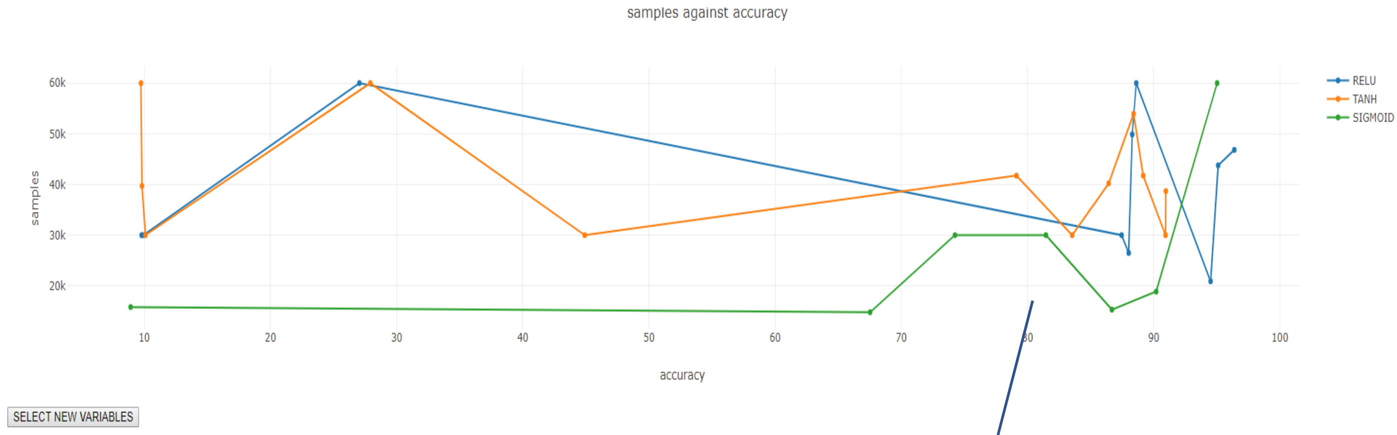
10. The last piece of testing I did was the validation checking on the different webpages. I have included these tests in video. The video also demonstrated how each element of the program functions. Link : https://youtu.be/5aFUpevWEYU

Conclusion

The main objective of this program was to discover what attribute values should be used to to optimise the accuracy of a neural network In relation to image recognition. Also, whether a neural network would be suitable to be used in the implementation of a child writing assistant application. After creating this tool, I trained a large number of neural networks while varying the values of each of the attributes.

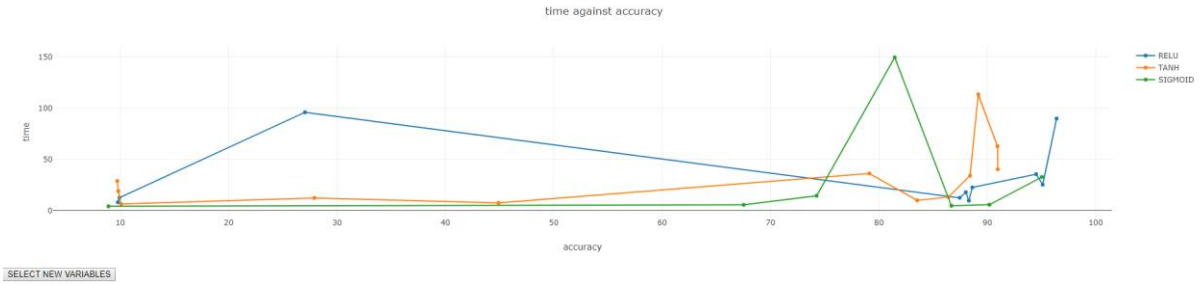
The collection of these networks allowed me to analyse how each of these attributes affected the accuracy of the netowrk and the time taken to train the network by viewing various graphs.

This graph shows training samples against accuracy:



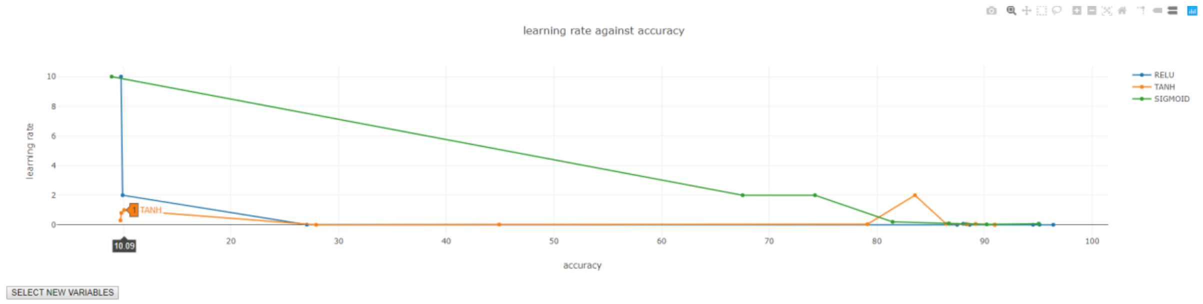
This graph shows that for the network to have a high accuracy it is not necessary for there to be a relatively large number of samples. For example the network with the highest accuracy does not use all available 60,000 samples.

This graph shows time taken to train against accuracy:

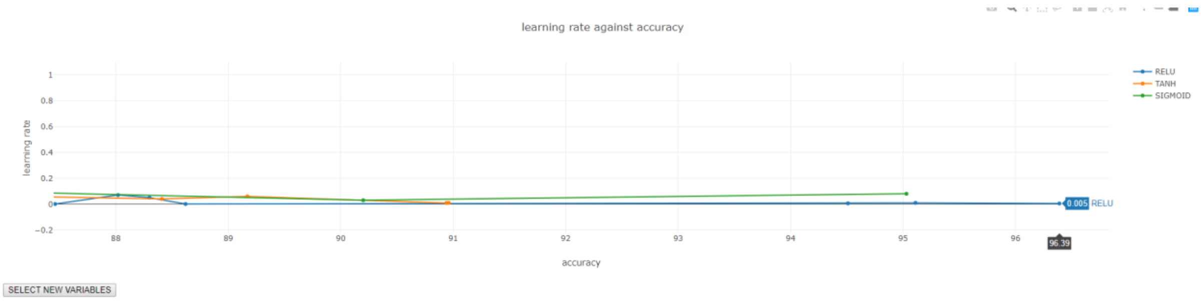


Excluding the anomalies at the beginning of this graph, it shows that it is possible for a netowrk to have a hight accuracy while being trained in a short amount of time. From this graph we can also see that at the higher accuracy networks, the relu and sigmoid functions seem to complete training faster than the tanh function.

This graph shows learning rate against accuracy:

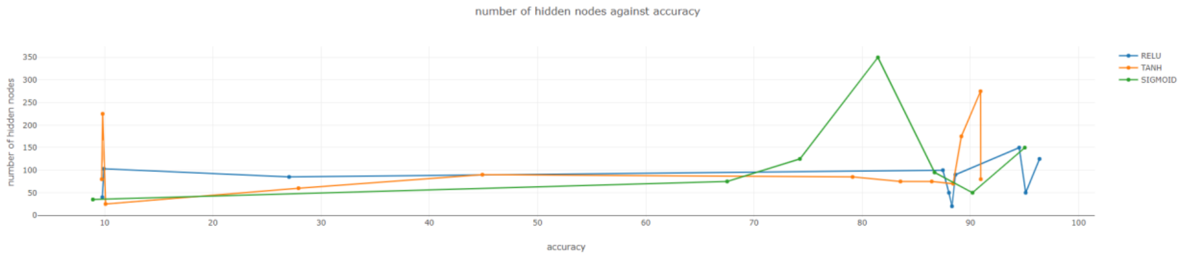


This shows that learning rate and accuracy seem to have a negative correlation, meaning that a high learning rate results in a worse accuracy. In addition to this the sigmoid function performs better when trained with a high accuracy. If we zoom into the high accuracy end of the graph:

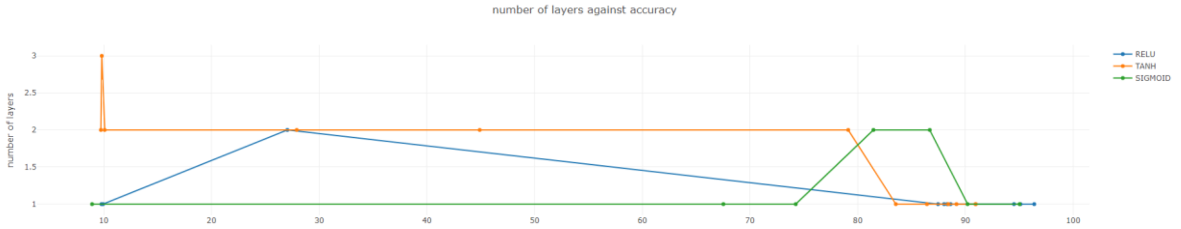


We can see that the optimal learning rate for a network is between 0.001 and 0.2. In addition, the relu function seems to be most accurate when its learning rate is below 0.01.

The next graph I looked at was accuracy against total hidden nodes and layers:

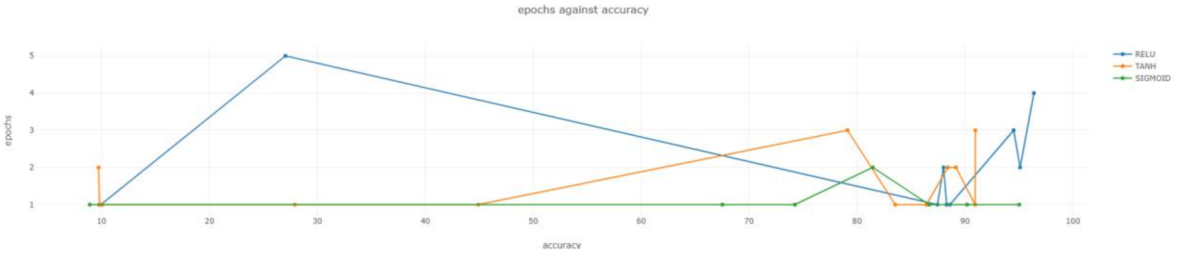


This graph shows that all the activation functions perform approximatley equally in regards to number of nodes, and that a larger number of hidden nodes does not always result in a higher accuracy, the optimal range is between 50 and 150 hidden nodes.



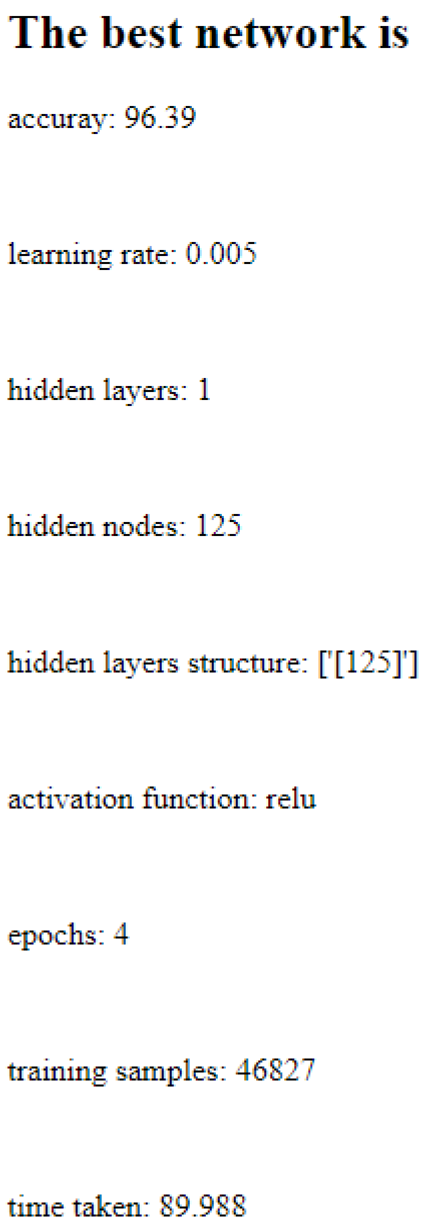
The accuracy against hidden layer graph clearly shows that having more than 1 hidden layer for this problem reduced the accuracy of the network, the most accurate networks were when only a single hidden layer was implemented. All three functions performing better at a single hidden layer.

Lastly, a graph of accuracy against number of epochs:



Once again this graph shows that too many epochs will stop the network from learning correctly, the sigmoid and tanh functions perform best at either 1 or 2 epochs whereas the relu function performed best between 2 and 4 epochs.

The network with the highest accuracy was:



The best network is

accuray: 96.39

learning rate: 0.005

hidden layers: 1

hidden nodes: 125

hidden layers structure: ['[125]']

activation function: relu

epochs: 4

training samples: 46827

time taken: 89.988

From the graphs I have found that the optimum network for this problem is:

- The use of the relu function
- 0.001 < learning rate < 0.1
- 1 hidden layer with 50-150 nodes
- 2-4 epochs
- Approx 45,000 training samples.

To conclude, neural networks have the potential to be used in a writing assistant application for children. They are suitable because as shown, it possible to achieve a high amount of accuracy in recognising handwritten digits, this is likely to yeild similar results for handwritten letters.

Evaluation

objectives

Before I began my program I had set of objectives that my program should include I have met all my objectives:

- Allow a user to input:
  - Number of hidden layers
  - Number of hidden nodes per layer
  - Activation function to use
  - Number of epochs
  - Number of training samples (between 0 – 60,000)

  This is done on the train page, the user is allowed to individually assign each of these variables.

- Create and a train a neural network using the user provided attributes
  To meet this objective, I have created a neural network class that is able to take all the users variables and create a neural network from them. I have created a web server that transfers the user selected variables into python. I have also obtained the mnist handwritten digit dataset and have formatted It correctly to be used for training. The neural network class has a train method that will train the network object with any given data.
- Output the accuracy and time taken to train network
  Using the python server, I have trained the neural network with the selected variables and have also implemented an evaluate method which determines the accuracy of the current network model using the 10,000 mnist test data samples. I have also used the time module to record the time taken for this process. These values are outputted onto the trained page.
- Store the neural networks attributes and the resulting accuracy and time to train
  This is done by using a standard format and writing a list containing all the relevant variables to a text file each time a network is trained, creating a text file containing all information about all the networks.
- Allow the user to select two different attributes
  On the results page I have provided the list of variables with x axis and y axis subheadings to allow the user to select a variable for each of the axis.
- Use stored information to produce a scatter graph of the selected attributes
  I have done this by reading the previously mentioned text file and from this selecting the user chosen variables and storing only these values in a list. Next the plotly module is used to plot a graph of these points with a different trace for each activation function. A div tag which contains all the html data is outputted to the webpage and to the user
- Allow users to select one of the stored networks
  When a network is trained the network object is stored in a pickle file, this means that the weights and all other attributes are also stored.
-

-
-
- Allow user to input their own img to test selected network
  I have done this on the draw page by using a canvas to allow the user to hand draw any image which is then ran through their chosen network and the prediction is then shown to the user.

- Ultimately the user should be able to come to a conclusion about what the optimum variables are to create a neural network which maximises accuracy. Whether a neural network is suitable in assisting children in learning to write.
  I have used the graphing tool that I created to arrive at a conclusion as to what the optimal attributes are for maximising accuracy for a neural network for this problem and whether neural networks are suitable for this task.

- The program should be able to be used to optimise a neural network for any problem that involves the use of supervised learning.

  I have created a python-based web server that is capable of being moved onto a paid server to allow public access to the program, anyone could experiment with networks and contribute to the stored network data.

- Using the results produced by this neural network tool I will create an image recognition tool that utilises the optimal neural network attributes, to maximise the accuracy of the network.
  I have created a webpage that is able to train a neural network with given attributes and using the data from this I have been able to produce graphs of the different attributes; I have also displayed the current best network and the attributes associated with it.

Feedback

I showed my project (the testing page) to a family member who Is a primary school teacher, they said the following: “the program is able to accurately predict my handwritten digits even when drawn in a variety of different ways, I can see this being useful in teaching children how to write”, “this would be better If it allowed it predicted handwritten letters, and letters written side by side”.

I told them that this would be possible, to achieve this I would have to acquire a handwritten letters dataset and then create a user interface appropriate for children to write words. This can be done by splitting the user input into separate boxes where a child can write each letter.

Improvements

Although I have met all my objectives, there are some extra features that I feel could have been implemented to make my program a better investigation tool. One of these features is to provide an option for a user to introduce their own dataset rather than having just handwritten digits, the user could customise this tool to their investigation/problem.

Another feature that would have made the program more useful would've been to implement an algorithm that automates the process of the user entering in attributes, this algorithm would randomly test attributes and begin to reduce the variation in the attributes as a greater accuracy is discovered. This can be done by randomly training a network with random attributes. Once a network with an accuracy greater than 50% (this can be any percentage) is found the algorithm will make slight adjustments to the new networks it trains until the accuracy increases.

The results produced by this application and using the neural network class implemented in this program can be implemented into an interface that is appropriate for a child. This program would have features that would allow a chid to write a word and then recognise the handwritten word; provide feedback to the child. This could be implemented for a mobile device making it easily accessible and easy to use as the input would be the touch of the child.

Refrences:

https://www.python-course.eu/machine_learning.php

https://medium.com/@o.kroeger/tensorflow-mnist-and-your-own-handwritten-digits-4d1cd32bbab4

https://www.kaggle.com/oddrationale/mnist-in-csv

http://yann.lecun.com/exdb/mnist/index.html