Language Identification Neural Network

A-LEVEL COMPUTER SCIENCE

Hannah Stone 2946
BARTON PEVERIL COLLEGE | 58231

1 CONTENTS

2 Analysis

2.1 Project Background

Artificial intelligence and machine learning is becoming a large part of not only computing, but also society. However, to the regular individual, it is normally overlooked and the complex fundamentals behind it are not noticed. I am interested in looking into how machine learning can make our lives easier and more efficient after reading posts and watching videos about the matter. I feel that this area of computing will become increasingly important and common place in daily life. This project will not only focus on my chosen area, language identification, but will also act as an investigation into neural networking in general.

2.2 Project Outline

The aim for this project is to create a program consisting of a neural network which will learn to understand the difference between various languages and for it to then predict what language a word is in when given a new input. I believe that this is a realistic type of machine learning program that would be used commonly. For instance, identifying what language a webpage is in to see if it needs to be translated for the user. Overall, it is useful for any situation where there is a possibility of needing to know what language a piece of text is in. The program will not store any data that is inputted into it, which means it won't just look up the word within a stored file for example. It will predict the language based off of what it has learned from the training data. This would reduce the program's size and means that it can predict the language of words that may be more colloquial and not recognised in a regular dictionary. This project will be an investigative one, which will try and attempt to answer the question to if neural networks are an effective solution to language identification.

2.3 Research

2.3.1 Natural Language Processing

NLP focuses on the interactions between computers and human languages. It is an area of computer science that is particularly interested in how we can program computers to process large amounts of natural language data. Applications of NLP include many fields of study such as, machine translation, user interfaces, speech recognition and text prediction. The overall purpose is to achieve human-like language processing for a range of tasks or applications. There are multiple types of language processing when humans produce or comprehend language and it is thought that humans normally use all of these levels in combination. However, many NLP systems use specific levels, or combinations of different levels and this is seen by the differences between various NLP applications.

Natural language processing through the use of computers is considered to be an area within Artificial Intelligence as NLP's goal is to have human-like performance. The choice of the word 'processing' instead of 'understanding' as it used to be referred to in the early days of AI is important because in order for the program to be understanding of text for example, it would need to be able to draw inferences from the text which has not been achieved.

The different approaches to natural language processing is able to be placed roughly into four categories: symbolic, statistical, connectionist, and hybrid. The program that I have chosen to create falls mostly into the connectionist category. This approach develops a generalised model from examples of the language. In my case, it will be thousands of individual words. A connectionist system is harder to observe than other systems such as a statistical one because they are less constrained. Overall, a connectionist model is a network that consists of interconnected simple processing units that has knowledge stored in the weights of the connections between these units. The units in my program will be the nodes where the inputs will be stored. The connections are the synapses that have individual weights to them which are used in calculating and adjusting the hidden layer nodes. Interactions among units can result in a dynamic global behaviour which is what leads to computation and improvement. My connectionist model program is more specifically a distributed model which means that a concept in the model is represented by the simultaneous activation multiple units and the activation of an individual unit is only a participant of the overall concept representation.

2.3.2 Backpropagation Artificial Neural Network

The term backpropagation refers to a broad family of Artificial Neural Networks. The general structure consists of different interconnected layers that are either hidden or visible. Backpropagation artificial neural networks (BPANN) learn by using the gradient descent algorithm in order to minimise the error. A BPNN needs to be provided an appropriate number of hidden units (nodes) to map any function to perform gradient descent on. Usually, the ANN needs at least two layers of hidden units if the function is quite complex or the data isn't that reliable so needs to pass through multiple levels to be filtered. BPNNs tend to distribute themselves across its connections which makes it's learning process more compact and efficient. However, it makes it difficult to show and make clear the implicit knowledge that the neural network has and acquires during its training process.

The backpropagation algorithm can be split into two phases: propagation and weight update. For the propagation phase, it will propagate forward through the network to produce output values by passing through the hidden layers. Then a calculation of the error needs to be made. After that, a propagation is made back through the network using the target value in mind to generate delta values which are the difference between the target and actual values. It will do this to all of the output and hidden neurons. In the weight update phase, every weight has its output delta value and input activation multiplied in order to find the gradient of the weight. Then, a percentage of the weight's gradient is subtracted from itself. The percentage that the weights are adjusted by is chosen and set in the program. The percentage amount influences how accurate and how quickly the network trains. A higher percentage trains quicker as this will make larger adjustments every pass of the algorithm, but a lower percentage produces more accurate results as it can find the optimisation to a more precise amount.

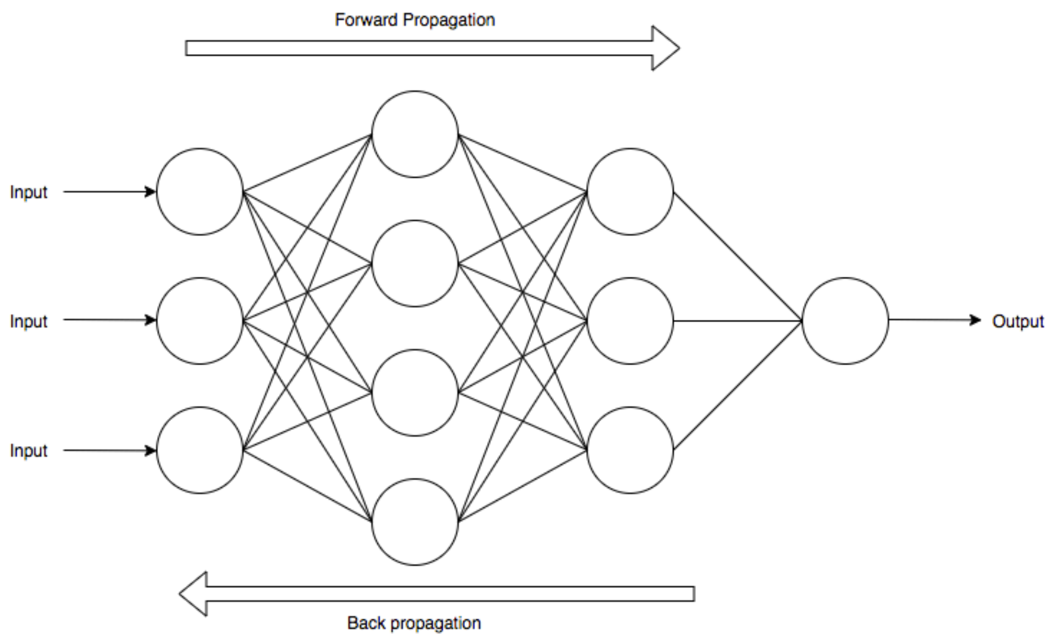


Figure 1: A simple diagram of a backpropagation artificial neural network

2.3.3 Existing Language Identification Applications

Google Detect Language

The most well-known language identification application that is widely used is the one created by Google. It is implemented within Google Translate and built-in to Google Chrome. When used in Google Translate, the user is able to type words into its text input and will attempt to identify what the language is. The more words that are entered of that language, the more accurate the identification will be. In Google Chrome, when the user loads a webpage that is not in their native language (which is set in the settings of Google Chrome when the user downloads the application), Chrome is able to identify what language the webpage is in and will ask if you would like it translated or not. Automatic translation can also be set so that all webpages will appear in the user's desired language when possible.

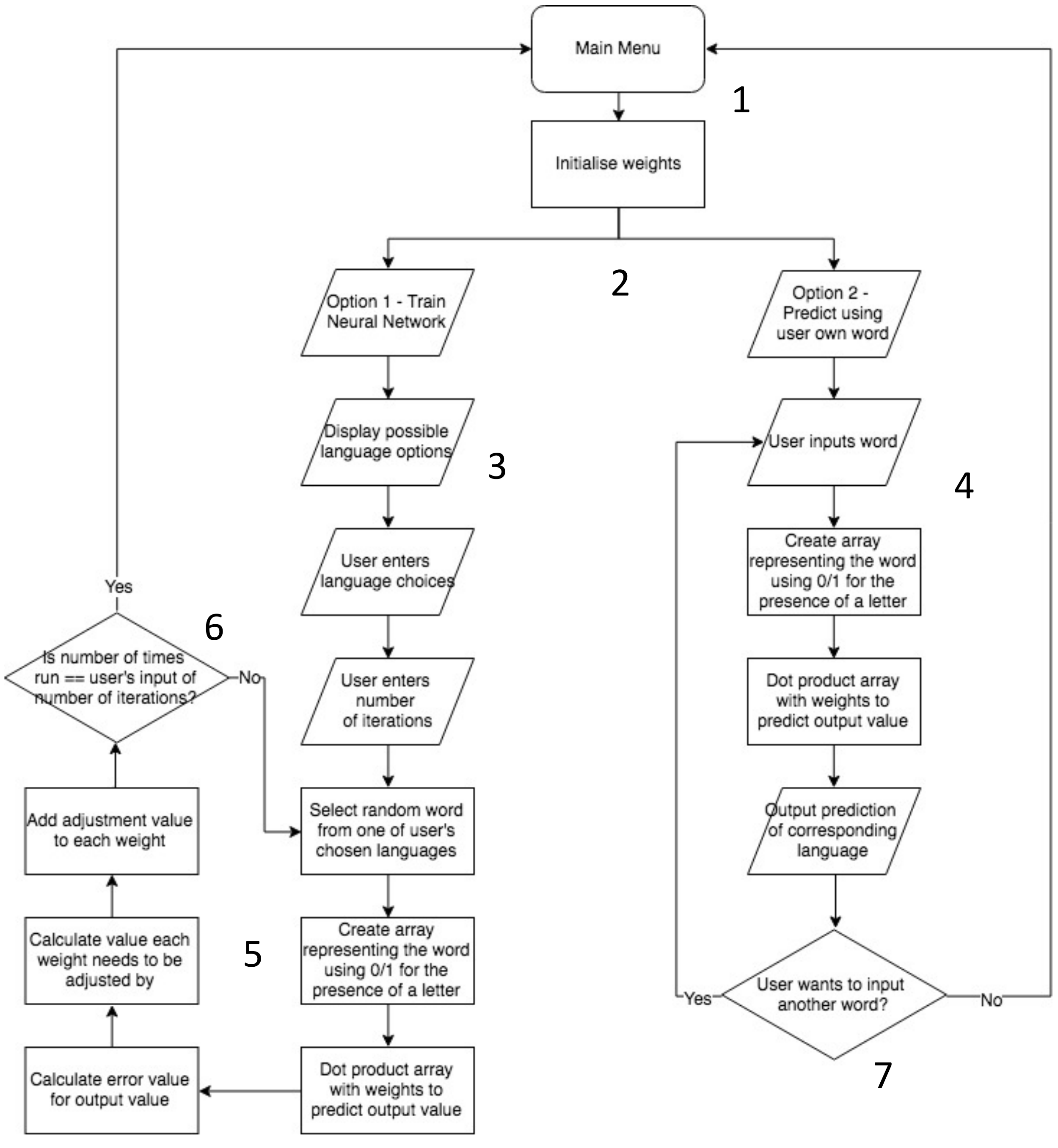
The way Google does language identification is through the use of statistical language identification. This is a different method to the one I am choosing to investigate. Google's method of statistical language identification scans a chunk of the given text and segments them into 4-character 'tokens'. These tokens are compared against a large reference of tokens and within the reference, language properties are associated with them. In order to build the reference tables for different languages, millions of words of source text had to be analysed.

2.4 Objectives

1. The user will be able to input a word into the program and it will then display what language it thinks it is
    a. The program should be able to switch between allowing the user to input a word to allowing the program to train on the data set
    b. The program must check that the word is not over a certain length
    c. The program must ensure that the user input only contains characters that are able to be processed by the program
2. The program will train to become more accurate by using a set of training data
    a. The number of languages will be set but will have the ability to be expanded by providing new training data
    b. The prediction given by the computer must be from its calculations performed in the feed forward back propagation algorithm and not from any referencing to stored data
    c. As the program runs, the program should tell the user what generation it is on in order to monitor the progress of the training
    d. The program will need to randomly select words from the training data set which will be stored in a separate text file
    e. Every letter in the chosen word needs to be converted into a 26 x 1 matrix with corresponding 1's and 0's for what that letter is eg. For letter C = [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
3. The program will need to perform the necessary steps of a backpropagation neural network
    a. Random starting weights need to be initialised between -1 and 1 for every neuron
    b. An activation function needs to be chosen and used on all of the generated weights
    c. The inputs and the weights need to be multiplied together as matrices by performing a dot product
    d. The predicted output and actual output needs to be calculated by finding the difference between them
    e. The derivative of the sigmoid function needs to be included in the program
    f. Gradient descent needs to be performed on every neuron to find each adjustment value
    g. The adjustment value needs to be applied to every weight
    h. The whole process needs to be repeated many times in order to improve the weight values so the predictions become more accurate
4. The program will need to import the numpy library for the use of higher level mathematical functions
    a. The exp, array and dot functions will need to be imported from numpy

3 DESIGN

3.1 HIGH-LEVEL OVERVIEW


Main Menu
1
Initialise weights
2
Option 1 - Train Neural Network
Option 2 - Predict using user own word
Display possible language options
3
User enters language choices
User enters number of iterations
Select random word from one of user's chosen languages
Create array representing the word using 0/1 for the presence of a letter
Dot product array with weights to predict output value
5
Calculate error value for output value
Calculate value each weight needs to be adjusted by
Add adjustment value to each weight
Is number of times run == user's input of number of iterations?
Yes
No
6
User inputs word
4
Create array representing the word using 0/1 for the presence of a letter
Dot product array with weights to predict output value
Output prediction of corresponding language
User wants to input another word?
Yes
No
7

1 – The program starts with displaying the main menu and initialising the weights that will be used to optimise the neural network. The weights are numbers that represent connections between nodes. In the program, the weights will be randomly generated and will only be between -1 and 1. They are placed in a 390 * 1 array to represent the 15 possible letters a word can be made of and the 26 different letters they can be. 15 * 26 = 390. This is how many connections there are. Each weight value will later be adjusted by the backpropagation algorithm.

2 – Once the weights have been initialised, the user is able to input their option choice. This will either be to train the neural network or to use it to predict the language of their own word. For best results, the user should choose to train the network first before selecting the second menu option.

3 – Option 1 allows the user to train the language to detect between two chosen languages. The languages the user can choose is limited to a set number as there is only a set amount of training data available in a set number of languages. The possible languages will be clearly displayed to the user and they will be able to select the two languages of choice.

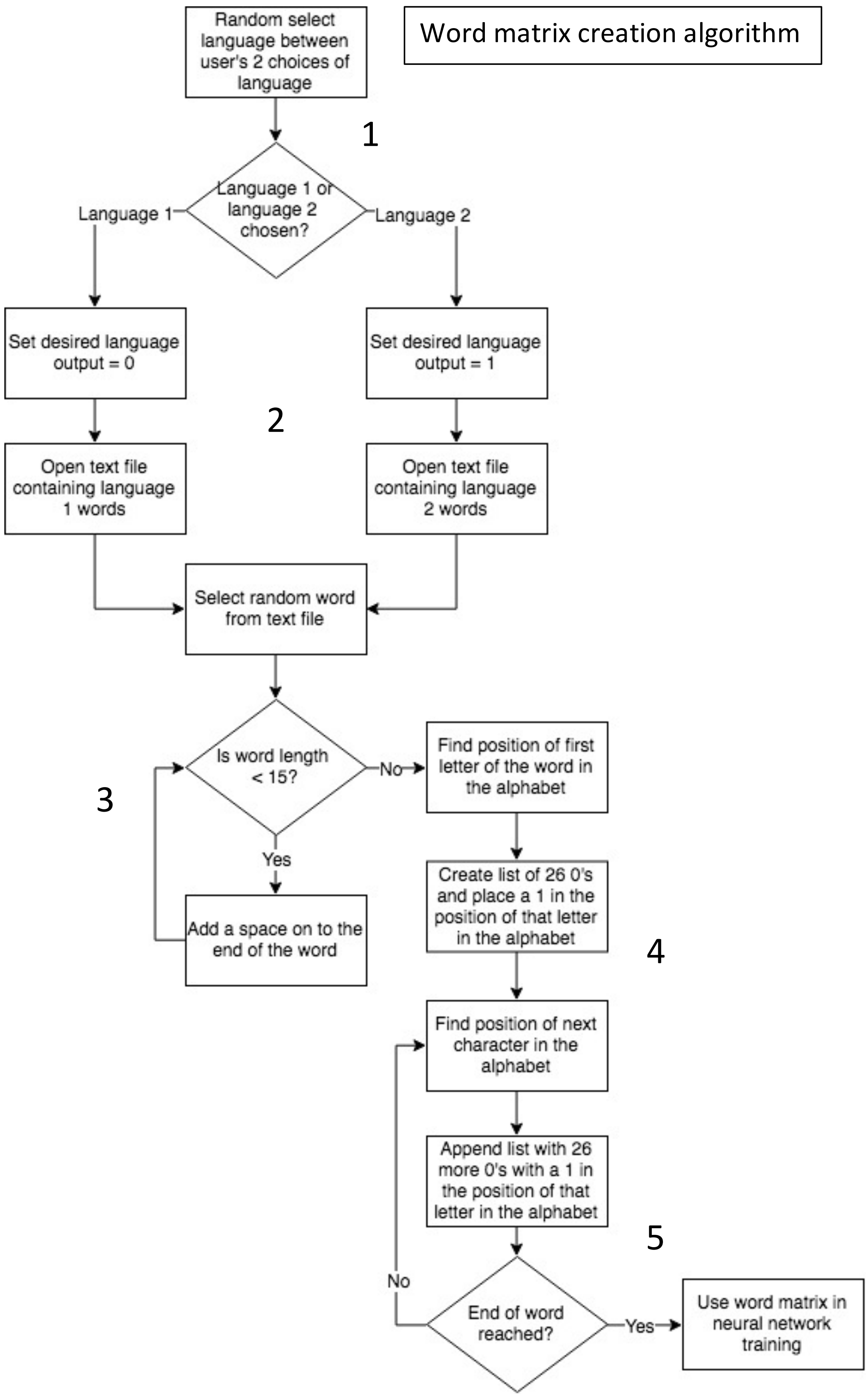
4 – Option 2 allows the user to input their own word to be used in the neural network. The accuracy of the prediction will be dependent on how many iterations the neural network has trained as the weights need to be optimised for the two languages. The user will have the ability to enter their own word and the prediction will be printed out to the user.

5 – After user language selection at the start of Option 1, the backpropagation algorithm is used to perform several calculations and functions to the weights and input word. A word will be randomly selected from training data from either one of the languages for it to be then converted a matrix representing each individual letter in the word. The neural network will attempt to predict the language of the word by performing a dot product of the weights and word matrix to find an output value. An error value is then found to determine how the weights need to be adjusted (ie. whether the neural network got it right) and the weights then have the adjustment value applied to each one.

6 – The program needs to run the backpropagation algorithm for the number of iterations the user had previously inputted at the start of Option 1. If the number of iterations that have been performed is less than the number of iterations the user inputted, then the algorithm is performed again meaning that the process is a loop that will only break until the number of desired iterations has been performed.

7 – The user is able to try their own word as many times as they want as the program will ask the user if they want to enter another word. If they do, Option 2 is repeated to allow a new user input. The process of the user entering their own word and then performing the prediction calculation does not affect the weights in the neural network as there is no error value calculation and adjustment value. This means the neural network can be indefinitely tested to find the accuracy of its language identification ability. If the user no longer wants to enter their own word and would like to return back to the main menu, they are able to do so when the program asks if they want to enter another word.

3.2 ALGORITHMS


Word matrix creation algorithm
Random select language between user's 2 choices of language
1
Language 1 or language 2 chosen?
Language 1
Language 2
Set desired language output = 0
Set desired language output = 1
2
Open text file containing language 1 words
Open text file containing language 2 words
Select random word from text file
3
Is word length < 15?
No
Yes
Add a space on to the end of the word
Find position of first letter of the word in the alphabet
Create list of 26 0's and place a 1 in the position of that letter in the alphabet
4
Find position of next character in the alphabet
Append list with 26 more 0's with a 1 in the position of that letter in the alphabet
5
No
End of word reached?
Yes
Use word matrix in neural network training

3.2.1 Word matrix creation algorithm

1 – The algorithm begins with a random selection between two of the user's previously selected languages. The program will check whether language 1 or 2 has been chosen and a certain training data file will be used depending on which language has been chosen.

2 – The desired output for the neural network to reach is set to either 0 or 1 depending on whether language 1 or 2 is chosen. This is done each time a new word needs to be used to train on. Once the desired output has been set, the text file containing training words from the selected language is opened and the program will select a random word from the text file. The text file consists of one word per line so the program will read the whole file and select a random line which will have the word in.

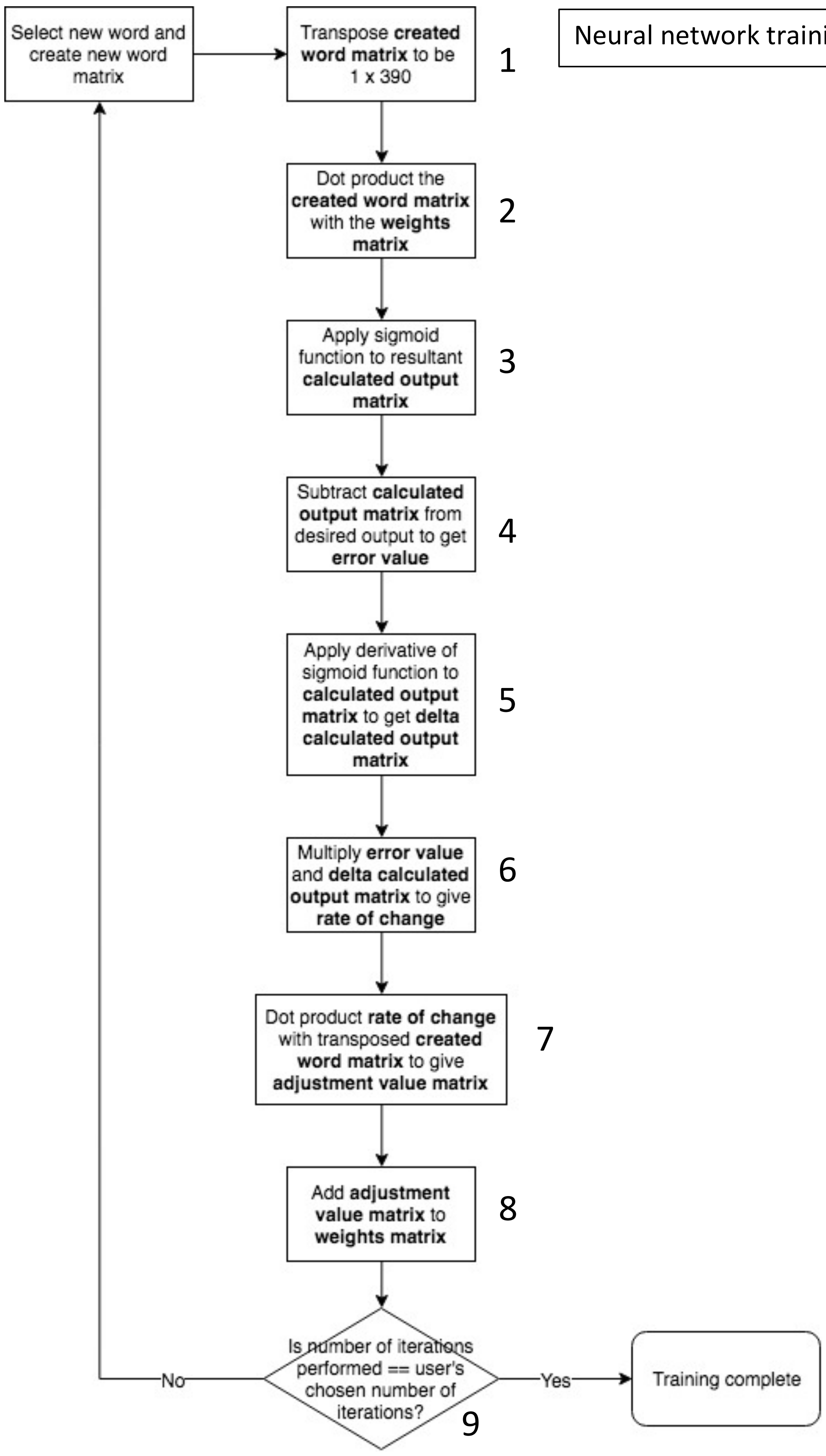
3 – The program checks the length of the word and if it is less than 15, spaces are added to the end so that it reaches 15 characters. This needs to be done because the size of the matrix that is going to be made from the word relies on the word being 15 characters long. The addition of spaces on the end doesn't affect the word as the space characters will just be represented as 0's in the matrix. If the word chosen is over 15 characters, the program will cut the extra characters off to make it 15. This is one of the limitations of the program but as it is unlikely for a word to be more than 15 characters long, it does not make an overall large effect on the working and accuracy of the neural network. The more iterations that are performed, the less of an impact this 15-character limitation will have.

4 – This part of the algorithm is where the word is converted into a numerical format. In the program, there is a reference to a list of all the letters in the alphabet. The program uses this list to reference the letters in the training word. It will begin with the first letter of the word and find what place in the alphabet it is. For example, if the first letter is 'e', it will know that the letter is 5th in the alphabet. The program will now create a list of 26 0's to represent each letter in the alphabet but then replace the letter's position with a 1 in this 26-element list. For example, for the letter 'e', the list will first have 26 0's but the 5th 0 will be replaced with a 1 to represent the presence of the letter 'e'. Therefore, there is now 25 0's and one 1 at the 5th position. After this has been done for the first letter of the word, the next letter has its position found. The program will take the previously made list and will extend it with 26 more 0's and replace one of the 0's with a 1 in the letter's corresponding position.

5 – The program will continue to extend the list for each letter in the training word. Once the end of the word has been reached, this word matrix will be used for the next algorithm in the program. This process of random word selection and then conversion into a binary style list is done for every iteration of the neural network. This allows each word to be treated independently. By using a 26-long list to represent each letter with a 1 in the letter's alphabet position, the neural network is able to distinguish each letter as a separate entity that doesn't actually have a numerical value. It may seem easier to create a list with the number 1 representing 'a', 2 representing 'b', 3 representing 'c' and so on, but this doesn't allow each letter to be taken as individual distinct inputs so patterns can't be found by the neural network. I needed to represent each letter as a 26-dimensional item so that it was representative of the concept of the alphabet. In theory, each list made for a letter, such as the letter 'p' for example, represents the absence of 'a', 'b', 'c' and so on and only the presence of 'p'. Additionally, the use of only 1's and 0's has meant that it is easier to program the neural network to optimise to either 0 or 1 depending on what the desired language output is.

Neural network training algorithm

* phrases in bold represent variables for easier reading


Select new word and create new word matrix
Transpose **created word matrix** to be 1 x 390
1
Dot product the **created word matrix** with the **weights matrix**
2
Apply sigmoid function to resultant **calculated output matrix**
3
Subtract **calculated output matrix** from desired output to get **error value**
4
Apply derivative of sigmoid function to **calculated output matrix** to get **delta calculated output matrix**
5
Multiply **error value** and **delta calculated output matrix** to give **rate of change**
6
Dot product **rate of change** with transposed **created word matrix** to give **adjustment value matrix**
7
Add **adjustment value matrix** to **weights matrix**
8
Is number of iterations performed == user's chosen number of iterations?
9
No
Yes
Training complete

3.2.2 Neural network training algorithm

1 – This algorithm starts with the word matrix that was created in the previous algorithm being transposed so that instead of being in a 390 x 1 shape, it is now 1 x 390. This is important for the functions and calculations that will be performed on the matrix later on in this algorithm.

2 – The created word matrix is dot producted with the matrix which contains all the weights. This multiplies each value in the word matrix which represents the absence or presence of a letter in the word. This produces a calculated output matrix which acts as the prediction for this word. If this is the first iteration of the neural network, the weights are completely random and so the output matrix will give a random prediction between the two languages.

3 – Once the calculated output matrix has been created, each value in the matrix has the sigmoid function applied to it. This is called the 'activation function' and will turn each calculated value into one that it between 0 and 1. This is useful in determining the final prediction value which will be a value either rounded to 1 or 0. In the word matrix creation algorithm, the training word is assigned a desired output value depending what language it is between the two the user selected earlier on in the program.

4 – Next in the algorithm, the calculated output matrix which has had the sigmoid function applied to it is now subtracted from the desired output value which was previously set earlier on in the program when the training word was selected. This produces a matrix of error values to find out how far away each value was from the desired output. The optimal error value to be found is 0, as this means that the calculated output value was exactly the same as the desired output. This is what the neural network is trying to optimise as it is trying to find weight values that will produce the desired output when the created word matrix is dot producted with it.

5 – The error value matrix has the derivative of the sigmoid function applied to it which is the first step in the backwards pass of the algorithm. The derivative of the sigmoid function is used because it is how gradient descent is performed (*more information in system rules*). This now gives a delta calculated output matrix which will be used in the calculation of adjustment values which ultimately is how the neural network becomes optimised for the two languages.

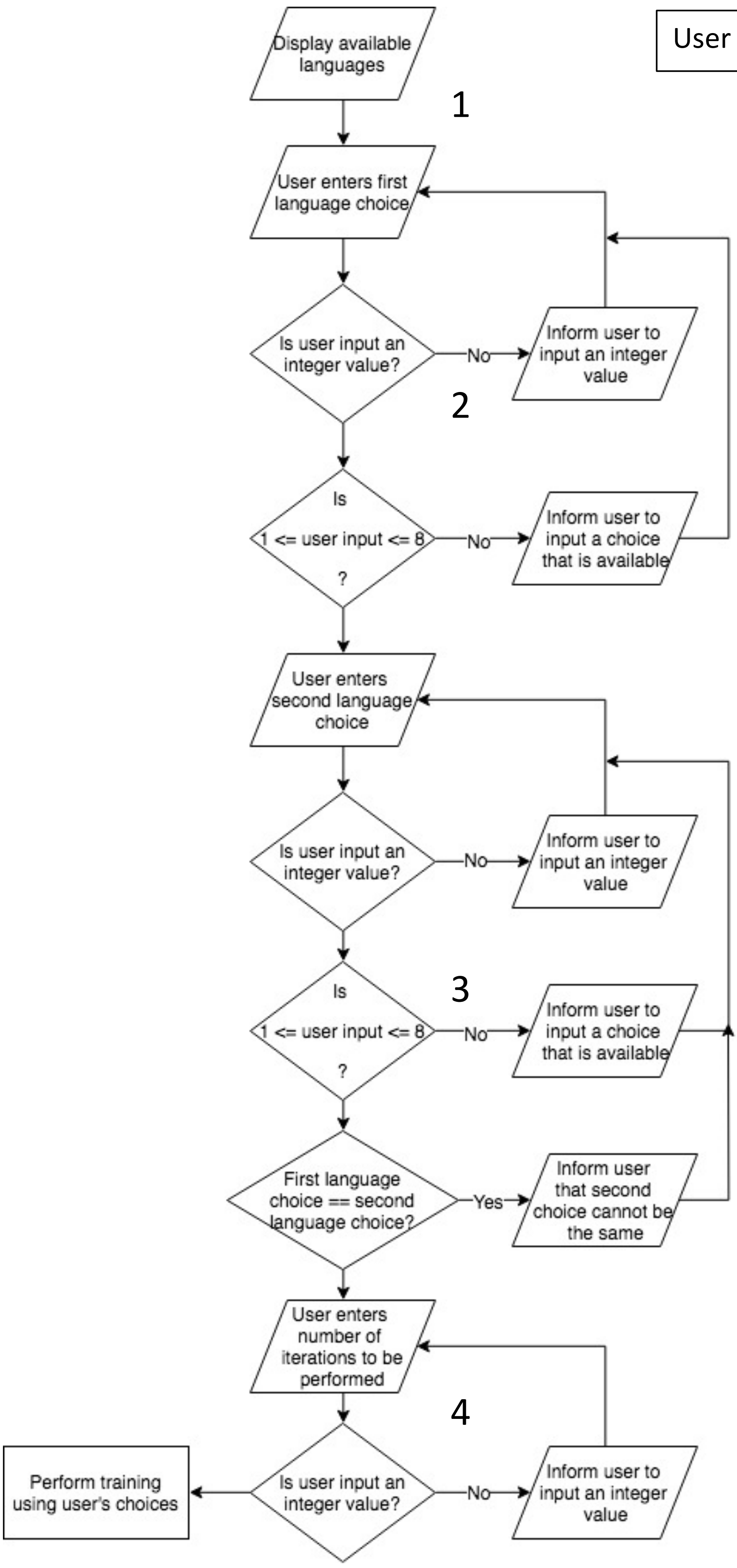
6 – The error value matrix is multiplied with the delta calculated output matrix in order to find the rate of change. The combination with the error value matrix gives the slope of the activation function at that error value. This allows the neural network to know how much the errors have contributed to the overall calculated output value so that it can be adjusted accordingly.

7 – To find the adjustment value matrix, the rate of change previously calculated is dot producted with the created word matrix. The values that are produced will be either positive or negative depending on how the weights need to be adjusted. This is the final step in the gradient descent algorithm.

8 – The adjustment value matrix is now applied to the weights matrix by adding the two matrices together. This means that the weights should now be better optimised for identifying the language of the word. Ideally, now when a word is inputted from one of the two available languages, the weights that multiply with the word matrix should give a value of 0 or 1 depending on what the language was.

9 – At the end of the algorithm, the program must check whether or not the algorithm has run for the number of times the user originally chose. If it has not run for the set number of times, the algorithm will loop again with the selection of a new word. The process of converting the word into a matrix and performing the necessary calculations to different matrices in order to optimise the weights will all be done again so that with each iteration, the weights will become better optimised and therefore the whole neural network will improve in accuracy.

User input error checking


Display available languages
1
User enters first language choice
Is user input an integer value?
No
Inform user to input an integer value
2
Is 1 <= user input <= 8 ?
No
Inform user to input a choice that is available
User enters second language choice
Is user input an integer value?
No
Inform user to input an integer value
3
Is 1 <= user input <= 8 ?
No
Inform user to input a choice that is available
First language choice == second language choice?
Yes
Inform user that second choice cannot be the same
User enters number of iterations to be performed
4
Is user input an integer value?
No
Inform user to input an integer value
Perform training using user's choices

3.2.3 User input error checking

1 – This algorithm is used in the first option in the main menu. This is where the user is able to select what two languages are to be used in the neural network training. Also, the user is able to enter how many iterations of the training are to be performed. There are error checking measures in place when the user enters their choices. The algorithm starts with the program displaying the available languages that can be chosen for the neural network to be trained on. The user then is able to enter their first choice of language. This will be a number corresponding to the language in the displayed list so that it is easier to error check.

2 – When the user enters their first choice of language, first the program checks if the input is an integer value because if it is not, then it is automatically invalid. If a non-integer value is inputted by the user, the program will inform the user that what they entered was an invalid input and allow for the user to re-enter an integer value. If an integer value has been inputted, the program then needs to check if the integer value is between 1 to 8. This is because there are 8 different language options to choose from which have been numbered 1 through to 8. This number would be extended if there are more language options added later on to the program. If the user does enter an integer value but it is a number that is not between 1-8, then the program will inform the user that their choice needs to be between 1-8 and allow for them to re-enter a valid language choice.

3 – After the user has entered their first option and it is valid, the user is then able to enter their second language choice. The program checks if the input is an integer value similarly to the first language input. If the input is not an integer, the program will again inform the user of their invalid input and then allow for the user to re-enter their second language choice. When an integer value has been inputted by the user, the program will check if the value is also between 1 to 8 and if it is not, the program will respond the same as how it does for the first language input. After this check, the program will then check if the second language choice is the same as the first language choice. If the second language choice is the same as the first, it will be invalid and the user will be asked to enter a different language choice. This is because the neural network needs to train on two different languages so it can identify between them.

4 – If both language inputs are valid, the user is then able to enter how many iterations they want the neural network to perform. The program only needs to check if the input is an integer value and if it is not, the program will allow the user to re-enter an integer value. If it is valid, the program can continue and begin it's training using the inputs the user has given.

3.3 SYSTEM RULES

3.3.1 Sigmoid Function

In my neural network, I have chosen to use the sigmoid function as my activation function. I have chosen the sigmoid function because it exists between 0 and 1. This is useful for my neural network because it will give me a value that is either closer to 1 or 0. The aim of my neural network is for it to predict what it thinks the language is of the word that is inputted. The way I have used its calculated prediction value, is to round the number to the closest integer (which is 0 or 1) and that will correspond to what language it thinks it is. The sigmoid function can also be used to give a probability between many different options. The calculated output using the sigmoid function can be thought of as a percentage so its final decision could be based on what is the highest probability between all of the outputs. This is utilised in neural networks that have an output layer with multiple nodes which then feed into one final decision node.

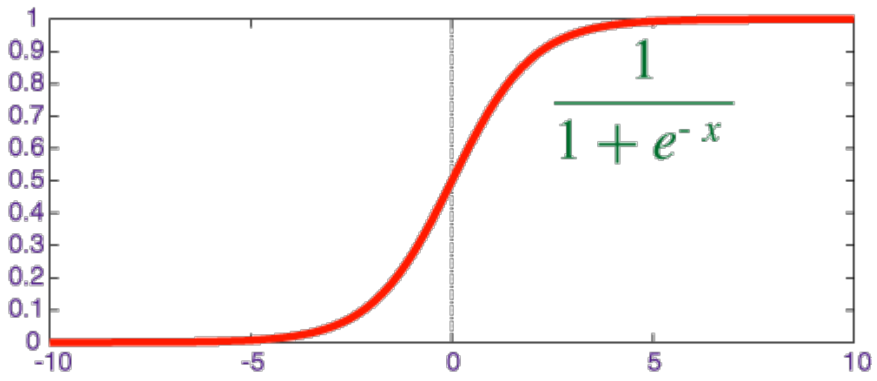


Figure 2: A graph representation of the Sigmoid Function

3.3.2 Gradient Descent

Gradient descent is the most used learning algorithm in machine learning. It is an optimisation technique used to minimise a cost function. The process can be thought of trying to find the lowest point of a mountainous terrain by finding the steepest way down. In Figure 3 below is a 3D visual example of a function that could be minimised using gradient descent. The starting location is randomly selected and from there the optimisation begins. Essentially, the gradient of the function at the current point is found and a step is taken in the negative of that gradient. This means that the next point should be closer towards the local minimum.

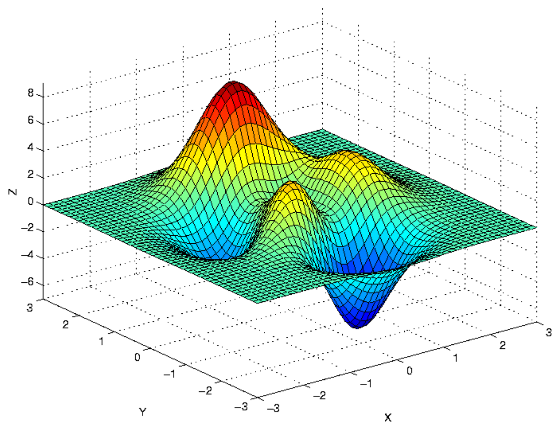
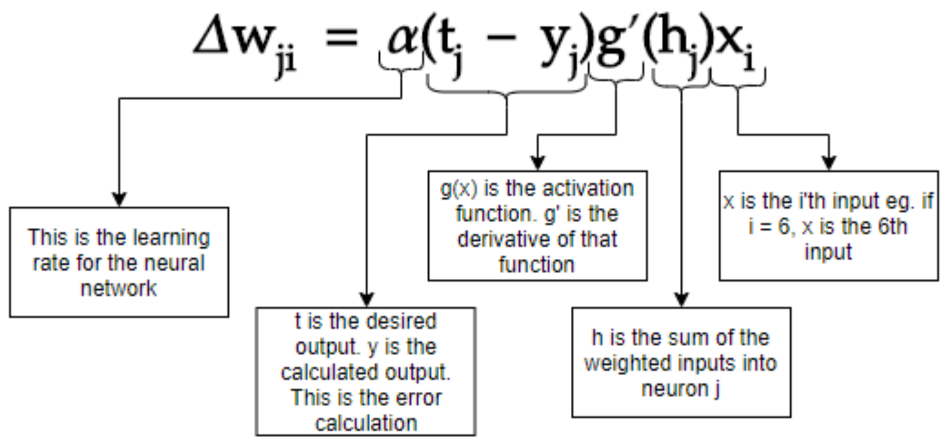


Figure 3: A 3D diagram of a function to be minimised by Gradient Descent

3.3.3 Delta rule and Sigmoid derivative

The Delta rule is a learning rule which is used in backpropagation neural network algorithms in order to update the weights in the network. This is the rule for the update of weight **w** for neurone **j**'s **i**th input.


$\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i$
This is the learning rate for the neural network
t is the desired output. y is the calculated output. This is the error calculation
g(x) is the activation function. g' is the derivative of that function
h is the sum of the weighted inputs into neuron j
x is the i'th input eg. if i = 6, x is the 6th input

The function g(x) that I have chosen is the sigmoid function so in the delta rule, the derivative (gradient function) of the sigmoid function is used. σ is the sigmoid function and σ′ is the derivative.

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Figure 4: The derivative of the sigmoid function

3.4 Choice of Language and Libraries

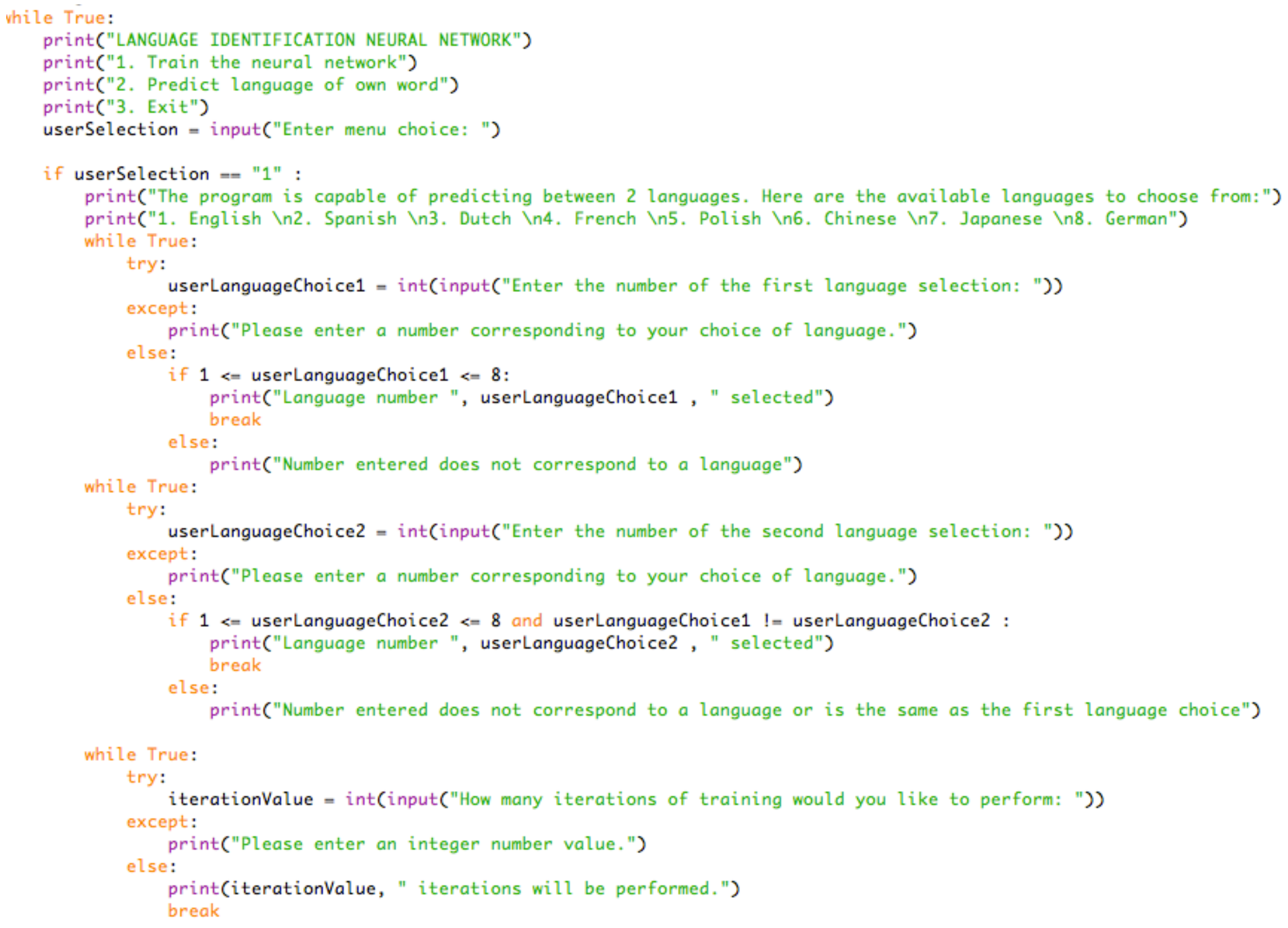
The language I have chosen to code my program in is Python. I chose Python for multiple reasons with some specifically orientated towards machine learning. Firstly, Python is a language that is relatively easy to read and use. This is important for something as complex as machine learning so that more time can be spent on writing the algorithm than debugging it for syntax errors. For someone like me who is not extremely skilled at programming, Python is one of the best languages for beginners to learn and even though I am not a beginner, it was best for me to use a programming language that was easier to read, write and understand as the task of creating a neural network is not that simple. It is also widely used in science and research because of how easy it is to experiment and prototype new ideas without having to write a large amount of code due to requiring only minimal syntax that is easy to understand for it to do what you want.

Another reason I chose to use Python to program my project in, is because it has many libraries that are useful for machine learning. These range from mathematical and statistical libraries such as NumPy, or high-level neural network libraries such as TensorFlow. For my program I chose to just use NumPy as I wanted to just program the neural network at a more basic level but, if I wanted to extend my program to be more complex such as having additional hidden layers in the neural network or allowing multiple languages to be checked at once, I would probably use one of the more higher level libraries to make this possible. I used NumPy in order to perform mathematical functions easier so I did not have to program the dot product function for example as this was not the purpose of my program. This allowed me to focus on the backpropagation neural network algorithm.

Python is a language that I already had knowledge of as well which made it a strong choice for what I language I wanted to code the program in. This made tackling the task of creating a neural network easier and allowed me to make the neural net more complex. The vast availability of resources for machine learning in Python was a final supporting factor in why I chose this programming language.

4 TECHNICAL SOLUTION

4.1 MAIN MENU AND USER SELECTION



```
while True:
    print("LANGUAGE IDENTIFICATION NEURAL NETWORK")
    print("1. Train the neural network")
    print("2. Predict language of own word")
    print("3. Exit")
    userSelection = input("Enter menu choice: ")

    if userSelection == "1" :
        print("The program is capable of predicting between 2 languages. Here are the available languages to choose from:")
        print("1. English \n2. Spanish \n3. Dutch \n4. French \n5. Polish \n6. Chinese \n7. Japanese \n8. German")
        while True:
            try:
                userLanguageChoice1 = int(input("Enter the number of the first language selection: "))
            except:
                print("Please enter a number corresponding to your choice of language.")
            else:
                if 1 <= userLanguageChoice1 <= 8:
                    print("Language number ", userLanguageChoice1 , " selected")
                    break
                else:
                    print("Number entered does not correspond to a language")
        while True:
            try:
                userLanguageChoice2 = int(input("Enter the number of the second language selection: "))
            except:
                print("Please enter a number corresponding to your choice of language.")
            else:
                if 1 <= userLanguageChoice2 <= 8 and userLanguageChoice1 != userLanguageChoice2 :
                    print("Language number ", userLanguageChoice2 , " selected")
                    break
                else:
                    print("Number entered does not correspond to a language or is the same as the first language choice")

        while True:
            try:
                iterationValue = int(input("How many iterations of training would you like to perform: "))
            except:
                print("Please enter an integer number value.")
            else:
                print(iterationValue, " iterations will be performed.")
                break
```

4.2 Random Language Selection

```
def chooseLanguage():
    global languageName1, languageName2
    languages = [languageName1, languageName2]
    languageChoice = random.choice(languages)
    if languageChoice == languageName1:
        outputWordValue = [0]
    elif languageChoice == languageName2:
        outputWordValue = [1]

    return languageChoice
```

4.3 Word matrix creation

```
def createWordMatrix(languageChoice):
    global languageName1, languageName2
    word = ''
    if languageChoice == languageName1:
        lines = open(languageName1 + 'Words.txt').read().splitlines()
        outputWordValue = [0]
        word = random.choice(lines)
    elif languageChoice == languageName2:
        lines = open(languageName2 + 'Words.txt').read().splitlines()
        outputWordValue = [1]
        word = random.choice(lines)


    word = word[0:15]
    if len(word) < 15:
        for x in range(15-len(word)):
            word += " "
    alphabet = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
    base = [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]
    inputWordMatrix = []

    for letter in word:
        letter_descriptor = base
        for pos, char in enumerate(alphabet):
            if letter == char:
                letter_descriptor[pos] = 1
        inputWordMatrix.extend(letter_descriptor)

    return inputWordMatrix
```

4.4 Neural Network Training

```
def sigmoid(x):
    return 1 /(1 + numpy.exp(-x))

def sigmoidDerivative(x):
    return x * (1-x)

def train(inputWordMatrix, outputWordValue, numberOfIterations):
    global weights
    for iteration in range(numberOfIterations):
        inputWordMatrix = numpy.reshape(createWordMatrix(chooseLanguage), (1, 390))
        calculatedOutput = calculate(inputWordMatrix)                         #gradient descent algorithm
        error = outputWordValue - calculatedOutput

        adjustment = numpy.dot(numpy.transpose(inputWordMatrix), error * sigmoidDerivative(calculatedOutput))
        weights = adjustment + weights

        if iteration == numberOfIterations / 4:
            print('25% complete')
        elif iteration == numberOfIterations / 2:
            print('50% complete')
        elif iteration == (numberOfIterations / 4) * 3:
            print('75% complete')
    print("Training complete")

def calculate(inputWordMatrix):
    return sigmoid(numpy.dot(inputWordMatrix, weights))
```

4.5 User's own word input

```
elif userSelection == "2":
    again = True
    while again == True:
        userWord = str(input("Enter a word "))
        userPrediction = (calculate(userInputWordMatrix(userWord)))
        userPrediction = int(round(userPrediction[0]))
        if userPrediction == 0:
            print("I predict", languageName1, ". How did I do?")
        elif userPrediction == 1:
            print("I predict", languageName2, ". How did I do?")

        userAgain = input("Again Y/N ")
        if userAgain == "N" or userAgain == "n":
            again = False
```

5 TESTING

5.1 TEST EVIDENCE TABLE

Link to video evidence: https://youtu.be/RgYkrYgGGYI

Test Number	Description	Test Data	Expected Outcome	Pass/Fail	Evidence
Objective 1 – The user will be able to input a word into the program and it will then display what language it thinks it is					
1	After training the program, option 2 will be selected and the user will enter their own word	"2", "acquaintance"	Display predicted language eg. English	Pass	Video – 0:05
Objective 2 – The program should be able to switch between allowing the user to input a word to allowing the program to train on the data set					
2	Choose option 1 and enter the necessary inputs and then select option two, and then return to the main menu	"1", "1", "5", "1000", "2", "hello", "n"	Goes to option 1, allows for user inputs, returns to main menu and then goes to option 2, allows for user inputs and then returns to the main menu	Pass	Video – 0:24
Objective 3 – The program must check that the word is not over a certain length					
3	Typing in a word that is over 15 characters in length	"2", "misinterpretation"	Should allow for the user input and not break the program. Still predicts the language of the word	Pass	Video – 0:54
Objective 4 – The program must ensure that the user input only contains characters that are able to be processed by the program					
4	Typing in a word that contains numbers	"2", "h3llo"	Should allow for the user input and not break the program. It should ignore the numbers and still predict the language	Pass	Video – 1:16
Objective 5 – The program will train to become more accurate by using a set of training data					
5	***Refer to Statistical Testing section***		The number of correct predictions should improve	Pass	

			as iteration number increases		
Objective 6 – The number of languages will be set but will have the ability to be expanded by providing new training data					
6	Selecting option 1 and having the different languages shown	“1”	The list of languages should be displayed and this is to be based off of the available training data sets that are available	Pass	Video – 1:24
Objective 7 - The prediction given by the computer must be from its calculations performed in the feed forward back propagation algorithm and not from any referencing to stored data					
7	The program has been created to not reference the training data when performing predictions		The program should use only the weights and users word to predict the language	Pass	Image 1
Objective 8 - The program will need to randomly select words from the training data set which will be stored in a separate text file					
8	Option 1 will be selected and the neural network will be trained	“1”, “1”, “5”, “100000”	The program should train from the data set of words which is stored in the corresponding text file	Pass	Video – 1:32
9	Reading from the right text files and choosing a random word		This shows that the words are selected randomly from existing text files	Pass	Image 2 Image 3
Objective 9 - Every letter in the chosen word needs to be converted into a 26 x 1 matrix with corresponding 1’s and 0’s for what that letter is eg. For letter C = [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]					
10	Conversion of the word into a list of numbers		Each letter should change the 0 in its position into a 1	Pass	Image 4
Objective 10 - The program will need to perform the necessary steps of a backpropagation neural network					
11	***Confirmed by objective 11-17 tests***				
Objective 11 - Random starting weights need to be initialised between -1 and 1 for every neuron					
12	Initialisation of weights		390 weights should be randomly chosen and then calculated to be between -1 and 1	Pass	Image 5
Objective 12 - An activation function needs to be chosen and used on all of the generated weights					

13	Sigmoid function implemented and used		The sigmoid function should be applied to the necessary part of the program	Pass	Image 6 Image 7 Image 8
Objective 13 - The inputs and the weights need to be multiplied together as matrices by performing a dot product					
14	Dot product of weights and inputs		The two matrices should dot product together successfully	Pass	Image 9
Objective 14 - The predicted output and actual output needs to be calculated by finding the difference between them					
15	Difference calculation		The error value needs to be found by subtracting the real output by the predicted output	Pass	Image 10
Objective 15 - The derivative of the sigmoid function needs to be included in the program					
16	Presence of the sigmoid derivative function		The sigmoid derivative function implemented as a function in the code	Pass	Image 11 Image 12
Objective 16 - Gradient descent needs to be performed on every neuron to find each adjustment value					
17	Adjustment value calculated		The original word matrix and calculated output needs to have the dot product found and have the sigmoid derivative applied	Pass	Image 13
Objective 17 - The adjustment value needs to be applied to every weight					
18	Adjustment value application		The adjustment value needs to be added to the weights matrix	Pass	Image 14
Objective 18 - The whole process needs to be repeated many times in order to improve the weight values so the predictions become more accurate					
19	Algorithm iterations		The program should have a loop that runs for the amount of times set by the user	Pass	Image 15
Objective 19 - The program will need to import the numpy library for the use of higher level mathematical functions					
20	Import of NumPy		The numpy library will be imported at the beginning of the program	Pass	Image 16

			and will be utilised when performing functions such as dot product		
Objective 20 - The exp, array, and dot functions will need to be imported from numpy					
21	Use of mathematical function		The functions need to perform the desired task correctly and be utilised within the code	Pass	Image 17

5.1.1 Image reference table for testing

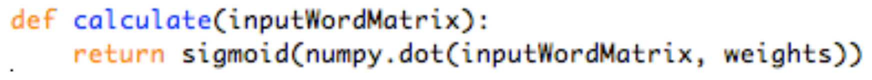
Image Reference Number	Image	Comments
Image 1	 `def calculate(inputWordMatrix):` `    return sigmoid(numpy.dot(inputWordMatrix, weights))`	This is the line of code which makes the prediction. The only components are the word and the weights, which have a dot product applied to it. This shows that no data reference is used in the prediction.

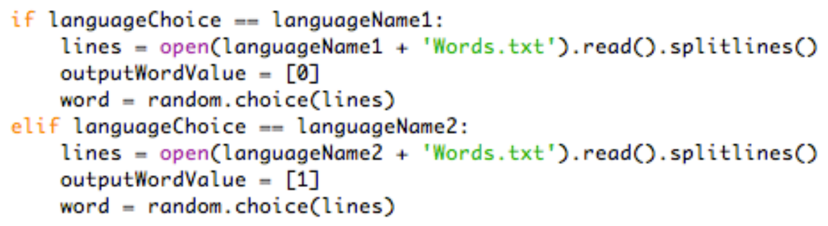
Image 2	 `if languageChoice == languageName1:`
	`    lines = open(languageName1 + 'Words.txt').read().splitlines()`
	`    outputWordValue = [0]`
	`    word = random.choice(lines)`
	`elif languageChoice == languageName2:`
	`    lines = open(languageName2 + 'Words.txt').read().splitlines()`
	`    outputWordValue = [1]`
	`    word = random.choice(lines)`
		This piece of code shows that a word is randomly chosen from a text file which already exists.

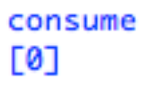
Image 3	 `consume` `[0]`	This is the output of a test script that has just the code in image 2. It shows the word that was randomly selected from a text file and the corresponding output word value.

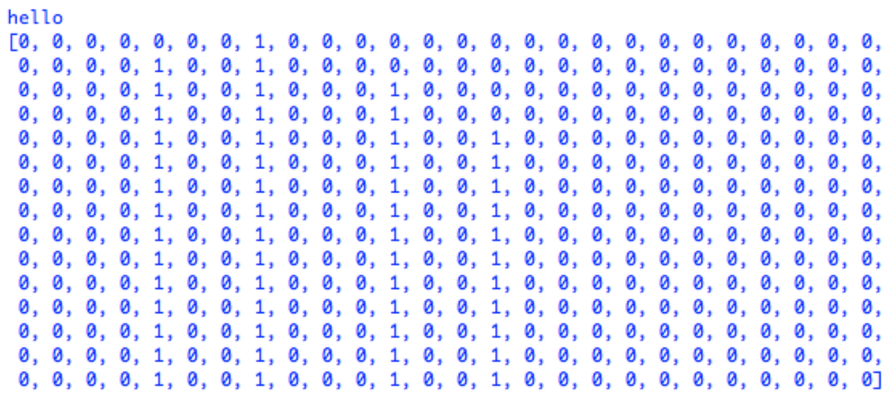
Image 4	
	`hello`
	`[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,`
	` 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
		This is the matrix that is created for the word 'hello'. Each line here is for each letter and as the word is less than 15 letters long, the list repeats itself after the 5th line. The 1's correspond to the positions of the letters in the alphabet, for example, the first 1 is in the 8th position which is the position of h in the alphabet.

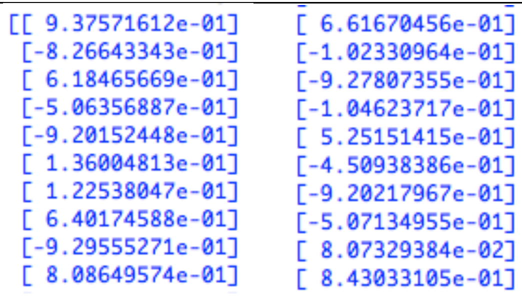
Image 5	
	`[[ 9.37571612e-01]      [ 6.61670456e-01]`
	` [-8.26643343e-01]      [-1.02330964e-01]`
	` [ 6.18465669e-01]      [-9.27807355e-01]`
	` [-5.06356887e-01]      [-1.04623717e-01]`
	` [-9.20152448e-01]      [ 5.25151415e-01]`
	` [ 1.36004813e-01]      [-4.50938386e-01]`
	` [ 1.22538047e-01]      [-9.20217967e-01]`
	` [ 6.40174588e-01]      [-5.07134955e-01]`
	` [-9.29555271e-01]      [ 8.07329384e-02]`
	` [ 8.08649574e-01]      [ 8.43033105e-01]`
		These are just some of the randomly generated weights which are between -1 and 1. The program produces a list of 390 random weights. This has been displayed by testing the initialising weights function separately.

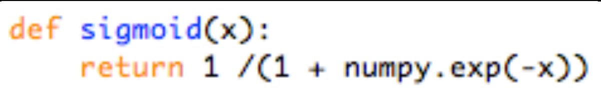
Image 6	
	`def sigmoid(x):`
	`    return 1 /(1 + numpy.exp(-x))`
		This is the sigmoid function that has been implemented as its own function to be used in the program.

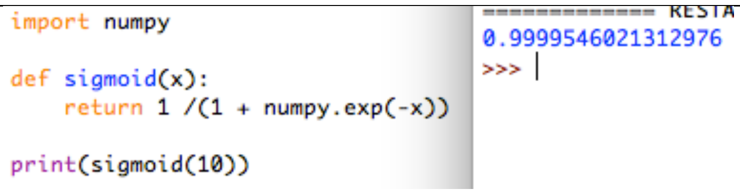
Image 7	
	`import numpy`
	
	`def sigmoid(x):`
	`    return 1 /(1 + numpy.exp(-x))`
	
	`print(sigmoid(10))`
	
	`============= RESTA`
	`0.9999546021312976`
	`>>> `
		This is the test to see if the sigmoid function has been implemented properly in the program. This test displays what 10 is when the sigmoid function is applied to it.

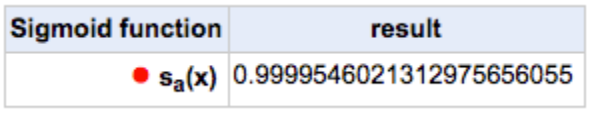
Image 8	 Sigmoid function: ● $s_a(x)$ — result: 0.99995460213129756055	This confirms that my programmed sigmoid function works correctly. This is the answer when x = 10 when inputted into an online sigmoid function calculator. It produces the same answer as Image 7.

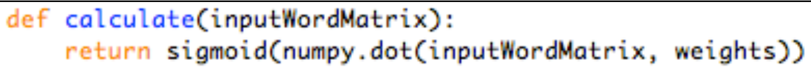
Image 9	
	`def calculate(inputWordMatrix):`
	`    return sigmoid(numpy.dot(inputWordMatrix, weights))`
		This is word matrix and weights have a dot product applied to it by using the NumPy dot product function

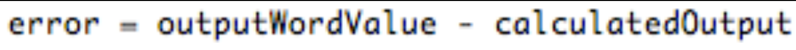
Image 10	 `error = outputWordValue - calculatedOutput`	This line of code finds the difference between the real output and calculated output by a simple subtraction.

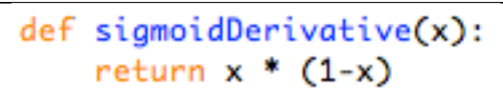
Image 11	 `def sigmoidDerivative(x):` `    return x * (1-x)`	A function in the program has been created so the sigmoid derivative function can be used.

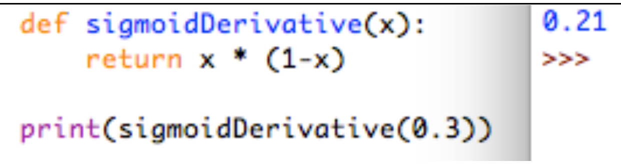
Image 12	 `def sigmoidDerivative(x):`
	`    return x * (1-x)`
	
	`print(sigmoidDerivative(0.3))`
	
	`0.21`
	`>>>`
		This is a test on the same sigmoid derivative function that is used in my program. This input of 0.3 has produced the answer of 0.21. 0.3 * 0.7 = 0.21

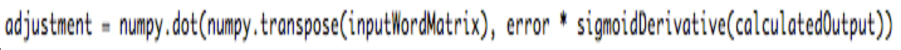
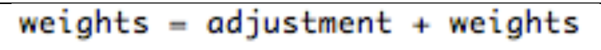
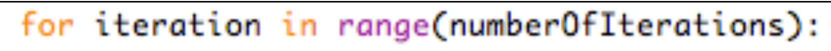
Image 13	 `adjustment = numpy.dot(numpy.transpose(inputWordMatrix), error * sigmoidDerivative(calculatedOutput))`	This is the adjustment value calculation. It shows a dot product between the original word matrix and the calculated output that has been multiplied with the calculated output.
Image 14	 `weights = adjustment + weights`	This is the simple addition of the adjustment values being added to the weights
Image 15	 `for iteration in range(numberOfIterations):`	This runs the loop for the users set number of iterations

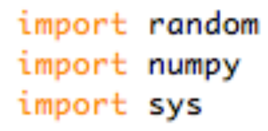
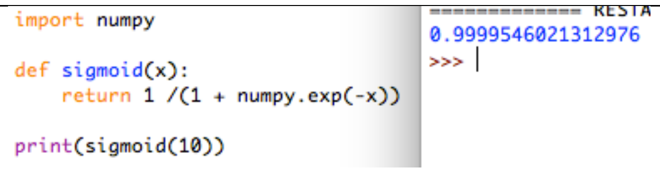
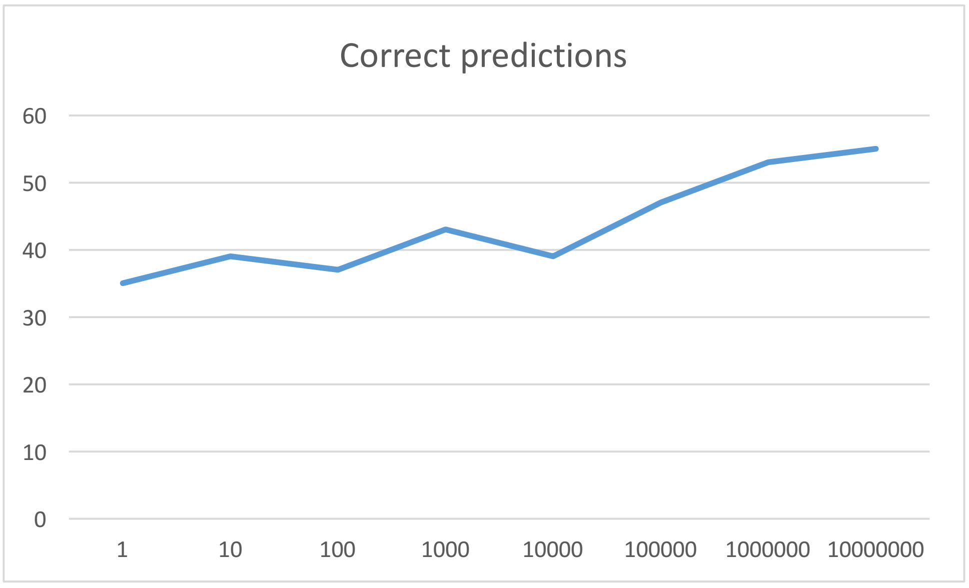
Image 16	 `import random`
	`import numpy`
	`import sys`
		This piece of code is at the beginning of the program which shows that the numpy library along with the random and system library, is imported so it can be used.

Image 17	 `import numpy`
	
	`def sigmoid(x):`
	`    return 1 /(1 + numpy.exp(-x))`
	
	`print(sigmoid(10))`
	
	`============= RESTA`
	`0.9999546021312976`
	`>>>`
		This is an example of the numpy functions being utilised and working. Here, the exp function is used which implements the natural log e into this function.

5.2 Statistical Testing

Here is a line graph showing the improvement the neural network makes when the number of iterations it performs increases by a factor of 10. Overall, there is a positive correlation which would indicate that the neural network is successful in completing objective 5. The data was collected by training the neural network for various iterations and inputting 70 different words from two selected languages. The two languages chosen was English and Japanese as I felt these were the most distinct languages so this would give the clearest results in terms of correct predictions due to there being less ambiguity. If the neural network was not self improving, the expected results would be a generally straight line at around 35 correct predictions as this would mean there is a 50 percent chance that the right language would be picked.



6 Evaluation

6.1 Overall Effectiveness

As stated in my project outline, the aim for this project was to *'create a program consisting of a neural network which will learn to understand the difference between various languages and for it to then predict what language a word is in when given a new input'*. Overall, this aim has been achieved as I have created a functional neural network that is able to train on a data set of words and then uses that training to predict what language it thinks a user's own word is. A small difference between what I have created and my original aim is that my program is only able to train and distinguish between two different languages at a time. This was due to how I chose to optimise my neural network as I had only used the values 1 & 0 which limited me to only choosing between two languages when the neural network trained. In order to try and add the availability of more than two languages, I added additional languages that the neural network could train on, however the user is still only able to choose between two out of the eight languages available.

In my original project outline, I also wanted to make sure that *'the program will not store any data that is inputted into it'* and that *'it will predict the language based off of what it has learned from the training data'*. This has been fully achieved in my program as no data corresponding to what language the certain words are, are stored which means the neural network is fully reliant on the optimisation of its weights and how those interact with every word that's inputted into it for training. The program is only able to read the data text files in order to choose a word to be trained on, but at no point uses that as a reference when predicting a user's own word.

6.2 Evaluation of Objectives

Objective	Evaluation	Possible Improvements
The user will be able to input a word into the program and it will then display what language it thinks it is	This has been done by implementing a second menu option to allow for a user input. This option is very clear and also allows the user to enter another input again once the first one has been calculated without having to go back to the main menu.	Ideally, I would have liked the program to not be menu based but be one functional user screen where there are buttons to start/stop training and a place for the user to input. This would mean the program would need to be done in a GUI package such as Tkinter or PyQt.
The program should be able to switch between allowing the user to input a word to allowing the program to train on the data set	The program has two menu options, one for training and one for user input. This means that the user is able to switch between the two modes set out in the original objective.	As mentioned in the previous objective, to improve this, I would have liked to make the interface graphical and make it even easier to both train the neural network and test it with the user's own word.
The program must check that the word is not over a certain length	The program does this very effectively by utilising array slicing which takes only a certain number of characters of an input. This means that the input used in the rest of the program is always 15 characters maximum	One flaw to the method of using array slicing is that it will cut off any characters after 15. This means that if someone wanted to test a word that is over 15 characters long, the neural network would not be predicting the language of the exact word the user

	as this was important in the functioning of the program.	had entered. An improvement would be changing the way words that are over the maximum length are handled so that perhaps instead the user has to enter a new word, or the whole program as a whole could be reworked so that the number of letters it can process can be flexible and not hard coded in.
The program must ensure that the user input only contains characters that are able to be processed by the program	The way the program handles unknown characters is also very efficient and seamless. There is a set alphabet that the program recognises to be valid characters and will ignore any invalid characters such as numbers.	A problem with this is that if someone were to enter a letter with an accent for example, this is not part of the alphabet currently so the program would overlook that word entirely. To improve on this, the alphabet needs to be extended in the program to accommodate other foreign characters, or a different exclusion system could be implemented such as checking for data types such as integers.
The program will train to become more accurate by using a set of training data	This is done quite efficiently in a function I created. It utilises the fact that the language is also the first word of the training data text file names. This allows the reading of the many different text files and doesn't require a separate line of code for each language. It will read each line of the text file and then pick a random line.	This relies on the fact that the training data only has one word per line and that all the words are under 15 characters long. The process of reading the whole text file and then choosing a line could become less efficient if the text file is extended with a lot more lines of words. To improve, a system where the whole file does not have to be read and split up could be done instead.
The number of languages will be set but will have the ability to be expanded by providing new training data	The program is expandable by adding new text files of training data. However, it must be named correctly with the language name followed by 'words' afterwards. This is so that it can fit in with the way the text file is selected and read. The new language name also has to be added to the language list and the display list that is given at the start of option 1.	To improve the expandability, the program would need to have easier, automated way to add a new training data file and for it to add it to the list of languages itself so that the user does not have to edit the code directly.
The prediction given by the computer must be from its calculations performed in the feed forward back propagation algorithm and not from any referencing to stored data	This has been achieved completely as at no point in the prediction part of the program is there any use of the text files or any stored values apart from the weights and the user's word	This is an objective I don't think can be improved on significantly as it has been completed fully in the program.

	which has been converted into a list consisting of 1's and 0's.	
As the program runs, the program should tell the user what generation it is on in order to monitor the progress of the training	This has been almost done to the way I had originally planned. In my program, I have attempted to achieve this objective by displaying the percentage of how far through the training iterations the program is. It takes the number of iterations it will perform and will display at 25, 50 and 75 percent completeness that it has reached that percentage of iterations. This helps to give an indication of the program running.	To improve on this objective, I would have liked to have included a progress bar style indicator for how far through the neural network was with training. Alternatively, an animated loading indicator such as dots moving just make clear to the user that the program is still running, as sometimes the training can take quite a while if performing a high number of iterations.
The program will need to randomly select words from the training data set which will be stored in a separate text file	The program does take its training data from a separate text file and it does do this for every single iteration. As mentioned in one of the previous objective evaluations, the program does have to read the whole file before selecting a word which may become inefficient if the training data files have an increased number of words	The improvement for this would be to program a way for words to be taken from a text file more efficiently without reading the whole file. The storage of words could be improved by using a data base instead so that separate files wouldn't have to be opened every time.
Every letter in the chosen word needs to be converted into a 26 x 1 matrix with corresponding 1's and 0's for what that letter is eg. For letter C = [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]	In my program, the word matrix function effectively replaces the corresponding 0 to the letter position in the alphabet to a 1 well. It makes sure to go through each letter of the word and find where it is in the alphabet which I set in the program.	An improvement to make would be to make the length of the matrix expandable to fit with an expandable alphabet. This could be done by using a loop to add 0's to a list until it's the number of items in the alphabet list. This would future proof it and reduce the amount of hard coding in the program.
The program will need to perform the necessary steps of a backpropagation neural network	The program performs the backpropagation algorithm in the correct order and it is simple to follow through to see how the code corresponds to the steps in the algorithm when written in plain text.	To help with debugging in the future, it would be better to split the calling of the functions up so that multiple functions aren't called on one line. Even though it is not as neat, this would help with the functionality of the written code.
Random starting weights need to be initialised between -1 and 1 for every neuron	This begins at the beginning of the program and does this in just one line of code. It uses the random library to make sure that the starting weights are completely random and a few basic maths operations ensure	To improve for the future, a way of saving the weights after being optimised could be included. A way to import weights would also need to be added but in general this would be a saving and loading feature for the program. For example, if a very

	that every single weight is between -1 and 1.	accurate optimisation was reached, the user may want to save it for future use.
An activation function needs to be chosen and used on all of the generated weights	The activation function chosen for my program is the sigmoid function as this placed all the numbers between 0 and 1. This function was a very suitable choice and worked well with my program for optimisation.	For further investigations into neural networks, a way of changing between different activation functions could be included so that comparisons could be made on the effects of activation functions on the functioning of the neural network.
The inputs and the weights need to be multiplied together as matrices by performing a dot product	This was done easily using the dot product function including with the numpy library. This meant that the process of dot product was more accurate and reliable that programming this function myself and then attempting to use it in the neural network.	No possible improvements for this objective that would significantly better the program.
The predicted output and actual output needs to be calculated by finding the difference between them	This was also another simple calculation of just subtraction between two values.	No possible improvements for this objective that would significantly better the program.
The derivative of the sigmoid function needs to be included in the program	As the sigmoid function was used as the activation, the derivative had to be used to find the gradient at the points. This was also easily implemented into the program by making it its own function and using the natural log function included with numpy. This helped to ensure that the calculated numbers were accurate and the function could be used anywhere else if needed to.	If the improvement mentioned in the sigmoid function objective was to be applied, the derivative of the function would also have to follow suit accordingly as the gradient descent algorithm requires the gradient function of the activation function to be found.
Gradient descent needs to be performed on every neuron to find each adjustment value	This has been done in my program by simply using variables which include the whole list/matrix in the calculations. A dot product is performed which requires the use of matrices and within the gradient descent algorithm, the sigmoid derivative has been simply applied to every value in the calculated output matrix.	To make it again easier to debug, splitting the algorithm up more in the code would make it easier to identify any problems during calculation and make it easier to trace through the code.
The adjustment value needs to be applied to every weight	This has been done with adding the adjustment values matrix to the weights matrix. The matrices line up accordingly making it very easy to just add them together.	No possible improvements for this objective that would significantly better the program.

The whole process needs to be repeated many times in order to improve the weight values so the predictions become more accurate	The whole training function is one loop that runs for the number of times the user chose originally. This makes sure that the whole algorithm is performed for every iteration to properly optimise the weights.	An improvement that could be made is to make it possible for the program to pause iterating to test the accuracy of the neural network at the number of iterations. Also, a possible addition of an accuracy indicator could be a useful amendment to show the rate of optimisation by the neural network.
The program will need to import the numpy library for the use of higher level mathematical functions	The numpy library is imported right at the beginning of the program which means its accessible throughout the entire program. This was very useful in making the code much more efficient and was necessary for my program.	No possible improvements for this objective that would significantly better the program.
The exp, array and dot functions will need to be imported from numpy	These functions specifically were used in my program and made the program much more effective and robust.	No possible improvements for this objective that would significantly better the program.

6.3 Conclusion

Overall, the program I have created has helped answer the question *'if neural networks are an effective solution to language identification'*. Although my neural network is not as complex as what a real-life program may be if this method of language identification was to be used commonly, I think that my program has shown that neural networks being used for this purpose would be useful, especially as it can become more accurate with time and increasing the amount of training data. It can also be refined with adding other mathematical calculations to the algorithm which would enhance its accuracy and ability to perform pattern recognition better. This program acts almost as a first step towards smarter programs which can improve themselves. It is showing on a more basic level how machine learning can be used to solve problems in a way that reflects human learning styles. My program I don't think completely answers the question of the effectiveness of neural networks for language identification as this would require development of various neural networks and comparisons between other methods of language identification programs as well, however it does show that neural networks are an option that should be considered more heavily, but not just for language identification, but for other tasks that require some form of learning and improvement.