|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Learner Name | Matthew Charles Beardwell |  | Candidate number | 3010 |
|  |  | |  | |
| Centre Name | King’s College London Mathematics School |  | Centre Number | 10953 |
|  |  | |  | |
| Unit Name | Can my computer guess the language of a word? |  | Unit Number | P304 |

**Project Activity Log**

|  |  |
| --- | --- |
| Date of Log Entry | Task Completed |
| 09/07/17 | I have been inspired to make a neural network by being subscribed to the YouTube channel belonging to Siraj Raval. I have been researching the architecture of the most classic network - the feedforward neural network and have watched tutorials on how the neurons send the data forward. |
| 14/07/17 | I copied code by Siraj Raval to the Python3 editor and ran it to observe the outputs and speed of the program. I then noted that he had used what is called an 'activation' function on each neuron before the data is sent forward again and to 'update' these later, I needed to understand how to differentiate this function. |
| 15/07/17 | I learnt the derivative of the activation function he used and set up my own gradient descent algorithm which mainly copied a lot of his own code to find the line of best fit of some data points. |
| 28/07/17 | After some thought about what my network will do, I watched some videos by WelchLabs to understand most of how the network would feed data forward and an experiment by 'carykh' on YouTube had such good results in word processing, I decided I wanted to take a word as an input to the network and let it learn the output for it. I have made an optimal hangman algorithm during my AS Computing lessons which I will repurpose to output information on the words it takes in. |
| 29/07/17 | I completely rebuilt the hangman code which took the same algorithm but instead it interfaced with itself rather than a person on a keyboard. It would then output the number of guesses it took for each hangman word into a text file along with the word itself. |
| 06/08/17 | I found out about 'one-hot encoding' which will allow me to take in a word as an input to my neural network when it's built. This was a suggestion made by the YouTube channel ‘carykh’ on a video of his and I understood that it would perform better theoretically than another standard method. |
| 10/08/17 | I have seen that you can use the simplification of the algorithm by using matrix mathematics, which I have understood about 50% of. I think I will consider this in the future when building my code. I want to build my program with the mathematics rather than use something that already has it built in because it gives me an opportunity to learn and research in further detail. I know that I need to propagate the data forward through the network and then propagate the errors backwards to update the network. There will need to be a way of inputting the words and a way of outputting a prediction based on what the task is. |
| 20/08/17 | The main basis of the network has been built but the hurdle of training the network has arisen. I need now to look up how to update the weights on the network by using 'backpropagation'. I read an online book by Michael A. Nielson on deep learning and I now understand how the four main backpropagation lines in the algorithm work. |
| 27/08/17 | I have collected the list of resources to learn from and to write up on my literary review. |
| 02/09/17 | I have started the literary review on the collected sources. |
| 10/09/17 | I scrolled through the Wikipedia page of backpropagation and watched the Welch Lab YouTube instructions on building a neural network mathematically titled “Neural Networks Demystified”. In the end, the most useful source I found on the type of neural network I was trying to build was Chapter 2 of Michael A. Nielson’s “Neural Networks and Deep Learning” as I mentioned earlier. |
| 16/09/17 | I have completed the literary review. |
| 22/09/17 | I made notes on the four main backpropagation algorithms and tried to understand their proofs. I understood all proofs apart from the fourth but I understand how to implement it and the main idea behind it wasn’t too farfetched. |
| 30/09/17 | I'm going to start by programming the hangman network. I have collected a large English dictionary and written all of them into a file with their difficulties using a program I wrote. |
| 05/10/17 | I spent very many hours adding the backpropagation algorithms into my program as it wasn't intuitive how to translate the calculus into Python code. |
| 15/10/17 | Many more hours have been put into debugging my code. Every time a bug is fixed, a new one appears. I must have run the program fixed and error, and rerun again about 100 times.  While debugging, I came across errors saying that matrices that were being multiplied or added were not the same shape – a requirement for combining matrices. This suggested that I was, hopefully, facing some ‘fencepost’ errors in my code. This means that some of my indices (numbers I was using to access matrices in my lists) were too big or too small by one. I was convinced that this was due to my poor code layout and lack of time I had spent thinking about how my network was set up.  I reread the equations and made notes on each including which matrices belonged to which layer of the network so that the indices matched in my code. I had a big list of weights and biases in the network and it turns out saying “Multiply weight 2 with layer 1” was easier than saying “Multiply weight 1 with layer 1” even though they were supposed to mean the same thing. This fixed most of my errors and the only disadvantage of making it like this is that the list storing the matrices had to have an empty first slot because there was no weight matrix 1 or bias matrix 1 as the layer 1 was the input and had no “arrows” going into it.  Weight 2 (represented by the arrow) belongs to layer 2 not layer 1, as I had it designed before  I also had an issue with the output always “maxing out” at 1 when the range of outputs could have only been between 0 and 1 because of the sigmoid activation. I looked this error up and found that it may be due to too large learning rates and too many nodes in the input with respect to the size of the hidden layers. I decreased the size of the learning rate as I didn’t believe the issue was down to the size of the input and the hidden layers and this seemed to have fixed the issue, but the network had another issue with the output always staying the same no matter what I input into the network. I will look into this later. |
| 19/10/17 | The network now outputs predictions but they are all roughly the same. After extensive investigation and tweaking of parameters, I think this may be because there is no correlation between the input word and its Hangman difficulty. This is where I will move on to having the program guess a word's language and I hope these two variables are correlated. |
| 25/10/17 | I have assembled 35k words for each of Latin, German, French, Italian, Greek and Japanese. I used a program to write them mixed up into a file with their language written next to them as a number. |
| 29/11/17 | I wrote my methodology and design principles for this new program which took a few days and I will begin programming it. |
| 11/11/17 | I have changed the input to the old network to now accept words in the one-hot encoding format. |
| 16/11/17 | I changed the format of the output of the network to be an array of 6 numbers, representing the prediction of the languages. I also added functions to change the input word into the matrix format and to change the input language as a six-item array for backpropagation. |
| 05/12/17 | I added a large section of the program to train and test the network and output the information to files in my file system. I learned in the Python3 documentation pages how to create folders and how to add lines to CSV files in those folders.  This included writing code to save weights and biases to files and writing a function to import them back in again, if I were to choose to continue running iterations and getting results on a network I was using before. This happened to be very useful and my computer get crashing during long runs of iterations on the networks which meant I didn’t completely lose all of my progress. To do this, I found the CSV Writer tool in Python and I also used the standard saving of text files to save the hyperparameters. I also looked up the ‘os’ module so that I could create folders to organise the results and the save files. |
| 14/01/18 | I split the program into one file with the network in it, and a main one in which I import the network as a class and instantiate it. When I write algorithms with which to use the network (if it is to train it, test it, import save files of weights and biases, etc.) then the program will not have to be many lines long which makes it confusing to scroll through. |
| 20/01/18 | I also wrote a function to estimate the time left to finish the iterations of training as it would take many hours to complete and I wanted to know when I could come back to use my computer for other purposes. |
| 27/01/18 | Because of issues I was having with various activation functions and their derivatives in the Hangman code, I left mine as the sigmoid activation function. Some gave runtime errors that their function wasn’t defined for the values I was giving it or that I was dividing by zero or some other annoying issue. I found it difficult and time consuming to fix these errors for even up to five different activation functions and some errors I found impossible to debug such as one involving the ‘maxing out’ of the neurons. The function that gave me the least issues was the sigmoid so I stuck with it. |
| 03/02/18 | I graphed the results and made conclusions on them in my EPQ. I also edited the program to output confidences and store them in another CSV file. This allowed me to also make conclusions on the confidences of the networks choices. |
| 16/02/18 | I rewrote the prediction function in the network class so that it will output the predictions to the console nicely and the commented out the training and testing algorithm to make way for a new loop – it now imports my best network, the 200n1h network, and allows users to input words to make predictions. |
| 24/02/18 | The main program is finished, the data text files are stored with it and the main body of the network’s writeup is finished. I will conclude that the network worked with a higher-than- random-chance accuracy. I was surprised the network even worked at all given the substantial amount of time I spent trying to implement the backpropagation algorithm. |
| 03/03/18 | I have made my conclusions on the data that I have collected and the progress I have made. |
| 07/03/18 | I made a presentation on how my network works from a very high-level perspective. This was so my audience could understand what I was talking about sufficiently so that I could explain the program’s results. I made animations and graphs to aid in the explanation and, after, I received questions from the audience that convinced me that they were interested and understood my explanation so I think the presentation went well. |
| 11/03/18 | I have tidied up my documents and have made any corrections suggested by my coordinator. These included formatting changes, adding a bibliography to tidy up citations, adding better evidence of monitoring, and including code screenshots. I have also added the necessary details for submission to the exam board. |