How can graph theory, gradient descent and partial derivatives be used to develop a letter recognising neural network?

Subject: Mathematics

Word count: 3997

Introduction:

My handwriting is atrocious, most people have to ask me what I have written every few words. I personally believe that they are exaggerating and can actually read my handwriting. To prove that my handwriting is genuinely legible, I will use graph theory and forward/backward propagation to train a neural network to read letters written by me and others. A neural network is a mathematical object that mimics the way a human brain learns, in this case, learning how to read letters accurately.

In this essay I will answer the question **"How can graph theory, gradient descent and partial derivatives be used to develop a letter recognising neural network?"** Firstly, I will introduce how visual images of letters can be represented by numbers. Followed by an introduction to graph theory and how it is helpful in the representation of neural networks. I will also use a methodology known as gradient descent of which includes the usage of partial derivatives to minimize the error of the neural network.

The aim of this essay is to create a neural network that can accurately 'read' the letters that I draw, to prove that people exaggerate how illegible my handwriting really is.

A variety of sources were used in my research including videos by Stanford university of engineering (8), Grant Sanderson (3Blue1Brown) (6)(7), the book Graph Theory by Robin J Wilson (10) and a dataset provided by Kaggle (5).

Body

Representing images mathematically:

Our neural network will need to 'read' images, unfortunately our neural network wasn't born with eyes, so we will need to convert images into some form of mathematical representation to input into our neural network.

Images can be represented as an x by y by z matrix, where x and y are the dimensions of the image, and z is the number of datapoints per pixel. In most cases,.the image has 3 data points per pixel, the amount of red, green and blue in the pixel, also known as RGB values. RGB values are integers and can range from 0 to 255.

For example, this image of a landscape can also be represented as a matrix.

Fig 1. An image of a landscape with the dimensions of 1000×667. (9)

```
[48 65 49][43 60 44][43 60 44][45 62 46][48 66 50][58 76 60][63 81 65][61 79 63][41 62 45][43 64 47][48 69 52][75 96 79][70 91 74][54 75 58][
[42 58 45][40 56 43][48 64 51][49 65 52][41 59 45][47 65 51][61 79 65][66 84 68][31 52 37][48 69 52][57 78 61][62 83 66][63 84 67][53 74 57][
[38 54 43][39 55 44][40 56 43][44 60 47][42 59 43][52 69 53][68 86 70][63 82 63][37 55 39][53 72 53][67 88 69][66 87 68][55 76 59][56 77 60][
[43 59 48][42 58 45][40 56 43][46 63 47][46 63 47][55 72 54][67 86 67][58 77 58][44 63 44][51 70 51][57 78 59][56 77 58][52 73 56][55 76 59][
[43 59 46][41 57 44][39 55 42][48 65 49][50 68 52][56 75 56][63 82 63][52 71 51][56 75 55][54 73 53][52 74 53][51 73 52][52 73 54][57 78 59][
[46 62 49][48 65 49][49 66 50][59 76 58][62 81 62][62 81 61][65 84 64][55 74 54][56 75 55][52 71 51][49 71 50][48 70 49][51 72 53][57 78 59][
[44 62 46][49 68 49][53 72 53][62 81 61][60 79 59][54 73 53][56 75 55][50 69 47][45 64 42][45 64 42][45 67 46][43 65 44][42 64 43][48 70 49][
[47 66 47][54 73 53][55 74 54][59 78 58][53 72 52][43 62 42][46 65 43][45 64 42][49 68 46][50 69 47][48 70 49][41 63 42][34 56 35][37 59 38][
[60 79 59][64 83 61][60 79 57][61 80 58][55 74 52][46 65 43][52 71 49][56 75 53][57 76 54][54 73 51][48 67 47][39 58 38][33 52 32][39 58 38][
[59 78 56][60 79 57][53 72 50][52 71 49][49 68 46][43 62 40][53 72 50][60 79 57][53 72 50][46 65 43][38 57 37][32 51 31][33 52 32][46 65 45][
[52 75 49][60 83 57][50 73 47][37 60 34][32 51 29][41 60 38][54 73 53][42 61 41][43 62 42][44 63 43][36 53 35][40 57 39][51 68 50][55 72 54][
[42 65 39][56 79 53][54 77 51][45 68 42][39 58 36][39 58 36][47 66 46][41 60 40][37 56 36][43 62 43][42 59 41][48 65 47][49 66 48][42 59 41][
[41 64 38][55 78 52][54 73 51][50 69 47][48 67 47][43 62 42][44 61 43][44 61 43][35 52 34][39 56 40][41 55 40][48 62 47][45 59 44][35 49 34][
[49 72 46][53 76 50][44 63 41][41 60 38][49 68 48][45 64 45][41 58 40][47 64 48][54 71 55][51 68 52][45 59 46][45 59 44][38 52 39][31 45 30][
[50 73 47][48 71 45][39 58 36][35 54 32][43 62 42][40 59 40][36 53 37][48 64 51][56 70 57][57 71 58][53 65 55][50 62 50][41 53 43][36 48 36][
[46 69 43][47 70 44][48 67 45][42 61 41][40 59 40][36 54 38][35 51 38][49 65 52][35 48 38][41 54 44][49 61 51][55 67 57][43 55 45][34 46 36][
[46 65 43][46 65 43][52 71 51][47 66 46][40 57 39][37 54 38][37 51 38][46 59 49][42 53 45][38 49 41][42 53 45][53 64 56][41 51 43][25 35 27][
[45 64 44][39 58 38][46 65 45][41 60 41][35 52 36][36 53 37][34 47 37][36 49 39][59 70 62][37 48 40][31 42 34][47 58 50][41 51 43][26 36 28][
[33 49 36][35 52 36][42 59 43][44 61 45][42 56 41][38 52 37][35 47 37][30 42 32][52 63 55][43 54 46][34 44 35][36 46 37][43 53 44][35 45 36][
[22 38 27][31 47 34][50 66 53][59 76 60][57 71 56][55 69 56][47 59 49][28 40 30][41 52 44][51 62 54][42 52 43][31 41 32][37 47 38][39 49 40][
[28 44 33][30 46 33][50 66 53][70 87 71][74 88 75][70 84 71][53 65 55][25 37 27][34 45 37][60 71 63][50 60 51][24 34 25][33 43 34][46 56 47][
[25 41 30][30 46 35][49 65 52][70 87 71][74 88 75][57 71 58][38 50 40][28 40 30][35 46 38][62 73 65][51 61 52][25 35 26][37 47 38][51 61 52][
[18 34 23][39 55 44][54 68 55][58 72 59][51 65 52][30 44 31][22 34 24][35 46 38][36 46 38][55 65 57][47 57 48][35 45 36][48 58 49][53 63 54][
[51 67 56][57 73 62][48 62 49][34 48 35][29 43 30][22 36 23][23 34 26][36 47 39][32 42 34][45 55 47][47 57 48][48 58 49][54 64 55][48 58 49][
[76 89 80][52 65 55][37 51 38][35 49 36][31 43 31][25 37 25][29 39 31][34 44 36][32 42 34][46 56 48][55 65 56][53 63 54][47 57 48][41 51 42][
[54 67 58][21 34 25][34 47 37][62 76 63][45 57 45][20 32 22][25 35 27][33 43 35][36 46 38][53 63 55][63 73 64][53 63 54][37 47 38][35 45 36][
[28 39 31][16 27 19][54 66 56][71 83 73][45 55 46][23 33 24][23 33 25][43 53 45][57 67 59][44 54 46][39 49 40][56 66 55][37 47 36][33 43 34][
[16 27 19][43 54 46][71 83 73][56 68 58][31 41 32][33 43 34][43 53 45][48 58 50][39 49 41][35 45 36][39 49 40][52 62 51][28 38 27][33 43 32][
[17 26 21][65 75 67][71 81 72][42 52 43][30 40 31][44 54 45][52 59 52][38 45 38][29 36 29][38 45 37][43 53 44][50 60 49][25 35 24][38 48 37][
[40 49 44][67 76 71][51 61 53][43 53 44][54 64 55][49 59 50][39 46 39][19 26 19][30 37 30][44 51 43][44 54 45][46 56 45][29 39 28][41 51 40][
[59 68 63][58 67 62][38 48 40][54 64 56][72 79 71][45 52 44][24 31 24][19 25 21][31 38 31][45 52 44][34 44 35][39 49 38][34 44 33][41 51 40][
[55 64 59][44 53 48][37 47 39][55 65 57][60 67 59][32 39 32][21 27 23][30 36 32][31 38 31][40 47 39][29 39 30][41 51 40][41 51 40][42 52 41][
```

Fig 2. The same image of the landscape represented as a matrix $M(1000,\ 667,\ 3)\{M_{x,y,z} \in \mathbb{Z} | 0 \leq M_{x,y,z} \leq 255\}$. $\mathbb{Z}$ is the set of integers.

These matrices will act as the input to our neural network, and will be like the neural network 'reading' the image of the letter. In order to make the neural network more applicable, pictures of letters have been grayscaled, so the neural network can perceive any colour used to write the letter. Matrices of grayscale images use the percentage brightness rather than the 0-255 system RGB uses. 0% or 0 would be black and 100% or 1 would be white. Images will also all be resized to 28×28 pixels as constant dimensions are crucial for the neural network. 28×28 pixels was chosen as it doesn't sacrifice too much detail while also keeping computational time relatively low.

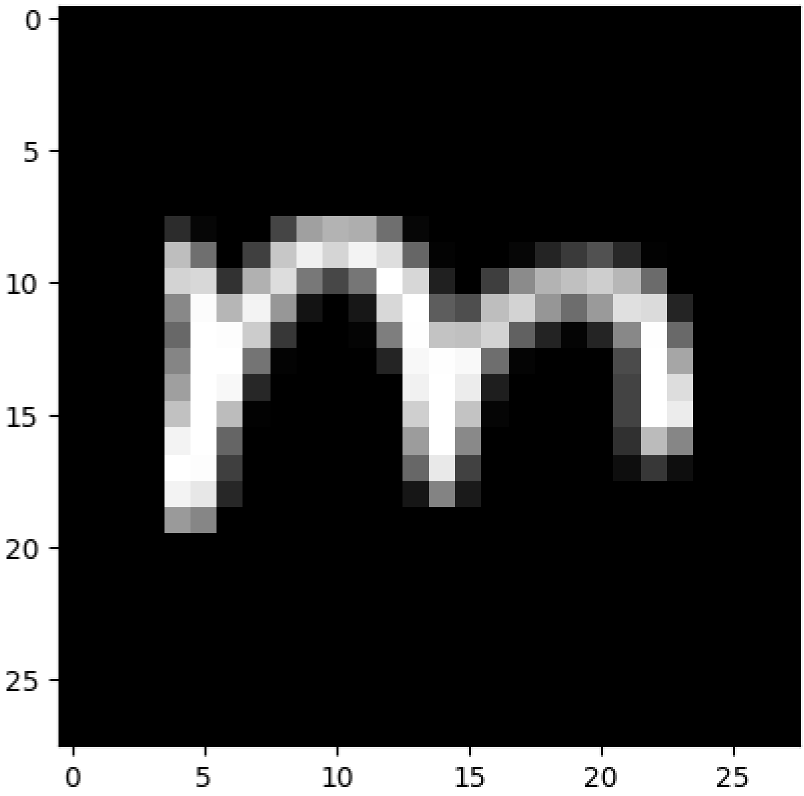

Fig 3. A grayscale image of the letter 'M' with the dimensions of 28×28.

```
[[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. 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. 0.173 0.024 0. 0. 0.271 0.627 0.702 0.682 0.435 0.024 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.745 0.435 0. 0.251 0.78 0.941 0.835 0.953 0.871 0.400 0.012 0. 0. 0.027 0.141 0.227 0.322 0.161 0.008 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.827 0.847 0.196 0.694 0.867 0.471 0.278 0.463 1.000 0.843 0.125 0. 0.243 0.545 0.706 0.753 0.808 0.718 0.42 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.533 0.988 0.714 0.953 0.592 0.075 0. 0.090 0.847 1.000 0.365 0.306 0.745 0.827 0.588 0.427 0.604 0.875 0.859 0.141 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.408 1.000 0.992 0.800 0.216 0. 0. 0.016 0.494 0.996 0.769 0.753 0.827 0.384 0.137 0.016 0.145 0.533 0.996 0.412 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.522 1.000 1.000 0.455 0.012 0. 0. 0. 0.145 0.969 0.996 0.973 0.431 0.016 0. 0. 0. 0.294 1.000 0.651 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.624 1.000 0.969 0.153 0. 0. 0. 0. 0. 0.945 1.000 0.922 0.118 0. 0. 0. 0. 0.263 1.000 0.867 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.757 1.000 0.737 0.008 0. 0. 0. 0. 0. 0.812 1.000 0.769 0.020 0. 0. 0. 0. 0.263 1.000 0.925 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.953 1.000 0.396 0. 0. 0. 0. 0. 0. 0.612 1.000 0.537 0. 0. 0. 0. 0. 0.192 0.733 0.525 0. 0. 0. 0.]
 [0. 0. 0. 0. 1.000 0.992 0.247 0. 0. 0. 0. 0. 0. 0.404 0.914 0.259 0. 0. 0. 0. 0. 0.055 0.216 0.055 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.953 0.906 0.153 0. 0. 0. 0. 0. 0. 0.086 0.514 0.102 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.604 0.525 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. 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.]]
```

Fig 4. The same image of the letter ‘M’ represented as a matrix. $M(28)\{M_{x,y} \in \mathbb{R} | 0 \leq M_{x,y} \leq 1\}$.

Creating a training dataset:

We can begin preparing images for our neural network to look at and learn. The more images the more it can learn and adapt to be able to read letters more accurately. The US National Institute of Standards and Technology provides 372450 different matrices of all 26 capital letters in the English alphabet (5). In order to not add dimensional complexity to the project, the dimensions of each matrix containing image data is flattened from 28×28 to 1×784. In other words, the next row in the matrix is appended to the back of the first row. Once each image data matrix is flattened, it is added to the large dataset matrix resulting in a 784×372450 matrix. The dataset matrix can be thought of as 372450 examples of pictures containing 784 pixels.

Lastly, one row will be added to indicate the actual letter that was written, this will be used to correct the neural network. (Resulting in 785×372450 matrix). Pixel numbers go from left to right, top to bottom. (Top left is 1, top right is 28, bottom left is 757, bottom right is 784).

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

Fig 5. An example matrix on the left with dimensions 3×3 is 'flattened' to the matrix on the right. Resulting in a matrix of dimensions 1×9.

	Letter(L) (Index in alphabet)(Indexing begins with 0, 'A'=0) $\{L \in \mathbb{Z} \mid 0 \leq L \leq 25\}$:	Pixel 1 (P_1) grayscale value $\{P_1 \in \mathbb{R} \mid 0 \leq P_1 \leq 1\}$:	Pixel 2 (P_2) grayscale value $\{P_2 \in \mathbb{R} \mid 0 \leq P_2 \leq 1\}$:	…	Pixel 784 (P_{784}) grayscale value $\{P_{784} \in \mathbb{R} \mid 0 \leq P_{784} \leq 1\}$:
Image 1	12	0.36	0.00	…	0.48
Image 2	4	0.18	0.27	…	1.00
…	…	…	…	…	…
Image 372449	9	0.00	0.00	…	0.00
Image 372450	25	0.33	0.00	…	0.00

Figure 6. A preview of the data set.

Introduction to graph theory:

It is time to begin making the neural network itself. The neural network is an example of a graph from graph theory. Graph theory is the branch of mathematics where graphs are used to represent different mathematical structures such as paths, networks and friendships (10). Graph theory allows computers to interpret different datapoints with differing importance, allowing computers to decide what information is relevant. Below is an example where graph theory might be used to represent the relationship between 5 friends.

Consider a group of 5 friends at a party. Each friend can be represented as a node. Nodes can also represent numbers.

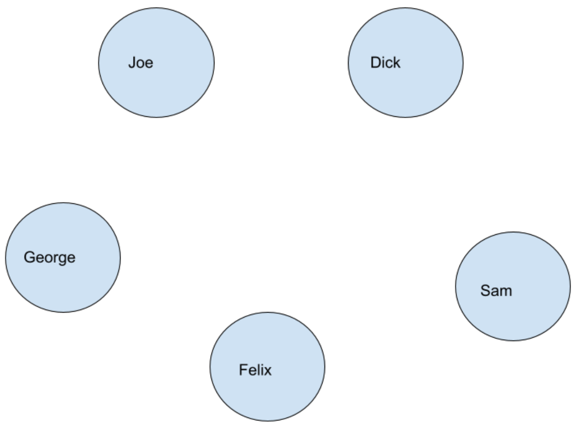

Fig 6.1. A graph of 5 people/nodes.

Joe and George begin to talk and form a relationship, this is known as an edge. Dick, Felix and Sam also form a relationship.

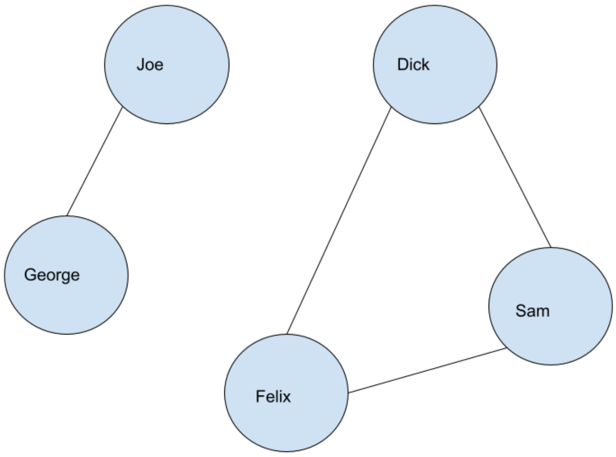

Fig 6.2. A graph with an edge between Joe and George, Felix and Sam, Felix and Dick and Dick and Sam.

Dick becomes very fond of Sam and they get along well, however Dick and Felix struggle to get it off. We can add a value to the edge to represent the strength of their relationship. This is known as the weight of the edge. In this case the weight will be a numerical percentage, where 1 means a great relationship and 0 means a terrible one.

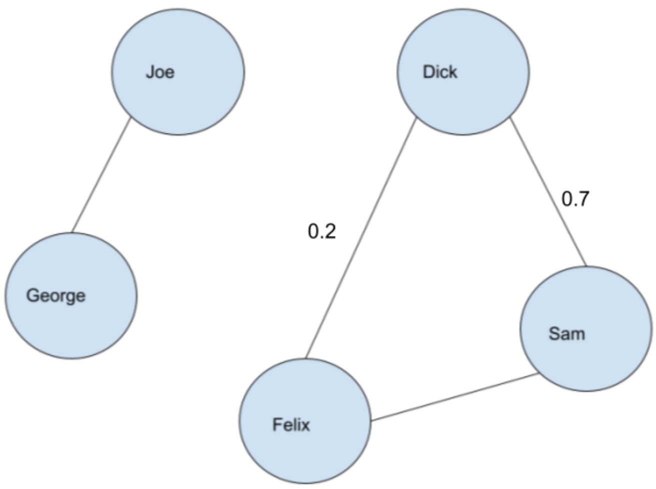

Fig 6.3. Dick-Sam edge has a weight of 0.7, whereas Dick-Felix edge has a weight of 0.2.

After a night of fun mingling, the graph may look like this. (Weights shown by boldness rather than number for visual clarity).

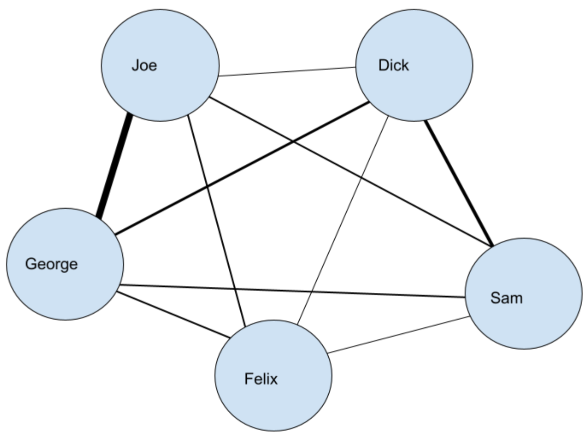

Fig 6.4. The different weights between friends (Bolder = 1, Thinner = 0).

Our neural network will be similar to this graph of 5 friends, where each pixel of the image will be given a different weight for each letter, this will allow the neural network to see what relationship a certain pixel's brightness has with different letters.

Using a graph as our neural network:

Our neural network graph will have 3 different layers, each layer will have a number of nodes connected to its neighboring layer, nodes of the same layer will not have an edge as they do not have a relation. The first layer, or the input layer will have 784 nodes, each node will correspond to a grayscale value of a pixel in the image of the letter. The output layer will include 26 nodes (Each node belongs to an English letter) where each value of the node is an abstract score given by the neural network to decide how sure it is that the image it is 'reading' corresponds to that letter. A 'hidden' layer will consist of 26 nodes between the input and output layer, it has no real concrete purpose other than acting as a chance to add more weights and increase accuracy. We are representing our neural network as a graph as it will allow us to find the relationship between a certain pixel's value and a certain English letter in the form of the value of the edges between the 2 nodes. The neural network will tune each weight until it has the correct edge values for each input and output node. (1)(3)(11)

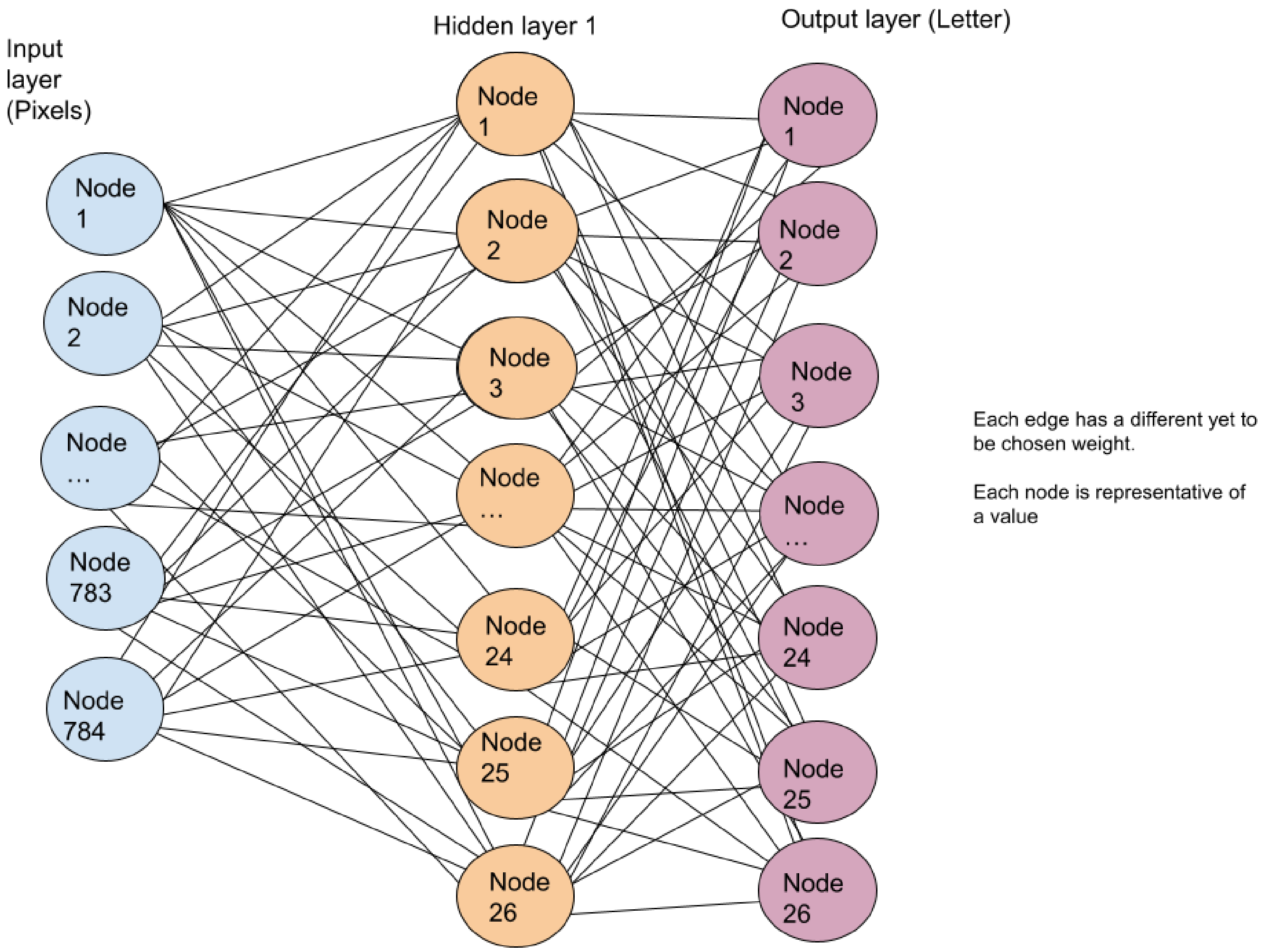

Fig 7. The neural network is represented as a graph.

For example, if the central pixel of the image were fully white, it would most likely be a ‘T’ or ‘I’ rather than an ‘O’ (As the center of ‘T’ and ‘I’ are coloured in, whereas ‘O’ isn’t). For this reason the central image node will have a higher weight with the hidden nodes that have a higher weight with ‘T’ and ‘I’ output nodes. On the other hand, the central image node will have a low weight with the hidden nodes that have a high weight with ‘O’. (1)(3)(11)

Representing the forward propagation behind the network:

The forward propagation of the network is the mathematical operations used to convert the data from the image to a chosen letter. Forward propagation can also be thought of as the computation of each of the node and edge values of the neural network. The dot product is used as it has the unique property of multiplying each input node by each individual weight and then condensing this into a vector. This can be seen as multiplying each input node by their respective weight to the first hidden node and setting the first hidden node to the sum of this value, then continuing this process until the last hidden node. (1)(3)(4)(7)(11)

We will begin by having the input $X(784, 372450)$ be our entire dataset excluding the column with the correct answer.

The first set of weights will be $W_1(26, 784)$ a matrix, containing each of the 784 input node's weight for each of the 26 hidden layer nodes. We will take the dot product between W_1 and X to start propagating the information of the image into a chosen letter. In order to provide as much accuracy as possible, we will also add a matrix $b_1(26, 1)$ to W_1X. b_1 is known as the bias and is used to give the function an additional transformation, and allow a full range of outputs, and will model in a linear fashion similar to $y = mx + c$. I have chosen to use a linear function due to its quick computation, other polynomial functions can also work. (Given the matrices can be multiplied).

Our first propagation can be written as:

$$Z_1 = W_1X + b_1$$

Where:

$X =$ Input matrix$(784,\ 372450)\{X \in \mathbb{R} | 0 \leq X \ \leq 1\}$(Grayscale values as numerical percentages)

$W_1 =$ First matrix of randomized weights$(26,\ 784)\{W_1 \in \mathbb{R} | -0.5 \leq W_1 \leq 0.5\}$

b_1 = First matrix of randomized bias $(26,\ 1)\{b_1 \in \mathbb{R}| - 0.5 \le b_1 \le 0.5\}$

Z_1 = Output matrix of first layer $(26, 372450)$ $\{Z_1 \in \mathbb{R}\}$ or the value of the hidden nodes for each of the 372450 images.

See Appendix 1. For the code of the creation of these random biases and weights.

We will then repeat this process to add an extra layer of propagation in order to increase accuracy and allow the neural network as many chances as possible to tweak values. In this case $Z_1(26, 1)$ will be our input matrix where each value corresponds to each node in the hidden layer. $W_1(26, 26)$ a matrix of weights corresponding to the edge weights between each hidden node and each of the 26 output nodes, and $b_2(26, 1)$ serving the same purpose as b_1 in the previous function. (1)(3)(4)(7)(11)

Our second propagation can be written as:

$$Z_2 = W_2 Z_1 + b_2$$

Where:

Z_1 = Input matrix$(26, 372450)$ $\{Z_1 \in \mathbb{R}\}$

W_2 = First matrix of randomized weights$(26,\ 26)\{W_2 \in \mathbb{R}| - 0.5 \le W_2 \le 0.5\}$

b_2 = First matrix of randomized bias $(26,\ 1)\{b_2 \in \mathbb{R} - 0.5 \le b_2 \le 0.5\}$

Z_2 = Output matrix of first layer $(26, 372450)$ $\{Z_2 \in \mathbb{R}\}$ or the value of each letter in the alphabet for each of the 372450 images. The greater the value of each letter the more the neural network believes the image corresponds to that letter.

See appendix 2. For these 2 functions written in python.

The matrix Z_2 is the output and the final weighing of each 26 letters foreach image in the dataset.

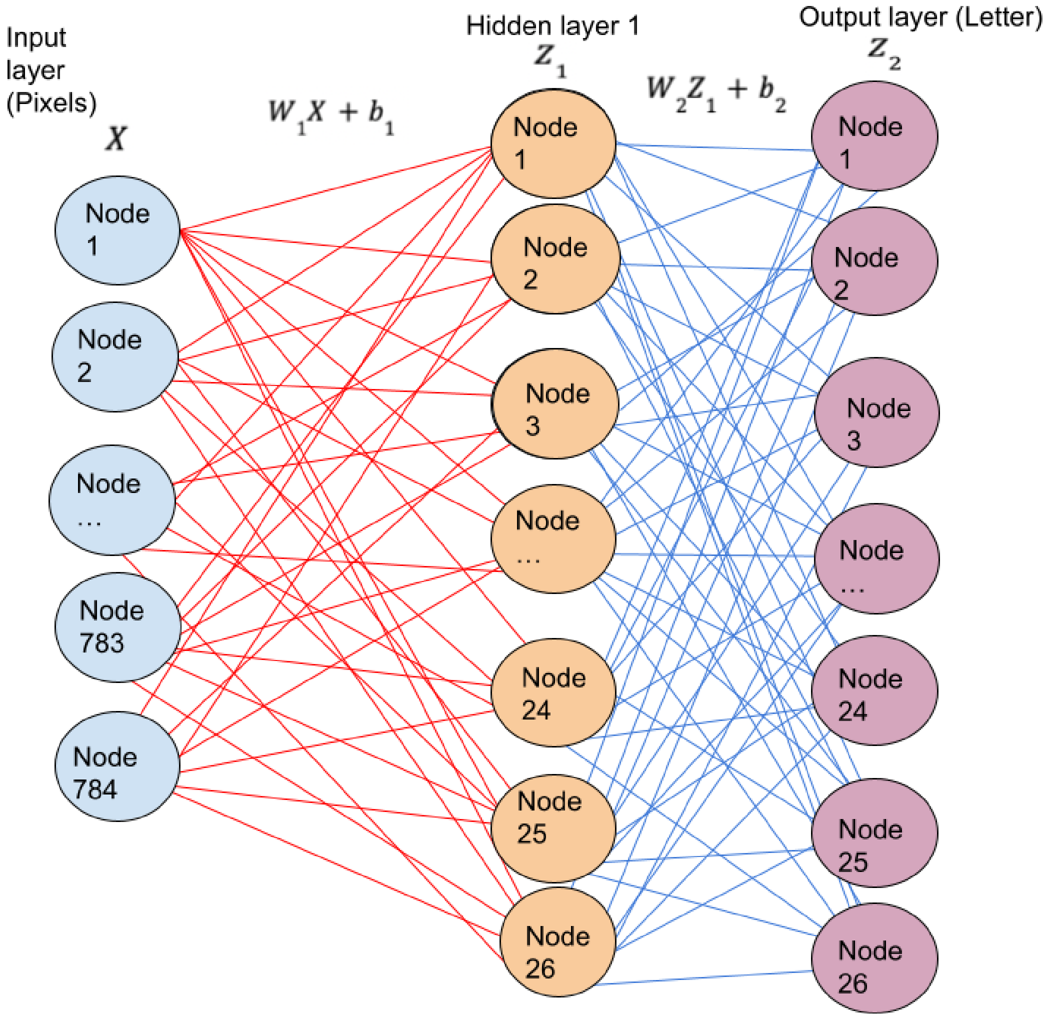

Fig 8. The red edges in our neural network indicate the first propagation, and the blue edges indicate the second propagation.

Activation function:

Keen readers may have realized that the 2 propagations can be written as one large linear function rather than 2.
$Z_2 = W_2Z_1 + b_2$

$$Z_1 = W_1 X + b_1$$

$$Z_2 = W_2(W_1 X + b_1) + b_2$$

$$Z_2 = W_2 W_1 X + (W_2 b_1 + b_2)$$

This would count as one propagation rather than 2. This propagation written in the form $y = mx + c$, will have $m =$ the constant $W_2 W_1$ and $c =$ the constant $W_2 b_1 + b_2$. In order to remove linearity between the 2 propagations and keep them separate, an extra transformation will be added. The activation function has to just be an non-linear transformation applied between Z_1 and Z_2. The activation function I have chosen is the ReLU (Rectified Linear Unit) denoted as $ReLU(x)$ due to its quick computation and simple derivative which will be helpful later on, however many other non-linear activations can be used such as the sigmoid or $tanh(x)$ (1)(3)(11). To clarify, the adding of the activation function assures the hidden layer still exists, and makes sure there can be as many weights as possible between nodes and more weights allow for more possible transformations, resulting in greater accuracy.

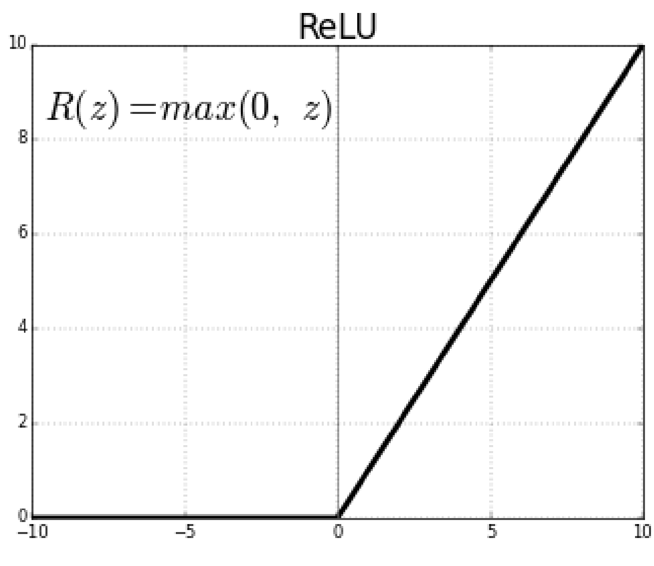

Figure 10. The rectified linear unit activation function.

See Appendix 3. For the ReLU function coded in python.

Softmax function:

The neural network will output 26 values. As the neural network can output anywhere from $-\infty$ to ∞, it might be helpful to standardize each of its 26 output values relative to the other output values in the form of a numerical percentage ranging from 0 to 1. Not only for human legibility when reading the neural network's output, but also to easily compare output values between different images. To do this we will use the softmax function. The softmax function converts each component in a vector to a percentage, relative to the sum of the vector. We can think of each of the 26 output values as components in a vector. By using the softmax function, each value for each letter will be converted to a percentage. This percentage can be thought of as how certain the neural network believes the image is of a certain letter. The usage of the softmax function (compared to simply taking the normal percentage without exponents) avoids any issues if the matrix includes a negative number as the range for the softmax function will always be $0 < y$ due to the usage of exponents. (1)(3)(11)

$$\sigma(\vec{z}) = \frac{e^{z_i}}{\Sigma e^{Z_i}}$$

Where:
σ = Softmax function
$\vec{Z}$ = Input vector
Z_i = Each value of the input vector

$$\sigma\left(\begin{bmatrix}1\\2\\3\end{bmatrix}\right) = \begin{bmatrix}0.09\\0.25\\0.66\end{bmatrix} or \begin{bmatrix}\frac{e}{e+e^2+e^3}\\\frac{e^2}{e+e^2+e^3}\\\frac{e^3}{e+e^2+e^3}\end{bmatrix}$$

Figure 9. Example of the softmax function, where the output as both numerical percentages and written algebraically.

Finally the forward propagation will look like:
$Z_1 = W_1X + b_1$ $A_1 = ReLU(Z_1)$ $Z_2 = W_2(A_1) + b_2$ $A_2 = \sigma(Z_2)$

See Appendix 4. For the softmax function coded in python.
See Appendix 5. For the entirety of the forward propagation.

Backwards propagation:

Backwards propagation is the process done to find how much we should adjust the weights and biases (originally randomly generated) and in what way in order to minimize the error of the neural network. To do this we will define a 'Cost Function', the cost function is simply representative of the error of the neural network, the value of the cost function will be the specific error of a specific guess by the neural network. (6)

In order to find the attributable error of the final matrix we need to find how accurate the final output layer (A_2) is compared to the correct answer. This can be easily done by subtracting A_2 from the correct letter of the image. However, as the correct answers are represented by the index of the letter in the alphabet, later letters such as 'Z' would be far more punishing as it would subtract 25 rather than 'A' that would subtract 0. (Indexing begins with 'A'=0 rather than 1). This challenge initially seems detrimental but can quickly be solved using one hot encoding. One hot encoding is the act of storing a vector of numbers as a binary matrix where the value is corresponding to its index rather than the number directly in the cell. (11)

$$x = \begin{bmatrix} 24 \\ 12 \\ 25 \\ 0 \end{bmatrix}$$

$$ohe(x) = \begin{bmatrix} 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Fig 10. An example of matrix $x(1, 4)$ containing indices for the letters 'Y', 'M', 'Z' and 'A' being converted to being one hot encoded(ohe). $ohe(x)$ has dimensions $26, 4$. Each column is corresponding to a letter in the alphabet.

Now that we have the issue of some letters being more punishing out of the way, we can make an arbitrary cost function $C(A)$ which represents the error of the neural network. However, this cost function is technically useless, what is important is minimizing the cost function or minimizing the error. For this reason we will simply define the derivative of this arbitrary cost function to be the neural networks output subtracted from the actual letter's one hot encoded value and not worry about the actual function itself.

$$\frac{\partial C}{\partial A_2} = A_2 - Y$$

$$\begin{array}{ccccc} A_2 & & Y & & \dfrac{dC}{dA_2} \\ \begin{bmatrix} 0.253 \\ 0.06 \\ \dots \\ 0.02 \\ 0.006 \end{bmatrix} & - & \begin{bmatrix} 1 \\ 0 \\ \dots \\ 0 \\ 0 \end{bmatrix} & = & \begin{bmatrix} -0.747 \\ 0.06 \\ \dots \\ 0.02 \\ 0.006 \end{bmatrix} \end{array}$$

$$\begin{bmatrix} 0.992 \\ 0 \\ \dots \\ 0.003 \\ 0.002 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ \dots \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.002 \\ 0 \\ \dots \\ 0.003 \\ 0.002 \end{bmatrix}$$

Fig 11. An example of $\frac{dC}{dA_2}$ being found for 2 different outputs. Note: the matrices have been reduced to be for just one image of the letter 'A' rather than the entire database for clarity. It can be seen that when the output of the neural network (A_2) is closer to the correct answer (Y), $\frac{dC}{dA_2}$ approaches 0 (the local minima of the cost function or the error of the neural network)

See appendix 6. For one hot encoding done in python.

In order to find the attributable error, we will partially differentiate C with respect to each weight and bias, W_1, W_2, b_1, b_2. This will tell us the rate of change that each of these variables has on C or in other words, which variable is causing the most error. If the derivative is negative, the weights and biases move to the right (greater) and vice versa for a positive derivative, furthermore, the change is dependent on the magnitude of each derivative on top of its direction (As a greater magnitude suggests the minimum is further away). The finding of the minimum cost function is known as gradient descent. (2)(6)(8)

In order to find the error attributable to each weight and bias we must use the chain rule, which essentially computes the rate of change caused by one variable to another, that when multiplied results in the total rate of change of the equation. (2)(6)(8)

Lets begin with partially differentiating the functions with respect to weights and biases by using the chain rule. As the softmax function doesn't change the ratio between the outputs of the neural network as it simply converts each value into a percentage we will be ignoring it as it has no impact on the error of the neural network. It will be substituted for 1.

Finding error attributable to W_2:

$$\frac{\partial C}{\partial A_2} = A_2 - Y$$

$$A_2 = \sigma(Z_2)$$

$$Z_2 = W_2(A_1) + b_2$$

$$\frac{\partial C}{\partial W_2} = \frac{\partial Z_2}{\partial W_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial C}{\partial A_2}$$

$$\frac{\partial C}{\partial W_2} = (A_1^{\ T})(1)(A_2 - Y)$$

A_1 is transposed (denoted by M^T) simply to assure a dot product can be performed.

$$\frac{\partial C}{\partial W_2} = \frac{1}{m}(A_1^{\ T})(A_2 - Y)$$

The derivative is averaged out across all 372450 (m) images in the dataset, this is so the dataset doesn't overfit to certain more common letters in the dataset, and treats each of the images in the dataset equally.

Finding error attributable to b_2:

$$\frac{\partial C}{\partial b_2} = \frac{\partial Z_2}{\partial b_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial C}{\partial A_2}$$

$$\frac{\partial C}{\partial b_2} = (1)(1)(A_2 - Y)$$

$$\frac{\partial C}{\partial b_2} = \frac{1}{m}\sum_{1}^{m}(A_2 - Y)$$

As we are averaging across the dataset all the values are added before dividing, this step won't have to be done with W_1 & W_2 as they are already added during the dot product.

Finding error attributable to Z_1 in order to find $\frac{dC}{dW_1}$ and $\frac{dC}{db_1}$.

$$A_1 = ReLU(Z_1)$$

$$Z_2 = W_2(ReLU(Z_1)) + b_2$$

$$Z_2 = W_2(A_1) + b_2$$

$$\frac{\partial C}{\partial Z_1} = \frac{\partial Z_2}{\partial ReLU(Z_1)}\frac{\partial C}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial ReLU(Z_1)}{\partial Z_1} \ or \ \frac{\partial Z_2}{\partial A_1}\frac{\partial C}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial A_1}{\partial Z_1}$$

$$\frac{\partial C}{\partial Z_1} = (W_2^{\ T})(A_2 - Y)(1) \bullet ReLU'(Z_1)$$

$$\frac{\partial C}{\partial Z_1} = (W_2^{\ T})(A_2 - Y) \bullet ReLU'(Z_1)$$

Finding error attributable to W_1:

$$Z_1 = W_1X + b_1$$

$$\frac{\partial C}{\partial W_1} = \frac{\partial Z_1}{\partial W_1}\frac{\partial C}{\partial Z_1}$$

$$\frac{\partial C}{\partial W_1} = \frac{1}{m}(X^T)(W_2^{\ T})(A_2 - Y) \bullet ReLU'(Z_1)$$

Finding error attributable to b_1:

$$Z_1 = W_1X + b_1$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial Z_1}{\partial b_1}\frac{\partial C}{\partial Z_1}$$

$$\frac{\partial C}{\partial W_1} = (1)({W_2}^T)(A_2 - Y) \bullet ReLU'(Z_1)$$

$$\frac{\partial C}{\partial b_1} = \frac{1}{m}\sum_{1}^{m}({W_2}^T)(A_2 - Y) \bullet ReLU'(Z_1)$$

Finally the entire backward propagation will look like:

$$\frac{\partial C}{\partial W_2} = \frac{1}{m}({A_1}^T)(A_2 - Y)$$

$$\frac{\partial C}{\partial b_2} = \frac{1}{m}\sum_{1}^{m}(A_2 - Y)$$

$$\frac{\partial C}{\partial Z_1} = ({W_2}^T)(A_2 - Y) \bullet ReLU'(Z_1)$$

$$\frac{\partial C}{\partial W_1} = \frac{1}{m}(X^T)({W_2}^T)(A_2 - Y) \bullet ReLU'(Z_1)$$

$$\frac{\partial C}{\partial b_1} = \frac{1}{m}\sum_{1}^{m}({W_2}^T)(A_2 - Y) \bullet ReLU'(Z_1)$$

See appendix 7, for derivative of $ReLU(x)$ written in python.

See appendix 8. Backwards propagation written in python.

A summed up version of all this math can be thought of if we consider the cost function and forward propagation to be a simple function with 784 inputs (each of the pixels of the image). $f(x_1, x_2, x_3 ... x_{784}, x_{784})$. This function is equal to the amount of error of the forward propagation. If we wish to minimize the error of the forward propagation we just equate the derivative of $f'(x)$ to 0. As $f(x)$ is a 785 dimension function; it is simply too complex to calculate its derivative. Instead we take the partial derivatives of $f(x)$ or the rate that $f(x)$ changes with respect to different values within $f(x)$ and follow these derivatives to the nearest local minima. (As the derivative of a function can act as a guide to the nearest local minima). This is why it is known as gradient descent. We are finding the gradient and 'descending' it to find the local minima.

Updating parameters:

Once we know the error amount for each weight and bias, we can slowly tune them, the learning rate is the rate at which the weights and biases are changed and is usually set to either 10^{0}, 10^{-1}, 10^{-2} and is represented with α. The greater the learning rate the faster the neural network will take to increase its accuracy at the start but will take longer to 'perfect' its weights. Whereas the opposite applies for the smaller learning rates. I have chosen the Goldilocks between the 3 and will use 10^{-1} as my learning rate as it allows for relatively fast learning throughout the tuning of the weights. (11)

$$W_1 := W_1 - \alpha \frac{\partial C}{\partial W_2}$$

$$b_1 := b_1 - \alpha \frac{\partial C}{\partial b_1}$$

$$W_2 := W_2 - \alpha \frac{\partial C}{\partial W_2}$$

$$b_2 := b_2 - \alpha \frac{\partial C}{\partial b_2}$$

See appendix 9. For updating parameters in python.

Avoiding overfitting to the dataset:

When iterating several times over the same dataset the neural network may adjust its weights to be perfect for the dataset but suffer when given an image outside of the data set. In order to not overfit the weights to the dataset several measures can be used, such as a greater dataset, going through different parts of the dataset then updating parameters rather than only once the entire dataset has been analyzed (Known as scholastic gradient descent (8)) and lastly decreasing iterations. Decreasing iterations is the easiest option but does sacrifice accuracy and a middle ground has to be reached. I have chosen to use iterations between

100-1000 (37,245,000-372,450,000 letters analyzed) in order to achieve a range of accuracy and 'overifttedness'. (11)

Conclusion

Does it work?:

I have chosen to use 3 iterations that will be trained 3 different times: 100, 500 and 1000. Once trained I will write each letter of the alphabet once with my own handwriting and see whether the neural network can recognise my handwriting.

It does!:

Number of iterations	100			500			1000		
Trial:	1	2	3	1	2	3	1	2	3
Accuracy tested against dataset	45.05%	48.56%	54.31%	76.09%	75.46%	75.68%	80.29%	80.81%	81.41%
A	[(array([[0.31742218]]), 'M'), (array([[0.19339505]]), 'T'), (array([[0.1176487]]), 'R')]	[(array([[0.23924301]]), 'P'), (array([[0.09715055]]), 'T'), (array([[0.08949161]]), 'A')]	[(array([[0.56053275]]), 'R'), (array([[0.10625979]]), 'N'), (array([[0.08960661]]), 'A')]	[(array([[0.54765309]]), 'R'), (array([[0.36361177]]), 'K'), (array([[0.02543991]]), 'F')]	[(array([[0.40832837]]), 'A'), (array([[0.26663318]]), 'R'), (array([[0.12260464]]), 'Z')]	[(array([[0.47457226]]), 'R'), (array([[0.1801121]]), 'F'), (array([[0.10570351]]), 'K')]	[(array([[0.523526]]), 'R'), (array([[0.21973785]]), 'A'), (array([[0.11254757]]), 'E')]	[(array([[0.82776862]]), 'R'), (array([[0.12435686]]), 'K'), (array([[0.03306185]]), 'N')]	[(array([[0.523526]]), 'P'), (array([[0.21973785]]), 'A'), (array([[0.11254757]]), 'E')]
B	[(array([[0.16705517]]), 'M'), (array([[0.16365269]]), 'J'), (array([[0.16077417]]), 'T')]	[(array([[0.27143536]]), 'T'), (array([[0.13756267]]), 'W'), (array([[0.09146924]]), 'S')]	[(array([[0.33767162]]), 'C'), (array([[0.15932703]]), 'E'), (array([[0.12282269]]), 'L')]	[(array([[0.25373518]]), 'A'), (array([[0.23350284]]), 'B'), (array([[0.10871818]]), 'E')]	[(array([[0.45871551]]), 'E'), (array([[0.06701903]]), 'N'), (array([[0.06574791]]), 'S')]	[(array([[0.78531813]]), 'K'), (array([[0.11438045]]), 'E'), (array([[0.03964891]]), 'F')]	[(array([[0.70394109]]), 'E'), (array([[0.11359101]]), 'S'), (array([[0.06915111]]), 'Z')]	[(array([[0.52279352]]), 'B'), (array([[0.18424885]]), 'E'), (array([[0.13473017]]), 'D')]	[(array([[0.3886695]]), 'E'), (array([[0.32931688]]), 'Z'), (array([[0.13484859]]), 'R')]
Average accuracy of first guess when reading my handwriting	9.19%			33.33%			53.85%		
Average accuracy of top 2 guesses when reading my handwriting	20.51%			47.44%			74.36%		
Average accuracy of top 3 guesses when reading my handwriting	34.62%			58.97%			80.77%		

Fig 12. A preview of the neural networks outputs across all 3 trials and all 3 iterations for the letters 'A' and 'B'.

If the square is green the neural network's first guess was correct, if it's yellow the second guess was correct, if it's red the third guess is correct. If there is no color the neural network's guess was wrong. Each guessed letter also has the guessed probability stated before it but this was not considered during testing.

See Appendix 10. For full data table.

The neural network can read my handwriting! (For the most part). This means that **graph theory, gradient descent and partial derivatives can be used to develop a letter recognising neural network.** The neural network had some success with guessing what I had written in its first attempt, getting an average of 53.85% correct after 1000 iterations, 33.33% after 500 iterations and only 9.19% after 100 iterations. It is clear that 100 iterations is nowhere near enough, and 500 iterations also falls a bit flat. This implies that for my neural network to be properly applied in a real world context it would need significantly more iterations to train from. This of course would take up a lot of computational time.

The neural network succeeds most when the top 3 results are considered rather than only the first. When using this metric, the 100 iteration trials had an average of 34.62% compared to 500's 58.97% and 1000's 80.77%.

The least recognised letters were 'I' and 'F" which each got recognised 0 times within the first guess. Whereas 'C' and 'S' were recognised 6 and 5 times out of the 9 trials respectively. This correlates directly to the quantity of images of each letter in the dataset. The lack of data is responsible for the neural networks' decreased understanding.

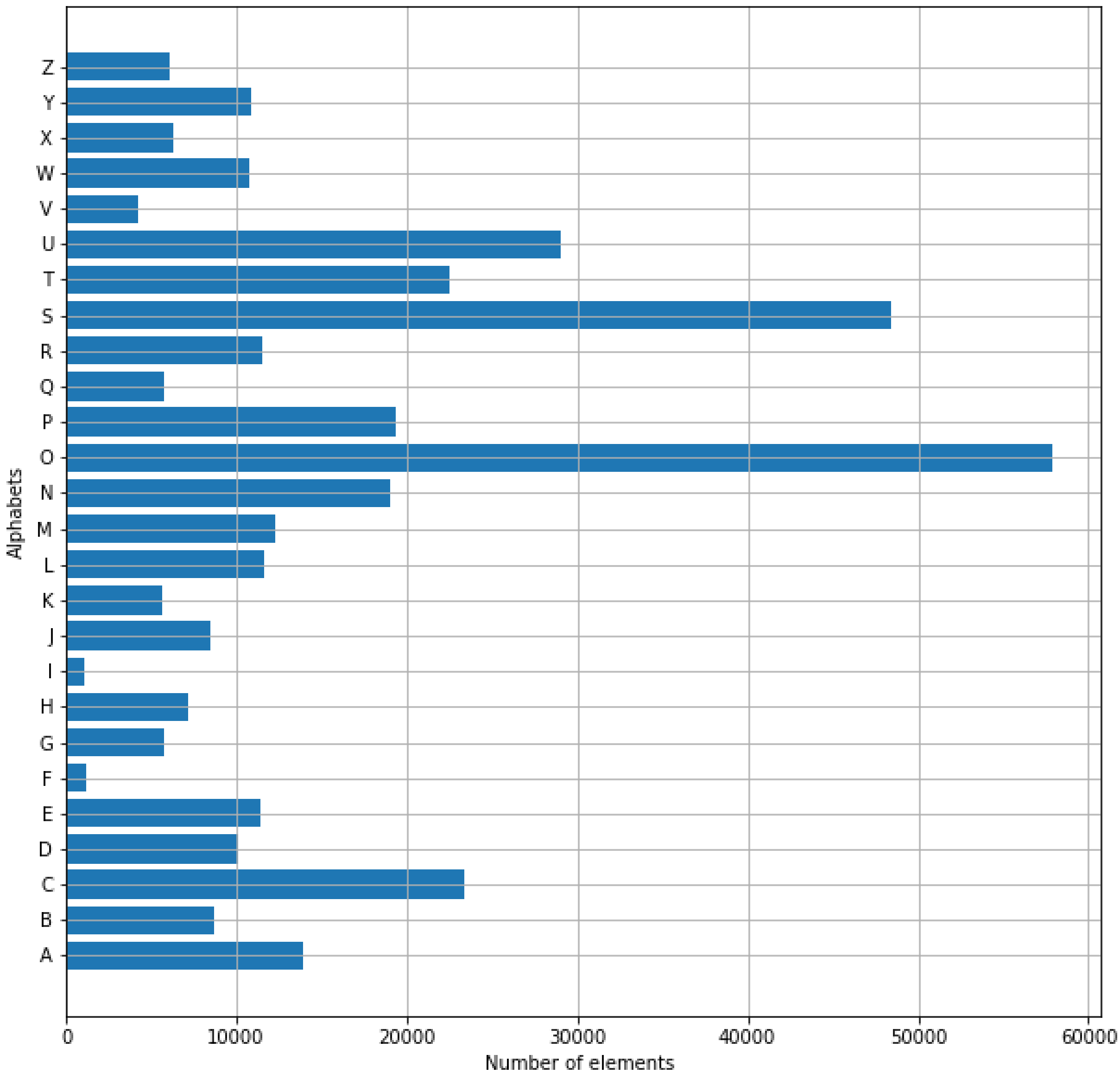

Fig 12, a graph displaying the frequency of each letter in the 372450 letter dataset (5).

With this data we can also subtract the accuracy of the neural network when compared to the data set from the accuracy of recognising my handwriting to find how overfitted each of the iterations are. Interestingly, iterations 100 and 500 feature similar average 'overfitting ratings' of around 40% whereas the 1000 iteration trials had an average of only 26%, this may suggest that there are multiple 'waves' of overfitting, and they appear before 100 and 500 iterations and after 1000.

Limitations:

The dataset chosen provides grayscale 28 by 28 pixel images of capital english letters that have been centered to a 20x20 central square within the image. This raises many limitations for the neural network, firstly, as every image is shrunk to a 28x28 picture many details may be lost when using real pictures, these details can make the difference between an ‘O’ and a ‘Q’ or between a ‘D’ and a ‘B’ differences that seem clear but may dissipate when scaled down. Secondly, while grayscale does allow the neural network to work faster and process several color letters, what happens if a letter is written in black on a white background (The opposite of what the neural network is used to). The network can also only identify english letters written a certain way, as the dataset includes only capital letters the neural network struggles to identify lower case or alternatively written letters. Additionally in my testing each letter would be written differently, this means if for one trial my hand might have slipped it may have misrepresented the neural network for that specific trial and iteration, instead I should use the same 26 images of letters written by me across all trials. Lastly, as the neural network is used to the letters being centralized for them within the 20x20 square of the image, the neural network struggles to identify clear letters that are not centralized.

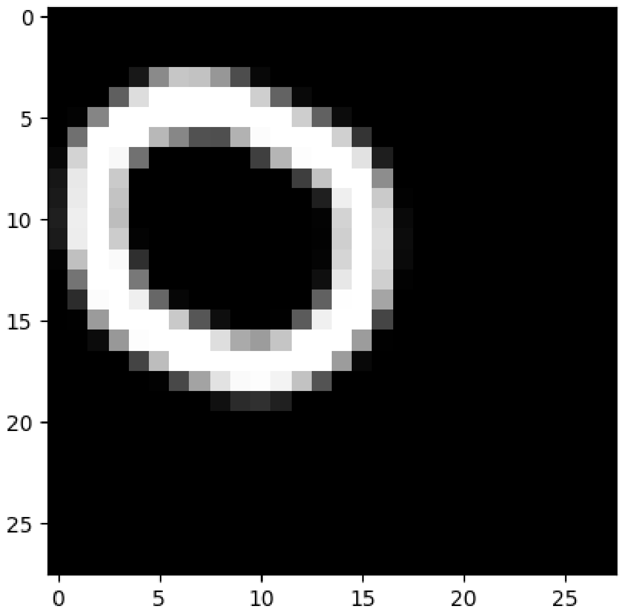

Fig 18. A non-centralised ‘O’ being recognised as a ‘T’, ‘P’ and ‘F' by the neural network.

Potential improvements:

The neural network isn't perfect and still has many flaws and limitations. However there are several potential improvements that can be done to improve the accuracy.

Adding another hidden layer:

Adding another layer of weights to our neural network will allow it to gain an extra layer of complexity and can make the difference between differentiating similar letters that couldn't be distinguished with only 2 matrices. To assure non-linearity is kept, another activation function will need to be added between layer 1 and 2. Adding a layer will also increase computational time.

Changing the activation function:

Different activation functions can be more useful for different tasks. While it is difficult to predict which activation function will be the best for which task, there is no pain in testing and seeing the results. Other functions include the sigmoid function and tanh.

Too small image size:

In order to keep smaller potentially crucial details in letters, a greater image size can be used, however this will result in the need to train a new neural network with a different number of input nodes. The greater number of input nodes will result in more dot product multiplication increasing computation time. Furthermore, a new dataset will need to be used, one that produces data to the wanted image size.

Adding more hidden nodes:

Adding more hidden nodes will increase the number of weights and biases that the neural network can adjust. This will give it more liberty and allow it to minimize error even further. Adding hidden nodes will also increase computational time and should be done in moderation.

Other improvements can include adding a centralizing algorithm to automatically center a letter or adding an algorithm to automatically change every image to be white letter on black background, so the network can become more universal.

Works cited:

1. Cornelisse, Daphne Cornelisse. “How to Build a Three-Layer Neural Network from Scratch.” *FreeCodeCamp.org*, FreeCodeCamp.org, 13 Feb. 2018, https://www.freecodecamp.org/news/building-a-3-layer-neural-network-from-scratch-99239c4af5d3/.
2. Kostadinov, Simeon. “Understanding Backpropagation Algorithm.” *Medium*, Towards Data Science, 12 Aug. 2019, https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.
3. Mesquita, Déborah. “Python Ai: How to Build A Neural Network & Make Predictions.” *Real Python*, Real Python, 27 Feb. 2021, https://realpython.com/python-ai-neural-network/.
4. “NumPy Documentation.” NumPy Developers, Apr. 2022.
5. Patel, Sachin. *A-Z Handwritten Alphabets in .Csv Format*, Kaggle, 2018, https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format. Accessed 13 Aug. 2022.
6. Sanderson, Grant, director. *Backpropagation Calculus | Chapter 4, Deep Learning*. *YouTube*, YouTube, 3 Nov. 2017, https://www.youtube.com/watch?v=tIeHLnjs5U8. Accessed 13 Aug. 2022.
7. Sanderson, Grant. “The Determinant | Chapter 6, Essence of Linear Algebra.” *YouTube*, YouTube, 10 Aug. 2016, https://www.youtube.com/watch?v=Ip3X9LOh2dk.

8. Socher, Richard, director. *Lecture 5: Backpropagation and Project Advice*. *YouTube*, Stanford University School of Engineering, 3 Apr. 2017, https://www.youtube.com/watch?v=isPiE-DBagM&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6. Accessed 14 Aug. 2022.

9. Sutton, Jeffrey L. "Spring Sunset in the Blue Ridge Mountain in Western North Carolina." *Reddit.com*, 4 Sept. 2017, https://www.reddit.com/r/EarthPorn/comments/6i7kyk/spring_sunset_in_the_blue_ridge_mountain_in/?utm_source=ifttt. Accessed 23 Oct. 2022.

10. Wilson, Robin J. *Introduction to Graph Theory*. Prentice Hall, 1995.

11. Zhang, Samson, director. *Building a Neural Network FROM SCRATCH (No Tensorflow/Pytorch, Just Numpy & Math)*. *YouTube*, YouTube, 24 Nov. 2020, https://www.youtube.com/watch?v=w8yWXqWQYmU. Accessed 13 Aug. 2022.

Appendix:

1. Initial creation of these random biases and weights:
   a.
   ```
   W1 = np.random.rand(26, 784) - 0.5
   b1 = np.random.rand(26, 1) - 0.5
   W2 = np.random.rand(26, 26) - 0.5
   b2 = np.random.rand(26, 1) - 0.5
   ```
2. Z_1 and Z_2 written in python.
   a.
   ```
   Z1 = W1.dot(X) + b1
   Z2 = W2.dot(Z1) + b2
   ```
3. ReLU Function
   a.
   ```
   def ReLU(Z):
       return np.maximum(Z, 0)
   ```
4. Softmax function

a.
```
def softmax(Z):
    A = np.exp(Z) / sum(np.exp(Z))
    return A
```

5. Forward propagation

a.
```
Z1 = W1.dot(X) + b1
A1 = ReLU(Z1)
Z2 = W2.dot(A1) + b2
A2 = softmax(Z2)
```

6. One hot encoding function

a.
```
def one_hot_encode(Y):
    one_hot_encoded_Y = np.zeros((Y.size, Y.max() + 1))
    one_hot_encoded_Y[np.arange(Y.size), Y] = 1
    one_hot_encoded_Y = one_hot_encoded_Y.T
    return one_hot_encoded_Y
```

7. Derivative of $ReLU(x)$:

a.
```
def ReLU_derivative(Z):
    return Z > 0
```

8. Backwards propagation:

a.
```
one_hot_Y = one_hot_encode(Y)
dZ2 = A2 - one_hot_Y
dW2 = 1/m * dZ2.dot(A1.T)
db2 = 1 / m * np.sum(dZ2)
dZ1 = W2.T.dot(dZ2) * ReLU_derivative(Z1)
dW1 = 1 / m * dZ1.dot(X.T)
db1 = 1 / m * np.sum(dZ1)
```

9. Updating parameters

a.
```
W1 = W1 - alpha * dW1
b1 = b1 - alpha * db1
W2 = W2 - alpha * dW2
b2 = b2 - alpha * db2
```

10. Testing results/data

	100			500			1000		
Trial:	1	2	3	1	2	3	1	2	3
Accuracy tested against dataset	45.05%	48.56%	54.31%	76.09%	75.46%	75.68%	80.29%	80.81%	81.41%
Time taken to train (minutes)	1.35	1.366666667	1.483333333	6.866666667	6.716666667	6.8	13.55	14.06666667	13.11666667
A	[(array([[0.31742218]]), 'M'), (array([[0.19339505]]), 'T'), (array([[0.1176487]]), 'R')]	[(array([[0.23924301]]), 'P'), (array([[0.09715055]]), 'T'), (array([[0.08949161]]), 'A')]	[(array([[0.56053275]]), 'R'), (array([[0.10625979]]), 'N'), (array([[0.08960661]]), 'A')]	[(array([[0.54765309]]), 'R'), (array([[0.36361177]]), 'K'), (array([[0.02543991]]), 'F')]	[(array([[0.40832837]]), 'A'), (array([[0.26663318]]), 'R'), (array([[0.12260464]]), 'Z')]	[(array([[0.47457226]]), 'R'), (array([[0.1801121]]), 'F'), (array([[0.10570351]]), 'K')]	[(array([[0.523526]]), 'R'), (array([[0.21973785]]), 'A'), (array([[0.11254757]]), 'E')]	[(array([[0.82776862]]), 'R'), (array([[0.12435686]]), 'K'), (array([[0.03306185]]), 'N')]	[(array([[0.523526]]), 'P'), (array([[0.21973785]]), 'A'), (array([[0.11254757]]), 'E')]
B	[(array([[0.16705517]]), 'M'), (array([[0.16365269]]), 'J'), (array([[0.16077417]]), 'T')]	[(array([[0.27143536]]), 'T'), (array([[0.13756267]]), 'W'), (array([[0.09146924]]), 'S')]	[(array([[0.33767162]]), 'C'), (array([[0.15932703]]), 'E'), (array([[0.12282269]]), 'L')]	[(array([[0.25373518]]), 'A'), (array([[0.23350284]]), 'B'), (array([[0.10871818]]), 'E')]	[(array([[0.45871551]]), 'E'), (array([[0.06701903]]), 'N'), (array([[0.06574791]]), 'S')]	[(array([[0.78531813]]), 'K'), (array([[0.11438045]]), 'E'), (array([[0.03964891]]), 'F')]	[(array([[0.70394109]]), 'E'), (array([[0.11359101]]), 'S'), (array([[0.06915111]]), 'Z')]	[(array([[0.52279352]]), 'B'), (array([[0.18424885]]), 'E'), (array([[0.13473017]]), 'D')]	[(array([[0.3886695]]), 'E'), (array([[0.32931688]]), 'Z'), (array([[0.13484859]]), 'R')]
C	[(array([[0.18273361]]), 'D'), (array([[0.09321293]]), 'Z'), (array([[0.0876644]]), 'S')]	[(array([[0.2903036]]), 'S'), (array([[0.08890185]]), 'B'), (array([[0.07502079]]), 'C')]	[(array([[0.39709027]]), 'C'), (array([[0.22046499]]), 'L'), (array([[0.1436615]]), 'G')]	[(array([[0.9493487]]), 'C'), (array([[0.01465304]]), 'G'), (array([[0.0138391]]), 'E')]	[(array([[0.58026678]]), 'L'), (array([[0.28104793]]), 'Z'), (array([[0.04945776]]), 'J')]	[(array([[0.99156655]]), 'C'), (array([[0.00565181]]), 'E'), (array([[0.00098138]]), 'J')]	[(array([[0.88849628]]), 'C'), (array([[0.05542512]]), 'E'), (array([[0.01974642]]), 'R')]	[(array([[0.96518937]]), 'C'), (array([[0.01149096]]), 'O'), (array([[0.00953962]]), 'J')]	[(array([[0.89475375]]), 'C'), (array([[0.0651964]]), 'E'), (array([[0.01102572]]), 'R')]
D	[(array([[0.3751112]]), 'O'), (array([[0.25463229]]), 'U'), (array([[0.11869346]]), 'D')]	[(array([[0.63744893]]), 'D'), (array([[0.13464416]]), 'V'), (array([[0.06631764]]), 'O')]	[(array([[0.16250735]]), 'S'), (array([[0.15732894]]), 'B'), (array([[0.15167319]]), 'Q')]	[(array([[0.5129512]]), 'D'), (array([[0.37147735]]), 'O'), (array([[0.05974166]]), 'Q')]	[(array([[0.62142589]]), 'U'), (array([[0.26209454]]), 'D'), (array([[0.05864251]]), 'O')]	[(array([[0.51777581]]), 'O'), (array([[0.25733417]]), 'U'), (array([[0.1242636]]), 'D')]	[(array([[0.77643009]]), 'D'), (array([[0.16230296]]), 'O'), (array([[0.02364741]]), 'Q')]	[(array([[0.71486437]]), 'D'), (array([[0.20737649]]), 'O'), (array([[0.03783788]]), 'B')]	[(array([[0.60865098]]), 'D'), (array([[0.23542376]]), 'O'), (array([[0.07203405]]), 'U')]
E	[(array([[0.16457691]]), 'D'), (array([[0.10798804]]), 'Z'), (array([[0.08403955]]), 'E')]	[(array([[0.23696965]]), 'S'), (array([[0.12767179]]), 'W'), (array([[0.0762674]]), 'Y')]	[(array([[0.44991005]]), 'C'), (array([[0.15957992]]), 'E'), (array([[0.09321765]]), 'O')]	[(array([[0.53859622]]), 'E'), (array([[0.35560268]]), 'L'), (array([[0.07486855]]), 'C')]	[(array([[0.23314956]]), 'E'), (array([[0.22150726]]), 'B'), (array([[0.18286349]]), 'R')]	[(array([[0.51295146]]), 'P'), (array([[0.3268541]]), 'T'), (array([[0.1014166]]), 'R')]	[(array([[0.34596983]]), 'L'), (array([[0.26247181]]), 'E'), (array([[0.18913398]]), 'K')]	[(array([[0.36996614]]), 'P'), (array([[0.14247084]]), 'R'), (array([[0.08082039]]), 'K')]	[(array([[0.67404066]]), 'E'), (array([[0.26073035]]), 'B'), (array([[0.01946532]]), 'P')]
F	[(array([[0.68467703]]), 'P'), (array([[0.094361]]), 'A'), (array([[0.0841904]]), 'E')]	[(array([[0.41632357]]), 'T'), (array([[0.10620321]]), 'P'), (array([[0.07146522]]), 'Y')]	[(array([[0.62654596]]), 'P'), (array([[0.15153498]]), 'T'), (array([[0.07515063]]), 'E')]	[(array([[0.62523186]]), 'P'), (array([[0.28428018]]), 'T'), (array([[0.06312221]]), 'R')]	[(array([[0.57456309]]), 'T'), (array([[0.28697463]]), 'P'), (array([[0.04625429]]), 'F')]	[(array([[0.41727139]]), 'E'), (array([[0.27246963]]), 'R'), (array([[0.20041238]]), 'F')]	array([[0.53203877]]), 'R'), (array([[0.21782765]]), 'F'), (array([[0.07762281]]), 'E')]	[(array([[0.58528917]]), 'P'), (array([[0.2607291]]), 'R'), (array([[0.07066792]]), 'F')]	[(array([[0.76970466]]), 'P'), (array([[0.06391688]]), 'E'), (array([[0.05213051]]), 'Y')]
G	[(array([[0.35755134]]), 'S'), (array([[0.12105183]]), 'O'), (array([[0.08247253]]), 'B')]	[(array([[0.1898283]]), 'U'), (array([[0.11682976]]), 'C'), (array([[0.07764447]]), 'Q')]	[(array([[0.37677825]]), 'S'), (array([[0.13782837]]), 'C'), (array([[0.09142178]]), 'G')]	[(array([[0.2885877]]), 'N'), (array([[0.21316494]]), 'M'), (array([[0.11554839]]), 'W')]	[(array([[0.63624566]]), 'S'), (array([[0.15084622]]), 'U'), (array([[0.08921201]]), 'G')]	[(array([[0.51551922]]), 'G'), (array([[0.40331797]]), 'S'), (array([[0.02255422]]), 'E')]	[(array([[0.53980727]]), 'G'), (array([[0.44816489]]), 'S'), (array([[0.00560835]]), 'Q')]	[(array([[0.55231839]]), 'G'), (array([[0.1892528]]), 'U'), (array([[0.14019909]]), 'Q')]	[(array([[0.46057615]]), 'C'), (array([[0.21370399]]), 'S'), (array([[0.11205175]]), 'G')]
H	[(array([[0.19926396]]), 'L'), (array([[0.14960627]]), 'F'), (array([[0.11751083]]), 'M')]	[(array([[0.24960486]]), 'U'), (array([[0.09241769]]), 'B'), (array([[0.08815236]]), 'S')]	[(array([[0.45837292]]), 'U'), (array([[0.30930686]]), 'N'), (array([[0.07231526]]), 'A')]	[(array([[0.30518188]]), 'N'), (array([[0.27832293]]), 'H'), (array([[0.13401562]]), 'E')]	[(array([[0.62406195]]), 'Y'), (array([[0.12185079]]), 'X'), (array([[0.10216659]]), 'T')]	[(array([[0.31279785]]), 'V'), (array([[0.28027701]]), 'R'), (array([[0.08144919]]), 'U')]	[(array([[0.56716934]]), 'H'), (array([[0.22426828]]), 'A'), (array([[0.14998542]]), 'N')]	[(array([[0.40393601]]), 'Y'), (array([[0.33256225]]), 'X'), (array([[0.16993852]]), 'R')]	[(array([[0.47357902]]), 'H'), (array([[0.3351869]]), 'M'), (array([[0.10665452]]), 'B')]
I	[(array([[0.23220774]]), 'T'), (array([[0.13520816]]), 'J'), (array([[0.12981734]]), 'D')]	[(array([[0.18888608]]), 'S'), (array([[0.08350693]]), 'D'), (array([[0.07118386]]), 'R')]	[(array([[0.25821869]]), 'E'), (array([[0.20453003]]), 'P'), (array([[0.13063372]]), 'T')]	[(array([[0.15716249]]), 'E'), (array([[0.15290641]]), 'B'), (array([[0.1325947]]), 'D')]	[(array([[0.3962686]]), 'T'), (array([[0.1840918]]), 'Y'), (array([[0.12604861]]), 'Z')]	[(array([[0.71732806]]), 'D'), (array([[0.15559551]]), 'I'), (array([[0.04400126]]), 'B')]	[(array([[0.37025253]]), 'S'), (array([[0.18427187]]), 'I'), (array([[0.13286502]]), 'J')]	[(array([[0.48088602]]), 'T'), (array([[0.20682582]]), 'I'), (array([[0.072275]]), 'E')]	[(array([[0.34905226]]), 'S'), (array([[0.31288702]]), 'T'), (array([[0.26151678]]), 'J')]

J	[(array([[0.26820761]]), 'P'), (array([[0.13970898]]), 'Z'), (array([[0.10346897]]), 'S')]	[(array([[0.28290943]]), 'S'), (array([[0.12886785]]), 'J'), (array([[0.08544854]]), 'B')]	[(array([[0.2476936]]), 'E'), (array([[0.15489077]]), 'C'), (array([[0.15326895]]), 'T')]	[(array([[0.76661407]]), 'T'), (array([[0.22684737]]), 'P'), (array([[0.00412821]]), 'R')]	[(array([[0.97850854]]), 'T'), (array([[0.00885968]]), 'P'), (array([[0.00712779]]), 'Y')]	[(array([[0.84740765]]), 'T'), (array([[0.06074884]]), 'P'), (array([[0.01295237]]), 'Q')]	[(array([[0.93850671]]), 'S'), (array([[0.04706996]]), 'J'), (array([[0.00333332]]), 'G')]	[(array([[0.37333331]]), 'T'), (array([[0.31141594]]), 'J'), (array([[0.1389471]]), 'C')]	[(array([[0.69535829]]), 'J'), (array([[0.23603332]]), 'S'), (array([[0.01597164]]), 'O')]
K	[(array([[0.19364099]]), 'R'), (array([[0.18991248]]), 'T'), (array([[0.08094462]]), 'N')]	[(array([[0.42456516]]), 'T'), (array([[0.21437703]]), 'C'), (array([[0.0817546]]), 'E')]	[(array([[0.5479129]]), 'C'), (array([[0.11007016]]), 'H'), (array([[0.10837933]]), 'L')]	[(array([[0.222428]]), 'K'), (array([[0.19656767]]), 'R'), (array([[0.1444724]]), 'C')]	[(array([[0.72978242]]), 'C'), (array([[0.07461774]]), 'T'), (array([[0.04082267]]), 'N')]	[(array([[0.28113457]]), 'F'), (array([[0.25304091]]), 'K'), (array([[0.17592407]]), 'R')]	[(array([[0.28980313]]), 'R'), (array([[0.22192388]]), 'E'), (array([[0.21342137]]), 'K')]	[(array([[0.73977537]]), 'E'), (array([[0.10058162]]), 'K'), (array([[0.10043079]]), 'R')]	[(array([[0.51475254]]), 'E'), (array([[0.22183073]]), 'K'), (array([[0.12251519]]), 'R')]
L	[(array([[0.48215918]]), 'O'), (array([[0.09147128]]), 'U'), (array([[0.07235455]]), 'Z')]	[(array([[0.11981601]]), 'C'), (array([[0.11684341]]), 'V'), (array([[0.06624987]]), 'G')]	[(array([[0.40730742]]), 'L'), (array([[0.14449345]]), 'G'), (array([[0.10908025]]), 'C')]	[(array([[0.2177385]]), 'L'), (array([[0.17016629]]), 'E'), (array([[0.16529096]]), 'K')]	[(array([[0.29959074]]), 'K'), (array([[0.21858926]]), 'V'), (array([[0.13324593]]), 'Z')]	[(array([[0.52939934]]), 'E'), (array([[0.29438483]]), 'C'), (array([[0.08834715]]), 'Z')]	[(array([[0.98326652]]), 'L'), (array([[0.0093178]]), 'U'), (array([[0.00264768]]), 'K')]	[(array([[0.47294601]]), 'L'), (array([[0.31571594]]), 'C'), (array([[0.16174568]]), 'D')]	[(array([[0.92829579]]), 'C'), (array([[0.02989508]]), 'O'), (array([[0.02749703]]), 'E')]
M	[(array([[0.13682682]]), 'R'), (array([[0.12070182]]), 'M'), (array([[0.08865266]]), 'U')]	[(array([[0.15727752]]), 'O'), (array([[0.14972735]]), 'A'), (array([[0.11739098]]), 'B')]	[(array([[0.46134433]]), 'D'), (array([[0.09273067]]), 'M'), (array([[0.07953082]]), 'E')]	[(array([[0.65758286]]), 'H'), (array([[0.15913456]]), 'K'), (array([[0.04248033]]), 'A')]	[(array([[0.50791359]]), 'R'), (array([[0.1862545]]), 'M'), (array([[0.07018608]]), 'A')]	[(array([[0.50541107]]), 'M'), (array([[0.2860363]]), 'C'), (array([[0.05408807]]), 'U')]	[(array([[0.44903138]]), 'Q'), (array([[0.13118601]]), 'S'), (array([[0.12005343]]), 'Y')]	[(array([[0.99586271]]), 'M'), (array([[0.00145136]]), 'H'), (array([[0.00093842]]), 'N')]	[(array([[0.46355961]]), 'A'), (array([[0.23224125]]), 'M'), (array([[0.11057115]]), 'P')]
N	[(array([[0.36641113]]), 'O'), (array([[0.20018011]]), 'U'), (array([[0.05226518]]), 'B')]	[(array([[0.47283305]]), 'U'), (array([[0.15668521]]), 'H'), (array([[0.12466162]]), 'V')]	[(array([[0.59764105]]), 'W'), (array([[0.11034388]]), 'U'), (array([[0.06639637]]), 'N')]	[(array([[0.35612714]]), 'N'), (array([[0.24413553]]), 'K'), (array([[0.20051225]]), 'H')]	[(array([[0.60168177]]), 'N'), (array([[0.26505742]]), 'R'), (array([[0.03339391]]), 'Q')]	[(array([[0.63105898]]), 'K'), (array([[0.15632094]]), 'R'), (array([[0.05239216]]), 'E')]	[(array([[0.37406951]]), 'N'), (array([[0.25289893]]), 'Y'), (array([[0.16990982]]), 'X')]	[(array([[0.37536629]]), 'R'), (array([[0.29056448]]), 'K'), (array([[0.08814506]]), 'M')]	[(array([[0.26984054]]), 'X'), (array([[0.21376911]]), 'H'), (array([[0.18615968]]), 'N')]
O	[(array([[0.22340769]]), 'U'), (array([[0.17279556]]), 'L'), (array([[0.09412036]]), 'O')]	[(array([[0.38225552]]), 'O'), (array([[0.35944113]]), 'C'), (array([[0.05439953]]), 'E')]	[(array([[0.55050568]]), 'O'), (array([[0.2479266]]), 'C'), (array([[0.07777696]]), 'U')]	[(array([[0.40807463]]), 'P'), (array([[0.15701126]]), 'R'), (array([[0.11473503]]), 'M')]	[(array([[0.33260492]]), 'S'), (array([[0.2653253]]), 'C'), (array([[0.1596783]]), 'Z')]	[(array([[0.95543685]]), 'O'), (array([[0.01948414]]), 'C'), (array([[0.01895675]]), 'D')]	[(array([[0.90461061]]), 'O'), (array([[0.08494297]]), 'D'), (array([[0.00426133]]), 'J')]	[(array([[0.90359414]]), 'O'), (array([[0.05602485]]), 'Q'), (array([[0.03174919]]), 'D')]	[(array([[0.77180826]]), 'O'), (array([[0.20248421]]), 'Q'), (array([[0.01034152]]), 'C')]
P	[(array([[0.20480246]]), 'X'), (array([[0.17296983]]), 'D'), (array([[0.13417921]]), 'Z')]	[(array([[0.18539513]]), 'T'), (array([[0.18168018]]), 'A'), (array([[0.14724842]]), 'D')]	[(array([[0.79523005]]), 'P'), (array([[0.06332382]]), 'D'), (array([[0.05202114]]), 'E')]	[(array([[0.9275946]]), 'P'), (array([[0.03686291]]), 'Y'), (array([[0.01263731]]), 'R')]	[(array([[0.46465497]]), 'T'), (array([[0.26858847]]), 'C'), (array([[0.22498955]]), 'Y')]	[(array([[0.6034389]]), 'P'), (array([[0.36258818]]), 'T'), (array([[0.01535728]]), 'O')]	[(array([[0.95421253]]), 'P'), (array([[0.03839544]]), 'T'), (array([[0.00357152]]), 'Y')]	[(array([[0.99763213]]), 'P'), (array([[0.0010738]]), 'R'), (array([[0.0008581]]), 'F')]	[(array([[0.51913991]]), 'P'), (array([[0.23537872]]), 'T'), (array([[0.08695891]]), 'E')]
Q	[(array([[0.2653574]]), 'D'), (array([[0.14939204]]), 'C'), (array([[0.11005514]]), 'A')]	[(array([[0.21432278]]), 'G'), (array([[0.20136936]]), 'A'), (array([[0.14375177]]), 'Q')]	[(array([[0.43055201]]), 'U'), (array([[0.37564388]]), 'S'), (array([[0.04455739]]), 'D')]	[(array([[0.6830055]]), 'Q'), (array([[0.17851005]]), 'D'), (array([[0.07859211]]), 'O')]	[(array([[0.37376709]]), 'D'), (array([[0.30090737]]), 'A'), (array([[0.20316846]]), 'Q')]	[(array([[0.43608554]]), 'W'), (array([[0.31719511]]), 'Q'), (array([[0.10193807]]), 'A')]	[(array([[0.97520525]]), 'D'), (array([[0.01805604]]), 'J'), (array([[0.00637292]]), 'O')]	[(array([[0.57148608]]), 'Q'), (array([[0.17857267]]), 'O'), (array([[0.14384545]]), 'D')]	[(array([[0.5596256]]), 'Q'), (array([[0.42496919]]), 'G'), (array([[0.01038113]]), 'O')]
R	[(array([[0.12627457]]), 'Q'), (array([[0.10227341]]), 'A'), (array([[0.08938058]]), 'T')]	[(array([[0.18441148]]), 'N'), (array([[0.15676669]]), 'L'), (array([[0.12968311]]), 'A')]	[(array([[0.52062228]]), 'P'), (array([[0.09007961]]), 'E'), (array([[0.07195204]]), 'D')]	[(array([[0.39518993]]), 'K'), (array([[0.20005926]]), 'R'), (array([[0.13819798]]), 'N')]	[(array([[0.31786282]]), 'B'), (array([[0.12340447]]), 'E'), (array([[0.11819098]]), 'Z')]	[(array([[0.60132521]]), 'E'), (array([[0.16002932]]), 'F'), (array([[0.04132386]]), 'B')]	[(array([[0.73240937]]), 'K'), (array([[0.09057887]]), 'R'), (array([[0.07426303]]), 'N')]	[(array([[0.28557082]]), 'Z'), (array([[0.23957307]]), 'D'), (array([[0.20297703]]), 'I')]	[(array([[0.61814922]]), 'R'), (array([[0.29985633]]), 'K'), (array([[0.04535839]]), 'E')]
S	[(array([[0.57819805]]), 'D'), (array([[0.08756216]]), 'Z'), (array([[0.07478275]]), 'S')]	[(array([[0.49836381]]), 'S'), (array([[0.11076356]]), 'V'), (array([[0.04804533]]), 'I')]	[(array([[0.31957209]]), 'T'), (array([[0.15821029]]), 'S'), (array([[0.1427065]]), 'Q')]	[(array([[0.51992025]]), 'W'), (array([[0.26986731]]), 'K'), (array([[0.12560686]]), 'L')]	[(array([[0.92932428]]), 'S'), (array([[0.02945409]]), 'G'), (array([[0.01200218]]), 'B')]	[(array([[0.91604603]]), 'S'), (array([[0.0311895]]), 'Q'), (array([[0.01963448]]), 'G')]	[(array([[0.9481134]]), 'S'), (array([[0.01992041]]), 'B'), (array([[0.01540325]]), 'G')]	[(array([[0.97346416]]), 'B'), (array([[0.01974061]]), 'S'), (array([[0.00339783]]), 'Z')]	[(array([[0.84380987]]), 'S'), (array([[0.05841573]]), 'B'), (array([[0.03675649]]), 'E')]
T	[(array([[0.65091843]]), 'T'), (array([[0.0840724]]), 'S'), (array([[0.07957313]]), 'J')]	[(array([[0.1856369]]), 'R'), (array([[0.17011928]]), 'T'), (array([[0.08700397]]), 'J')]	[(array([[0.48934582]]), 'S'), (array([[0.20193173]]), 'C'), (array([[0.03832136]]), 'V')]	[(array([[0.39585813]]), 'S'), (array([[0.37279888]]), 'T'), (array([[0.08037228]]), 'E')]	[(array([[0.97334062]]), 'T'), (array([[0.02431528]]), 'Y'), (array([[0.00096166]]), 'J')]	[(array([[0.9947697]]), 'T'), (array([[0.00401177]]), 'P'), (array([[0.00039137]]), 'I')]	[(array([[0.75473856]]), 'T'), (array([[0.23994683]]), 'P'), (array([[0.00427597]]), 'Y')]	[(array([[0.92541834]]), 'T'), (array([[0.04484739]]), 'I'), (array([[0.00892502]]), 'Q')]	[(array([[0.99137672]]), 'T'), (array([[0.00819874]]), 'P'), (array([[0.00015306]]), 'F')]

U	[(array([[0.34886562]]), 'D'), (array([[0.11102243]]), 'O'), (array([[0.09836732]]), 'X')]	[(array([[0.23772757]]), 'T'), (array([[0.15378636]]), 'W'), (array([[0.12162036]]), 'N')]	[(array([[0.50685419]]), 'O'), (array([[0.11704172]]), 'C'), (array([[0.10049582]]), 'U')]	[(array([[0.90929824]]), 'V'), (array([[0.02845439]]), 'W'), (array([[0.02802638]]), 'U')]	[(array([[0.96989214]]), 'U'), (array([[0.01003981]]), 'O'), (array([[0.00382128]]), 'G')]	[(array([[0.54089223]]), 'V'), (array([[0.20449585]]), 'W'), (array([[0.06886058]]), 'H')]	[(array([[0.43352842]]), 'Y'), (array([[0.23069702]]), 'N'), (array([[0.09464839]]), 'P')]	[(array([[0.83399539]]), 'U'), (array([[0.0660916]]), 'D'), (array([[0.04099788]]), 'W')]	[(array([[0.85954814]]), 'U'), (array([[0.0447089]]), 'V'), (array([[0.02744518]]), 'J')]
V	[(array([[0.2859512]]), 'N'), (array([[0.20332509]]), 'R'), (array([[0.1436052]]), 'H')]	[(array([[0.17640036]]), 'Y'), (array([[0.12756764]]), 'U'), (array([[0.1039186]]), 'J')]	[(array([[0.21441988]]), 'H'), (array([[0.10829336]]), 'Q'), (array([[0.09968956]]), 'N')]	[(array([[0.83169306]]), 'V'), (array([[0.05811176]]), 'Y'), (array([[0.0490813]]), 'H')]	[(array([[0.78539065]]), 'V'), (array([[0.07473416]]), 'Y'), (array([[0.04011519]]), 'U')]	[(array([[0.70766945]]), 'Y'), (array([[0.21600666]]), 'T'), (array([[0.02887128]]), 'X')]	[(array([[0.53298167]]), 'U'), (array([[0.439969]]), 'V'), (array([[0.01850015]]), 'J')]	[(array([[0.84471077]]), 'U'), (array([[0.05320468]]), 'C'), (array([[0.03775504]]), 'V')]	[(array([[0.91796556]]), 'V'), (array([[0.02880352]]), 'Y'), (array([[0.01376491]]), 'U')]
W	[(array([[0.26521746]]), 'U'), (array([[0.1344968]]), 'N'), (array([[0.07201335]]), 'G')]	[(array([[0.42139429]]), 'A'), (array([[0.11830299]]), 'T'), (array([[0.08070585]]), 'N')]	[(array([[0.36504798]]), 'L'), (array([[0.15835822]]), 'W'), (array([[0.11296195]]), 'N')]	[(array([[0.57582932]]), 'R'), (array([[0.18324913]]), 'N'), (array([[0.15658857]]), 'W')]	[(array([[0.86854116]]), 'U'), (array([[0.05701397]]), 'N'), (array([[0.02534986]]), 'W')]	[(array([[0.39132588]]), 'V'), (array([[0.29741709]]), 'U'), (array([[0.28206326]]), 'W')]	[(array([[0.52006758]]), 'W'), (array([[0.47530983]]), 'U'), (array([[0.00377811]]), 'L')]	[(array([[0.90400168]]), 'U'), (array([[0.03303712]]), 'V'), (array([[0.0319164]]), 'N')]	[(array([[0.92252012]]), 'W'), (array([[0.06669113]]), 'N'), (array([[0.00366553]]), 'U')]
X	[(array([[0.23394138]]), 'C'), (array([[0.15357837]]), 'D'), (array([[0.08578747]]), 'J')]	[(array([[0.28550051]]), 'T'), (array([[0.13247123]]), 'C'), (array([[0.09589432]]), 'L')]	[(array([[0.52070837]]), 'M'), (array([[0.12597466]]), 'W'), (array([[0.07802404]]), 'T')]	[(array([[0.48239206]]), 'R'), (array([[0.36667749]]), 'K'), (array([[0.09672772]]), 'N')]	[(array([[0.24511674]]), 'E'), (array([[0.18806269]]), 'K'), (array([[0.10345744]]), 'R')]	[(array([[0.18413498]]), 'R'), (array([[0.15190403]]), 'P'), (array([[0.14920299]]), 'I')]	[(array([[0.50742837]]), 'L'), (array([[0.17626057]]), 'X'), (array([[0.12328394]]), 'K')]	[(array([[0.8972593]]), 'X'), (array([[0.06446068]]), 'K'), (array([[0.02211627]]), 'Y')]	[(array([[0.9566823]]), 'Y'), (array([[0.02618069]]), 'X'), (array([[0.01217231]]), 'B')]
Y	[(array([[0.18208575]]), 'T'), (array([[0.16504798]]), 'S'), (array([[0.13082974]]), 'Q')]	[(array([[0.14583016]]), 'T'), (array([[0.118293]]), 'S'), (array([[0.0855256]]), 'D')]	[(array([[0.32987962]]), 'X'), (array([[0.19720429]]), 'R'), (array([[0.1361013]]), 'B')]	[(array([[0.95371121]]), 'Y'), (array([[0.04570236]]), 'X'), (array([[0.00022438]]), 'T')]	[(array([[0.63100041]]), 'P'), (array([[0.12491978]]), 'Y'), (array([[0.11966395]]), 'T')]	[(array([[0.97467691]]), 'Y'), (array([[0.01416913]]), 'X'), (array([[0.00399637]]), 'I')]	[(array([[0.59597229]]), 'Y'), (array([[0.20821708]]), 'T'), (array([[0.11488259]]), 'X')]	[(array([[0.97564881]]), 'Y'), (array([[0.00967525]]), 'I'), (array([[0.00563217]]), 'X')]	[(array([[0.88147206]]), 'Y'), (array([[0.04257007]]), 'B'), (array([[0.03284772]]), 'X')]
Z	[(array([[0.18637554]]), 'D'), (array([[0.16288372]]), 'T'), (array([[0.1050119]]), 'K')]	[(array([[0.17716357]]), 'T'), (array([[0.13177398]]), 'A'), (array([[0.09995405]]), 'D')]	[(array([[0.69846147]]), 'T'), (array([[0.09675195]]), 'Z'), (array([[0.08610298]]), 'Y')]	[(array([[0.45047587]]), 'J'), (array([[0.37491478]]), 'Z'), (array([[0.12495858]]), 'B')]	[(array([[0.58633704]]), 'J'), (array([[0.20718286]]), 'Q'), (array([[0.08683831]]), 'D')]	[(array([[0.579794]]), 'Z'), (array([[0.22060641]]), 'J'), (array([[0.12353063]]), 'L')]	[(array([[0.83874035]]), 'J'), (array([[0.14646194]]), 'S'), (array([[0.00834702]]), 'Q')]	[(array([[0.42129594]]), 'Z'), (array([[0.27726237]]), 'B'), (array([[0.15000329]]), 'I')]	[(array([[0.84320207]]), 'Z'), (array([[0.0760809]]), 'B'), (array([[0.02708855]]), 'V')]
Accuracy of first guess when reading my handwriting	3.85%	8.33%	15.38%	38.46%	26.92%	34.62%	46.15%	53.85%	61.54%
Accuracy of top 2 guess when reading my handwriting	7.69%	19.23%	34.62%	57.69%	38.46%	0.4615384615	76.92%	69.23%	76.92%
Accuracy of top 3 guess when reading my handwriting	23.08%	30.77%	50.00%	65.38%	53.85%	57.69%	80.77%	76.92%	84.62%
Average accuracy of first guess reading my handwriting	9.19%			33.33%			53.85%		
Average accuracy of top 2 guess when reading my handwriting	20.51%			47.44%			74.36%		
Average accuracy of top 3 guess when reading my handwriting	34.62%			58.97%			80.77%		

Accuracy of dataset - accuracy of handwriting (top 1 guess)	-41.21%	-40.22%	-38.92%	-37.63%	-48.54%	-41.07%	-34.13%	-26.96%	-19.88%
Accuracy of dataset - accuracy of handwriting (top 2 guess)	-37.36%	-29.32%	-19.69%	-18.40%	-37.00%	-29.53%	-3.36%	-11.58%	-4.49%
Accuracy of dataset - accuracy of handwriting (top 3 guess)	-21.97%	-17.79%	-4.31%	-10.71%	-21.61%	-17.99%	0.48%	-3.88%	3.20%
Average accuracy of dataset - accuracy of handwriting (top 1 guess)	-40.12%			-42.41%			-26.99%		
Average accuracy of dataset - accuracy of handwriting (top 2 guess)	-28.79%			-28.31%			-6.48%		
Average accuracy of dataset - accuracy of handwriting (top 3 guess)	-14.69%			-16.77%			-0.07%		