The Math in Neural Networks

Abstract

This investigation will be focusing on the math used to efficiently design an algorithm that will detect patterns in images and have the ability to make out what that image is This question focuses on the overall efficiency and success of the algorithm built.

Introduction

In this exploration, the question "the math behind Neural Networks to develop efficient algorithms to detect the number 4 from images" will be investigated. Neural Networks are architectures used in Artificial intelligence (AI). Their main function is to recognize patterns and solve common problems in the field of AI, machine learning and deep learning. This method of AI has been used by Tesla to create self-driving cars that are able to detect when there is a stop sign, a human, a cat or a dog for example. Throughout this investigation, we will be focusing on the math used to efficiently design an algorithm that will detect patterns in images and have the ability to make out what that image is. Gradient descent is a complex idea in math that uses vector calculus to determine the numerical estimation of where a function outputs its lowest value. In other words, its an algorithm that is used to determine a local minimum on a curved surface. This is done by taking steps that approach the minimum, in the appropriate direction. Similar to regular calculus, where we found local minimums or maximums, gradient descent is looking for the minimum on curved surfaces. Gradient descent is used specifically because, when analysing Neural Networks, the curved surface can exist in 9 dimensions, 10 dimensions and even 1 million, it all depends on the number of weights in the network. Below, figure 1 depicts how this algorithm works, by showing the small steps that are taken to reach the local minima on the curve.

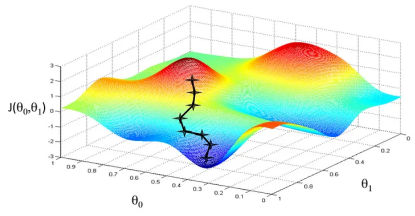


Figure 1

The efficiency of Neural Networks relies heavily on mathematical processes to optimize the results. Neural Networks consist of multiple layers. There is an input layer, hidden layers and an output layer. I will provide a much deeper look and definition to each of these layers, for now, it's important to understand that as programmers we pass in the information that starts at the input, and that data goes through hidden layers where a bunch of computations take place. All the computations take place in something called nodes. To begin with, the number of nodes in the input layer depends on the amount of information passed, in our case, from the image to the algorithm. This is usually dependent on the number of pixels in the image or some other parameter. When the information is passed through the hidden layers and to the output layer, where it is assigned a value or in other words what the system believes the image is. This could be a cat, dog, human etc. When the system is run over and over something called a loss function is used to determine the average loss over the entire training dataset. Gradient descent is used to minimise the loss function in order to optimize the algorithm. Neural networks are trained by adjusting the link weights.

The investigation of the mathematical implication in Neural Networks is crucial for understanding the method to its fullest. Furthermore, with this understanding new innovation and a better overall grasp of how machine learning can be obtained.

Investigating

In order to begin writing this paper, I reinforced my understanding of the calculus basics and my understanding of simple vectors and matrices. I used the textbook provided by my school and also did independent research. I used sources like ‘Khan Academy and ‘3blu1brown’. To develop a better understanding of Neural networks I took the Machine Learning with Python course offered by ‘FreeCodeCamp’. I also watched many youtube videos including the ‘1brown3blues’ playlist devoted to this topic and used websites like http://colah.github.io/

I reached out to the professors at my local university (University of Windsor) to develop a better understating of both Gradient descent and Neural Networks and many experts on LinkedIn. I spoke with a Grand Master on Kaggle who took university courses to understand the math behind these networks as well.

Body

The following section outlines the necessary terms and concepts necessary for understanding mathematics.

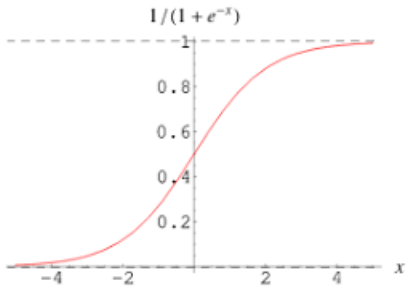
Loss function: This is a function that is responsible for measuring the number of errors in the machine. Therefore it looks at the predicted value from the machine against the actual value. Its main function is to provide feedback to the algorithm

Weights: parameter within a neural network that represents the strength of the connection between two neurons

Bias: parameter in the Neural Network which is used to adjust the output

Activation function: a function used by the neural network to squash the values from the input between two given values. In this case between 1 and 0

Sigmoid Function: a function that is used to squash values between 1 and 0 because it has horizontal asymptotes at 0 and 1, $\frac{1}{1+e^{-x}}$



Node: Can be thought of as a little room where all the computations in a Neural Network take place. There are multiple nodes in each layer of the network, in the figures presented in this paper, they are represented by little circles. (note node and neuron are synonymous)

Variables and Constants

i: input value

b: bais

σ(x): the sigmoid function $\frac{1}{1+e^{-x}}$

y: the value of the actual answer

$\hat{y}$: the guess produced by the network

n: the number of input/ elements in the network

η: learning rate

Out: the value produced by each node, after running values through the sigmoid function

Net: the value produced simply from the computation between the previous layer and the weights attaching to the next layer.

K: the final layer in the network

$\vec{net}^{k}$: vector containing all the k values from the nodes

$\vec{out}^{k}$:

Matrices

Before diving deep into the math behind Neural Networks, I would briefly like to define what a matrix is as I speak of them quite a bit in the first half of this paper. In mathematics, there is something called matrices, which are used to organize large dimensions of data. In terms of the Neural Networks input and weighted matrices are used to organize data and information to pass to the next layer of the network.

Explained through example

To being to understand how these networks work I would like to look at the face of a die. There are several different faces we can get, 1, 2, 3, 4, 5, or 6 and each will look different from one another. To us humans this is pretty easy, we look at a die with 1 dot and we understand it to be a die of value 1. However, imagine trying to explain this to a computer, that's when things become difficult. To break this down and make it make sense, I like to explain pretend like the computer is a child that has yet to live through the world and gain knowledge. Note computers are really good at math and can compute anything lightning fast, so we can consider the computer's language to be math, this would be the children's first language equivalent. For this specific example, the parents

have to teach the children what a die is and what the different patterns on the die mean. To communicate this to the computer we need to somehow convey the image of a die to a computer in their language. We can start by assigning a filled-in circle of value 1 and a blank spot as 0 and have the different potential areas for the circles to be filled in different values in a vector.

If we look at a die it has 7 locations for possible dots to be filled in to make different values.

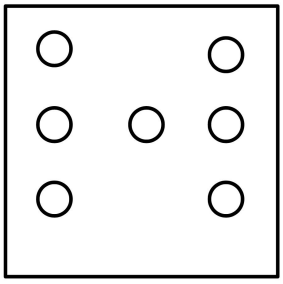


Figure 2

If we assign each of the dots a value a to g, the vector that would be fed to the network would look something like this.

$$[a,\ b,\ c,\ d,\ e,\ f,\ g]$$

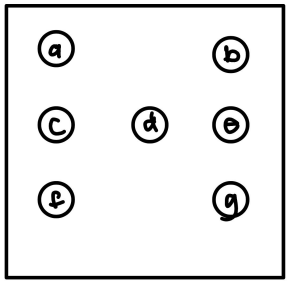


Figure 3

Looking at an example with the dots coloured in it would look something like this

$$[1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1]$$

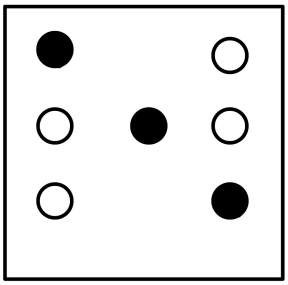


Figure 4

Now that the image is broken down into the computer's language, we also have to tell the computer what patterns to look for. We can do this by programming something called a Neural Network. In simple terms, a Neural Network is just the computer doing a bunch of fun calculus and matrix math to train the computer to identify patterns, like a human does, using data from a training sample. There are different layers to the network, we have the input layer, hidden layers and finally an output layer. The input layer is sending in the initial information, for the die example, its sends in that vector, each of the hidden layers are assigned different roles to look for patterns that would like to identify what the potential output is. Therefore, the hidden layer is responsible for breaking down key characteristics and specific patterns. Using the die example, when we feed this vector into the network, we can have a hidden layer for each of the possible locations to have a coloured-in dot. Now the idea of a Neural Network is to break this image down in the hidden layer. To really understand this, let's say that each layer looks for whether a dot is filled in at each specific location. So the first hidden layer may look for the middle, then the second for the top left and so on. Note this would look a little different for a real example and we are just using this to explain the fundamental idea. Finally, the output tells us the prediction the network made.

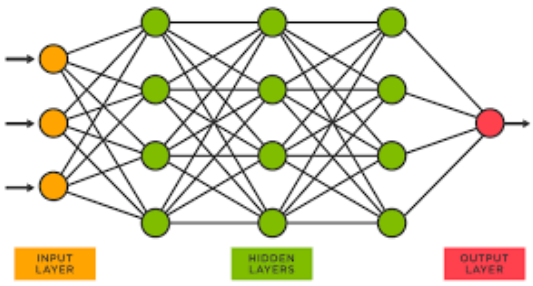
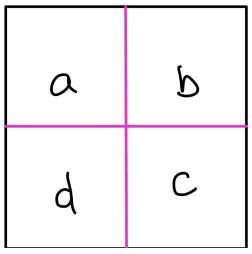


Figure 5

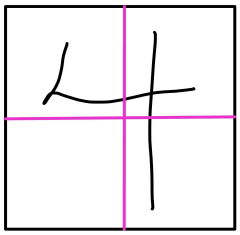
Let's go back and take a look at a die of value 3. When we humans first look at the die, we see three dots, and our brain analyses the dots by recalling past imagines of a die of value 3 that you have seen in your life and assigning the value to the die by comparing them and identifying the same pattern that you previously saw, the line of three dots. This is exactly how a network works, goes through a training process, where it receives data and makes a guess about that information, and we the programmers tell the network whether or not they did a good job. As we communicate with the computer, telling it how efficient it was, the computer is learning and becoming better at identifying the pattern for a die of value 3. This learning process is done through a process called backpropagation, we are able to quantify the efficiency, or in other words how well the network performed, back to the network. This can be thought of as childhood, where you learn patterns, and parents tell you whether those are right, or wrong and eventually, you are able to make decisions on your own when you get older. Therefore, once the computer goes through its "childhood" it can make its own decision. Backpropagation using the loss function

The example presented with the die would be way too complex to compute by hand, so lets go for something a little bit easier and would with this picture of the four is drawn using a 2x2 pixel screen and we zoomed in to see this picture. So the first thing to do is to convert this to the computer's language. To understand how the values in the vector are determined, each pixel would be assigned a value based on how "filled in" or in other words how dark the pixel is, the darker the pixel, the closer the value is to 1. Below I demonstrated how each component of the vector was determined.



$$[a, b, c, d]$$

For this example of the 4.



the vector would look something like this

$$[0.5,\ 0.56,\ 0.3,\ 0]$$

Now moving onto the network this is kind of random, and the design of the network is usually determined by the programmer. Therefore, I designed this network here to only have one layer to keep the computations relatively simple. Note each of the values in the vector will act as input.

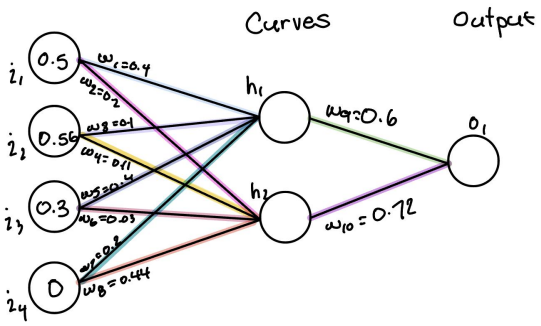


Figure 6

Here we can see that each of the values from the pixel image is acting as inputs into the networks. I also labelled the function of each hidden layer, maybe one looks for sticks and the other looks for curves in the images. Now you may be a little confused about all these lines here. The lines are just randomized values picked by the computer. The weights are what connect the values from one layer to another. The weight applies the emphasis on each of the values. Which essentially means that the greater the weight, the greater the importance of that value. These values will change through the learning process (backpropagation). Now that we have a visual, let's start by calculating the values of the network layer. This can be done by taking the input values and multiplying them by the weight that attaches them to the next layer.

Computation in Node

Before jumping into calculations, I would to formally address the computations that occur in each node. There are two parts of a node, the left side is the linear side, where the weighted matrix and the input matrix are multiplied, using the dot product. Then the sum of all these computations is used, then put into the sigmoid function. Essentially, all the weights that connect the previous layer to the new layer are multiplied by the value that the weight was connecting. This resulting vector or matrix is run through the sigmoid function to make the values between 1 and 0, so the computer can manage everything.

$$[0.5,\ 0.56,\ 0.3,\ 0]\begin{bmatrix}0.4\\0.1\\0.4\\0.2\end{bmatrix}$$

$$= (0.4)(0.5) + (0.1)(0.56) + (0.4)(0.3) + (0)(0.2)$$

$$= 0.376$$

Now that we got this value, we need to run it through the sigmoid function

$$\sigma\ (0.376) = 0.407$$

$$[0.5,\ 0.56,\ 0.3,\ 0]\begin{bmatrix}0.2\\0.11\\0.05\\0.44\end{bmatrix}$$

$$= (0.5)(0.2) + (0.56)(0.11) + (0.3)(0.05) + (0)(0.44)$$

$$= 0.1766$$

$$\sigma\ (0.1766) = 0.455$$

This same process is done output layer

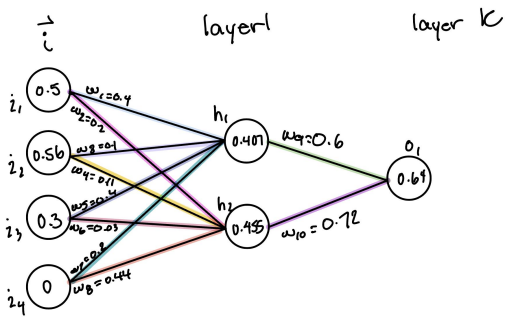


Figure 7

To better understand the mathematics, I would like to zoom in on this picture and define some variables. As shown above, each node in the network is broken down into two sections. Each is responsible for different computations. I would like to define Net as being half of the node responsible for multiplying the weights with the values of the nodes in the previous layer. When that value is run through the sigmoid function, that will be defined at the out value.

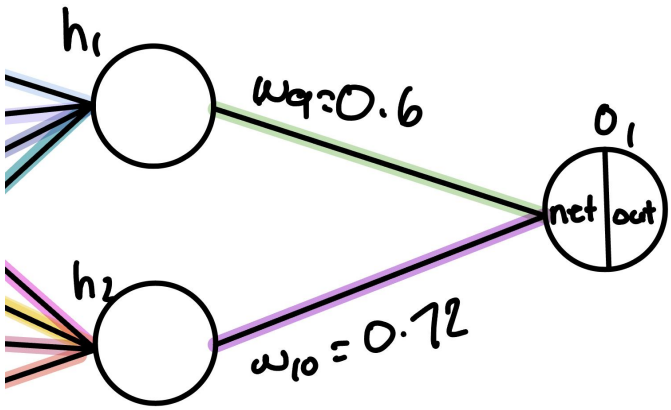


Figure 8

Now that the computer worked through all the math in the forward direction and made a decision, we as the programmers have to tell the program how well it did. For example, when you are younger and trying to learn, your parents always tell you if the action/ decisions you just made was right or wrong, and you go back and try to fix it. This is exactly what the backpropagation process does. We use something called the loss function to measure how well the network performed, then we tell it to go back and fix the weights to make the network as

efficient as possible. To measure how well the network did, we simply compare what the network should have predicted to what it did, and square the value to make it positive. When the add all the decisions made analyze it. Now we want to minimise this, in order to reduce the loss as much as possible. This is where we use something called partial derivatives.

In basic calculus, we learn about normal derivatives. They tell us information about the change of the y values with respect to the x value. However, what happens when we expand that to higher dimensions? To explain this I will be using GeoGebra which was a tool used in school when learning about vectors and matrices. The green axes represent the y-axis, the blue is the z-axis and the red is the x-axis.

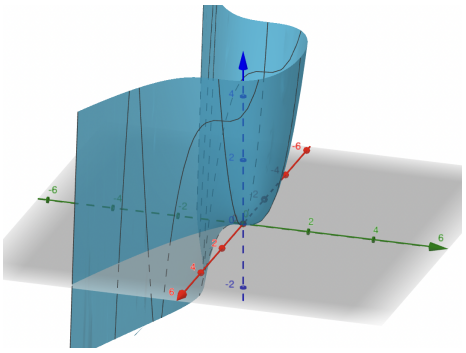


Figure 9

Figure 9 is the graph of f(x,y) = $x^2 + y^3$. In this case, there are multiple dimensions to the input space rather than just one which is typically used when learning elementary calculus.

$$\frac{dz}{dx}f(x,y)$$

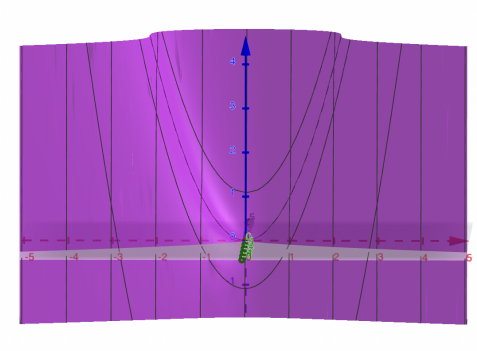


Figure 10

Figure 10 nicely demonstrates that if you take slices parallel to the plane y=0, each slice looks like a quadratic function. That's what the partial derivative represents. It looks at the change of the function with respect to one variable while all others are held constant.

$\frac{dz}{dy}$f(x,y)

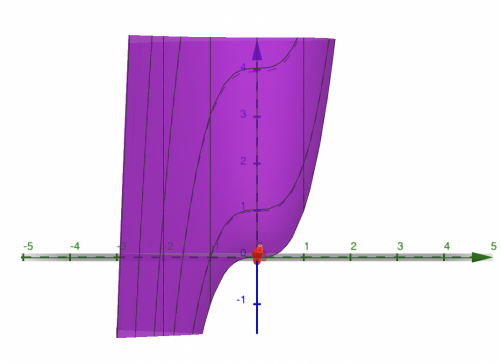


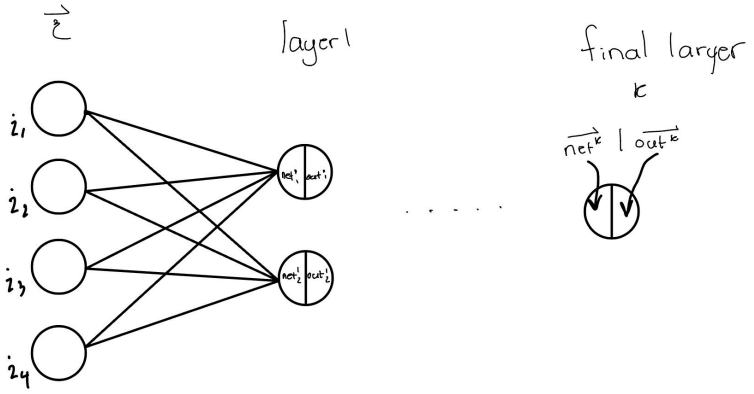
Figure 11

We can do the same thing here and examine the curve from this angle and see how the function changes with respect to the y-axis. From this perspective, it is important to note that the function looks cubic.

Overall when finding partial derivatives we first decided with what variable we are taking the partial derivative of let's take a look at f(x,y) = $x^2 + y^3$. If we wanted to take the partial derivative of the function with

respect to x we would consider y to be a constant and take the derivative in that way $\frac{\delta z}{\delta x}$f(x,y) = 2x. So what does this tell us, this gives us information on how a small change in x is affecting the function.

Okay let's move back to the loss function, we define the loss function as J(w,b) =$\frac{1}{n}\sum(y - \hat{y})$, where we add all the loss from each output node. The only thing the network can change is the weights and the bias. Therefore, we must take the partial with respect to the weights and biases. We want to minimize the loss function by adjusting the parameters w and b. Before I define the following variables please note the general network.



Allow me to define some key variables here, $\vec{net}^{k} = \vec{w} \bullet \vec{out}^{k-1} + \vec{b}$ and $\vec{out}^{k} = \sigma(\vec{net}^{k})$. We will also define $\hat{y}_i$ as the ith component of the vector $\vec{out}^{k}$ and y_i is the actual values of the component. We will use the partial derivatives and the chain rule to find $\frac{\delta J}{\delta b}$ and $\frac{\delta J}{\delta w}$ to minimize J with respect to each parameter, independently. Going back to the network we're using for this paper, recall that e $\vec{net}^{k} = \vec{w} \bullet \vec{out}^{h} = w_9($ $net_{h_1}) + w_{10}(net_{h_2})$, and note there are no biases used in this network.

In general, this is how to calculate $\frac{\delta J}{\delta w}$ for a weight connecting to the final layer.

$$\frac{\delta J}{\delta w} = \left(\frac{\delta J}{\delta \hat{y}_i}\right)\left(\frac{\delta \hat{y}_i}{\delta \vec{net}^{k}}\right)\left(\frac{\delta \vec{net}^{k}}{\delta w}\right)$$

Specifically looking at changing the values of w_9

$$\frac{\delta J}{\delta w_9} = \left(\frac{\delta J}{\delta \hat{y}_i}\right)\left(\frac{\delta \hat{y}_i}{\delta \vec{net}^{k}}\right)\left(\frac{\delta net}{\delta w_9}\right)$$

In general, the computation for $\frac{\delta J}{\delta w}$ weights in the hidden layer would look like this.

$$\frac{\delta J}{\delta w} = \left(\frac{\delta J}{\delta \hat{y}_i}\right)\left(\frac{\delta \hat{y}_i}{\delta \vec{net}^{k}}\right)\left(\frac{\delta \vec{net}^{k}}{\delta \vec{out}^{k-1}}\right)\left(\frac{\delta \vec{out}^{k-1}}{\delta \vec{net}^{k-1}}\right)\left(\frac{\delta \vec{net}^{k-1}}{\delta \vec{out}^{k-2}}\right)\ldots\left(\frac{\delta \vec{net}^{*}}{\delta w_*}\right)$$

(w_* is the desired weight in the *th layer.)

Now specifically looking at changing the values of w_1

$$\frac{\delta J}{\delta w_1} = \left(\frac{\delta J}{\delta \hat{y}_i}\right)\left(\frac{\delta \hat{y}_i}{\delta \vec{net}^{k}}\right)\left(\frac{\delta \vec{net}^{k}}{\delta \vec{out}_1^{1}}\right)\left(\frac{\delta \vec{out}_1^{1}}{\delta \vec{net}_1^{1}}\right)\left(\frac{\delta \vec{net}_1^{1}}{\delta w_1}\right)$$

Considering the derivative of the sigmoid is used extensively in the backpropagation process, I would like to quickly show the proof for the derivative.

Backpropagation

Let's move back to the example of the drawing of the 4 we had. Normally, when computing a neural network we add all the output values, however, in our case we only have one output value, therefore, we just need to manage the one value.

Gradient descent (Backpropagation)

The gradient of the loss function tells how well the network is performing. When the difference between the predicted value and the actual value is high, then we must learn how to go back and change the biases and the

weights to make it more efficient. This process is called backpropagation. This process involves going back into the network and changing the weights based on what the derivative told us. Each new weight can be adjusted using the formula.

$$w_{new} = w_{old} - \eta \frac{\delta J}{\delta w_{old}}$$

It's important to note the learning rate is controlled by us and just allows us to choose how big of steps we want to take towards the minimum point, in order to minimise the error.

This formula must be used to update each weight throughout the network. $\frac{\delta J}{\delta w_{old}}$ can be obtained uniquely for each weight. In other words, updating the weight looks different for weights in the hidden and output layer. I will be providing an example of how this is done, by computing one for the hidden layer and one for the output layer.

$$\frac{\delta J}{\delta w_9} = \left(\frac{\delta J}{\delta \hat{y}_i}\right)\left(\frac{\delta \hat{y}_i}{\delta \vec{net}^k}\right)\left(\frac{\delta net}{\delta w_9}\right)$$

Let's move back to our example and work this out

Output

$$\frac{\delta J}{\delta w_9} = \frac{\delta J}{\delta out} * \frac{\delta out}{\delta net} * \frac{\delta net}{\delta w_9}$$

$$J = \frac{1}{2}(y - out)^2$$

$$\frac{\delta J}{\delta out} = -2\left(\frac{1}{2}\right)(y - out)$$

Let y =1 as the network is trying to predict whether the picture is of a 4 or not. An output with the result 1 would indicate that the image was a 4. This is the desired output.

$$\frac{\delta J}{\delta out} = (1 - 0.64)$$

$$\frac{\delta J}{\delta out} = -0.36$$

$$\text{Out} = \frac{1}{1+e^{-net_1}}$$

$$\text{Note Out} = \sigma(net_1)$$

$$\sigma'(net_1) = \frac{e^{net_1}}{1+e^{net_1}}$$

$$\sigma'(net_1) = \frac{e^{net_1}(1+e^{net_1})(e^{net_1})}{(1+e^{net_1})^2}$$

$$\sigma'(net_1) = \frac{e^{net_1}}{(1+e^{net_1})^2}$$

$$\sigma'(net_1) = \sigma(net_1)(1-\sigma(net_1))$$

$$\frac{\delta out}{\delta net} = (0.64)(1-0.64)$$

$$\frac{\delta out}{\delta net} = 0.2306$$

$$\text{net} = w_9 * out_{h1} + w_{10} * out_{h2}$$

$$\frac{\delta net}{\delta w_9} = 1 * out_{h1}$$

$$\frac{\delta net}{\delta w_9} = 0.407$$

$$\frac{\delta J}{\delta w_9} = (-0.36)(0.2306)(0.407)$$

$$\frac{\delta J}{\delta w_9} = -0.033$$

$$w_{9\ new} = 0.6 - \eta(-0.033)$$

Let the learning rate (η) be 0.1. This is a somewhat arbitrary decision made by the programmer. Note this is also where the idea of gradient descent comes in, as we are taking small steps in direction of the gradient, to find the local minima of the loss function.

$$w_{9\ new} = 0.6 - (0.1)(-0.033)$$

$$w_{9\ new} = 0.603$$

Hidden

$$\frac{\delta J}{\delta w_1} = \left(\frac{\delta J}{\delta \hat{y}_i}\right)\left(\frac{\delta \hat{y}_i}{\delta \vec{net}^{k}}\right)\left(\frac{\delta \vec{net}^{k}}{\delta \vec{out}_1^{1}}\right)\left(\frac{\delta \vec{out}_1^{1}}{\delta \vec{net}_1^{1}}\right)\left(\frac{\delta \vec{net}_1^{1}}{\delta w_1}\right)$$

The computations will look a little different for the hidden layer as we need to account for the fact that the hidden layer output's error as well. Therefore we can say

note

$$\left(\frac{\delta J}{\delta \hat{y}_i}\right) = -1$$

$$\left(\frac{\delta \hat{y}_i}{\delta \vec{net}^{k}}\right) = \sigma'(net^{k})$$

$$= (0.64)(1-0.64)$$

$$= 0.2304$$

$$\frac{\delta out_1^{1}}{\delta net_1^{1}} = out_1(1-out_{h1})$$

$$= (0.407)(1-0.407)$$

$$= 0.241$$

$$net_{h1} = w*i + w3*i2$$

$$\frac{\delta net_{h1}}{\delta w_1} = i1$$

$$\frac{\delta net_{h1}}{\delta w_1} = 0.5$$

$$\frac{\delta \vec{net}^{k}}{\delta \vec{out}_1^{1}} = w_9$$

$$\frac{\delta \vec{net}^{k}}{\delta \vec{out}_1^{1}} = 0.6$$

$$\frac{\delta J}{\delta w_1} = (-1)(0.2304)(0.6)(0.241)(0.5)$$

$$\frac{\delta J}{\delta w_1} = -0.016$$

$$w_{1\ new} = 0.4 - \eta(-0.016)$$

$$w_{1\ new} = 0.4 - (0.1)(-0.016)$$

$$w_{1\ new} = 0.4016$$

In conclusion, this exploration determines how to update the parameters in a Neural Network in order to have a model as efficient as possible. This is done by using gradient descent. Which looks at taking small steps to find the local minima. This exploration was a very simple and straightforward example of how these computations occur. In a real example, jacobian matrices would be used to store all the partial derivatives that we calculated. Additionally, all the computations would occur in the same fashion but in might higher dimensions. There would also likely be more than one output. This would change the calculation for the change in a given weight with respect to the loss function, as we would need to consider all of the outputs and their weights attached. We would also get different values for the loss function as it is a scalar value that accounts for all the outputs. If I could I

would explore different architectures. I would also be interested in looking at the computations for Federated Learning and on-device/edge learning.

Work cited

"What is Gradient Descent?" *IBM*, https://www.ibm.com/topics/gradient-descent. Accessed 16 April 2023.

"Jacobian, Chain rule and backpropagation." *Landing Page...*, 4 April 2018, https://suzyahyah.github.io/calculus/machine%20learning/2018/04/04/Jacobian-and-Backpropagation.html. Accessed 16 April 2023.

"The Matrix Calculus You Need For Deep Learning." *explained.ai*, https://explained.ai/matrix-calculus/. Accessed 16 April 2023.

"An overview of Gradient Descent algorithms | by Anjani Kumar." *DataDrivenInvestor*, 14 May 2020, https://medium.datadriveninvestor.com/an-overview-of-gradient-descent-algorithms-e373443afa7f. Accessed 16 April 2023.

"An overview of Gradient Descent algorithms | by Anjani Kumar." *DataDrivenInvestor*, 14 May 2020, https://medium.datadriveninvestor.com/an-overview-of-gradient-descent-algorithms-e373443afa7f. Accessed 16 April 2023.

"A Review of the Math Used in Training a Neural Network." *Level Up Coding*, https://levelup.gitconnected.com/a-review-of-the-math-used-in-training-a-neural-network-9b9d5838f272. Accessed 16 April 2023.

Liu, Clare. “5 Concepts You Should Know About Gradient Descent and Cost Function.” *KDnuggets*, 16 September 2022, https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html. Accessed 16 April 2023.

Norén, Anders. “Stochastic Gradient Descent on your microcontroller.” *Eloquent Arduino*, 10 April 2020, https://eloquentarduino.github.io/2020/04/stochastic-gradient-descent-on-your-microcontroller/. Accessed 16 April 2023.

Jason,“The Math behind Neural Networks - Backpropagation.” *Jason {osa-jima}*, 18 July 2018, https://www.jasonosajima.com/backprop. Accessed 16 April 2023.