A Mathematical Introduction to the Foundation of Neural Networks

Candidate Code: jzh886

IBDP HL Mathematics EE

1 Introduction

"Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy" [17]. A subfeild of machine learning is deep learning and the backbone of its algorithms is comprised of artificial neural networks (neural networks for short). A neural network with more than three layers is a deep learning algorithm and it mimics the human brain through these algorithms [17].

Artificial intelligence is an area of study that has undergone immense development in the recent years. We have seen many breakthroughs in multiple industries where AI is solving problems that were previously thought as impossible. Examples of these breakthroughs include self-driving vehicles, chat bots, facial recognition, and natural language processing. We even see these applications in our day-to-day lives, like Siri on our cellphones, Tesla autopilot on the roads, and Google's search engine and question answers. For example, the following image is an example of a neural network algorithm that is being used by a self driving vehicle.

However, concepts within this fascinating field are usually studied by people in their master's program after various prerequisite courses. The knowledge of neural networks is not easily accessible to grade 12 students. What I have found is that the explanations behind such topics require excessive prerequisite knowledge, which hinders younger audiences to learn and have interest in this topic.

This paper aims to bridge the gap between the basic concepts of neural networks and high school and undergraduate students. It will dissect the mathematics behind artificial intelligence using fundamental concepts. This paper provides a mathematical lens on neural networks from the standpoint of a student who has finished their undergraduate requirements.

1.1 Neural Network Architecture

A neural network is a model that can make predictions based on a relationship between inputs and an output. The inputs (x_1, x_2, x_3) are connected to a set of weights (w_1, w_2, w_3), which connect to an output $\hat{y}$. Figure 1 demonstrates an example of a simple neural network architecture that has three inputs, three weights, and one output.

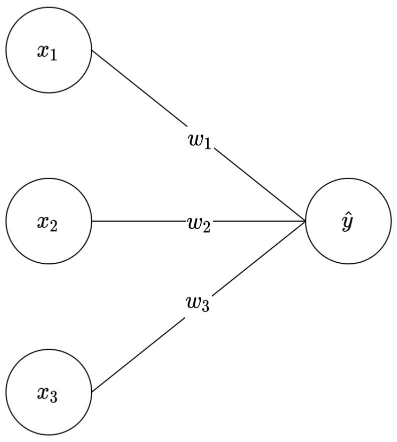


Figure 1: Neural Network of $n = 3$ inputs

The output, $\hat{y}$, is the network's prediction and can also be represented by the equation that follows,

$$\hat{y} = w_1x_1 + w_2x_2 + w_3x_3$$

where w_i is the ith weight and x_i is the ith input for $i \in \{1, 2, 3\}$. Similarly, a more generalized network of n inputs and weights can be represented by the following,

$$\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n \tag{1}$$

An alternate representation of Equation 1 is,

$$\hat{y} = \sum_{i=1}^{n} w_ix_i \tag{2}$$

which can also be displayed in vector notation as,

$$\hat{y} = \vec{x} \cdot \vec{w} \tag{3}$$

where $\vec{x} = (x_1, x_2, \ldots, x_n)$ is the input vector of length n and $\vec{w} = (w_1, w_2, \ldots, w_n)$ is the weight vector of length n. The weights, $\vec{w}$, of the neural network are initialized randomly over the range $x_i \in [0, 1]$ for $i = 1, \ldots, n$. The neural network can make a prediction, $\hat{y}$, given a set of inputs, $\vec{x}$, and a weight vector, $\vec{w}$.

The goal of training a neural network is to correctly make predictions for a given problem. This can be done by finding a set of weights, $\vec{w}$, that make the network's prediction equal to a desired target value y for all possible inputs, $\vec{x}$. Any deviation from this desired target can be seen as the error e_i from the network's prediction, $\hat{y}$, given an input set, $\vec{x}$. Thus, the network's aim is to reduce the error to its lowest outcome. The error can be defined as:

$$e_i = \frac{1}{2}(y_i - \hat{y}_i)^2 \tag{4}$$

for the ith prediction.

1.1.1 Loss Function

Simulating all possible errors, e_i, for all possible weights given a set of inputs returns a loss function. A loss function is an abstract representation of all possible errors a neural network can make for all possible weights given an input. It should be noted that this is a theoretical concept because computationally, this is impossible to generate for complex problems as there can be an infinite number of weights [7].

Prior to a visual representation of a loss function, let us recall the following terminologies.

Definition 1.1 (Algebraic Definitions) *A function f(x)*

- *is a global maximum at $x = a$ if $f(x) \leqslant f(a)$ for any x in its respective domain.*
- *is a global minimum at $x = a$ if $f(x) \geqslant f(a)$ for any x in its respective domain.*
- *is a local maximum at $x = a$ if $f(x) \leqslant f(a)$ for any x in a particular interval around $x = a$*
- *is a local minimum at $x = a$ if $f(x) \geqslant f(a)$ for any x in a particular interval around $x = a$*

Note : The saddle point is a point $x = a$ on the graph where there is both a local maximum and local minimum.

An example of a loss function is given in the following Figure 2.

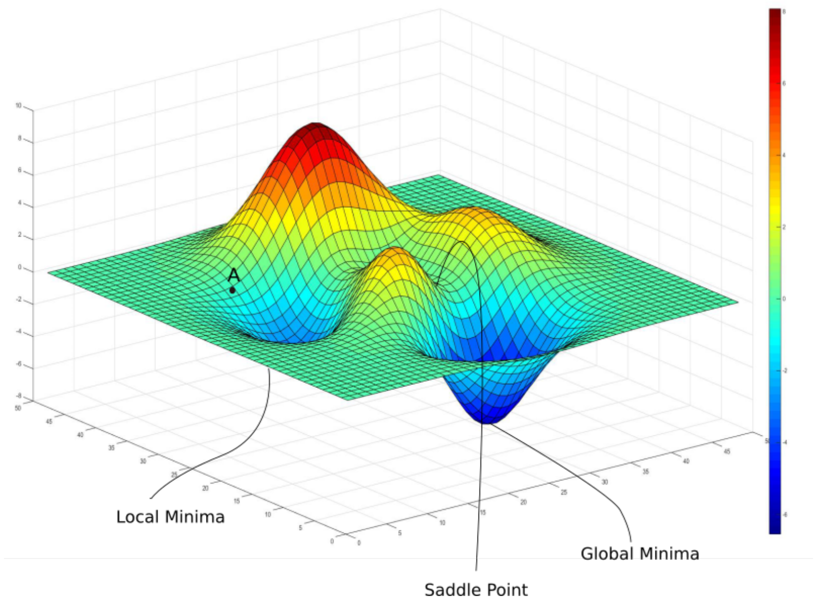


Figure 2: Loss Function of a Neural Network

where the x and y axes are the weights and z axis is the error. Recall that the goal of the neural network is to match its predicted value, $\hat{y}$, with its target value, y, by reducing the error, e_i, to its smallest form. The loss function conveniently shows the least possible error the network can have using local minimums. Thus, the local minimums are the points on the graph whose error is the least value compared to the error of the points around it. The global minimum is the smallest local minimum error value, where the error is the very least it can be [14].

'Learning' in the context of neural networks is finding a set of weights that globally minimize the loss function. One method to approximate the global minimum of the loss function is using an optimization algorithm called Gradient Descent [1][5].

1.1.2 Gradient Descent

Gradient descent is an optimization algorithm that is used to train neural networks by finding the local minimum of the loss function for the neural network. Let us review what a gradient of a function is.

Definition 1.2 (Gradient of a Function) *A gradient is differential operator, ∇ applied to a n-dimensional vector-valued function to yield a vector whose n components are the partial derivatives of the function with respect to its vectors. $\nabla f = \vec{i_1} f_1 + \vec{i_2} f_2 + \cdots + \vec{i_n} f_n$, where $f_1, f_2, f_3, \cdots f_n$ are the first partial derivatives of f and $\vec{i_1}, \vec{i_2}, \vec{i_3}, \cdots \vec{i_n}$ are the unit vectors of the vector space.*

The motivation behind gradient descent is based on a well-defined multi-variable function that is differentiable for any given point. Applying this to the loss function as seen previously, a 'gradient' can be calculated. This can be interpreted as the partial contribution to the total error in the neural network's prediction contributed the network's weights [21]. The gradient can be redefined as,

$$\nabla F(y, \vec{x}, \vec{w}) = \frac{\partial e}{\partial \vec{w}} \tag{5}$$

which can also be written as,

$$\nabla F(y, \vec{x}, \vec{w}) = [\frac{\partial e}{\partial w_1}, \frac{\partial e}{\partial w_2}, \dots, \frac{\partial e}{\partial w_n}] \tag{6}$$

The gradient can inform the direction of increase (or decrease) in the loss function based on a change in the weight, w_i for $i \in \mathbb{N}$. The negative of the gradient can then be used to update the weight of the neural network to incrementally lower the error. Equation 7 below summarizes the process of gradient descent.

$$w_{new} = w_{old} - \alpha \frac{\partial e}{\partial w_i} \tag{7}$$

where α is the learning rate. Figure 3 demonstrates an example of gradient descent at two different starting points where the errors are minimized incrementally by changing the weights.

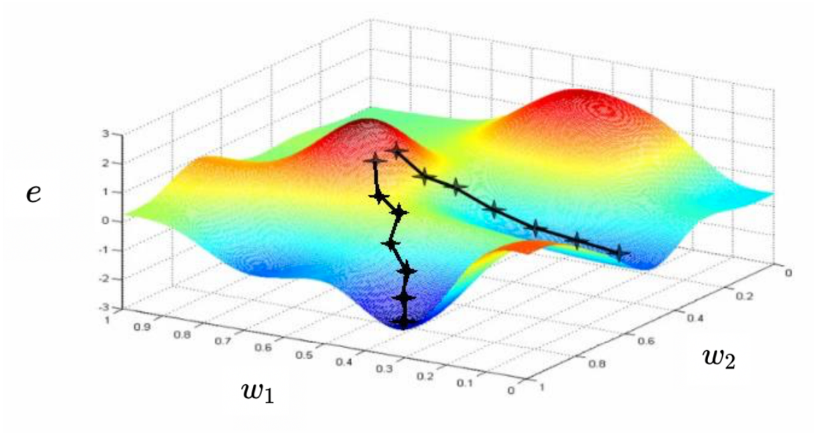


Figure 3: Gradient Descent over a Loss Function

The learning rate specifies the magnitude of change to the weight during each increment of training. The gradient does not specify the magnitude of change as it is only a slope (with direction). If the learning rate is too big, the network may never reach a local minimum as the weights of the network can oscillating between large values. If the learning rate is too small, the network may fail to find the local minimum due to limited training data and training time. The optimal learning rate is problem specific [2].

For the neural network in Equation 2 the gradient descent algorithm from Equation 7 can be applied to find a weight updating formula to train the network. The weight updating formula for a neural network with one layer is also known as the delta rule.

1.1.3 Delta Rule

The delta rule is a formula for updating the weights of a one layer neural network. For a one layer network, the prediction $\hat{y}$ is defined as Equation 1 and Equation 3. The network's error was defined as one half of the squared difference between the prediction and the target value as seen in Equation 4. Since $\hat{y} = f(\vec{w})$, we can rewrite this as,

$$e = \frac{1}{2}(y - f(\vec{w}))^2 \tag{8}$$

Using the gradient descent algorithm from Equation 7 in the previous section, the gradient for each weight is as follows,

$$\frac{\partial e}{\partial w_i} = \frac{\partial[\frac{1}{2}(y - f(w_i))^2]}{\partial w_i}$$

The aim is to find the partial derivative of the error with respect to the weights. Since the error is a function of the prediction, $\hat{y}$, and the prediction is a function of the weights, chain rule can be applied to get,

$$\frac{\partial e}{\partial w_i} = \frac{\partial e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i}$$

which simplifies to,

$$\frac{\partial e}{\partial w_i} = (y - \hat{y}) \frac{\partial \hat{y}}{\partial w_i}$$

$$\frac{\partial e}{\partial w_i} = (y - \hat{y}) \frac{\partial[w_1 x_1 + w_2 x_2 + \ldots + w_n x_n]}{\partial w_i} \,. \tag{9}$$

Only one component of the prediction in Equation 9 is used in the partial derivative. As a result, the equation simplifies to,

$$\frac{\partial e}{\partial w_i} = (y - \hat{y}) x_i \,, \tag{10}$$

which represents the gradient. This can be used in the gradient descent formula to get the weight updating equation, defined in Equation 7. Substituting the gradient, $\frac{\partial e}{\partial w_i}$ derived in Equation 10, Equation 7 becomes,

$$w_{new} = w_{old} - \alpha(y - \hat{y}) x_i \,.$$

Since $(y - \hat{y})$ is the error term, e_i,

$$w_{new} = w_{old} - \alpha \times e_i \times x_i \,. \tag{11}$$

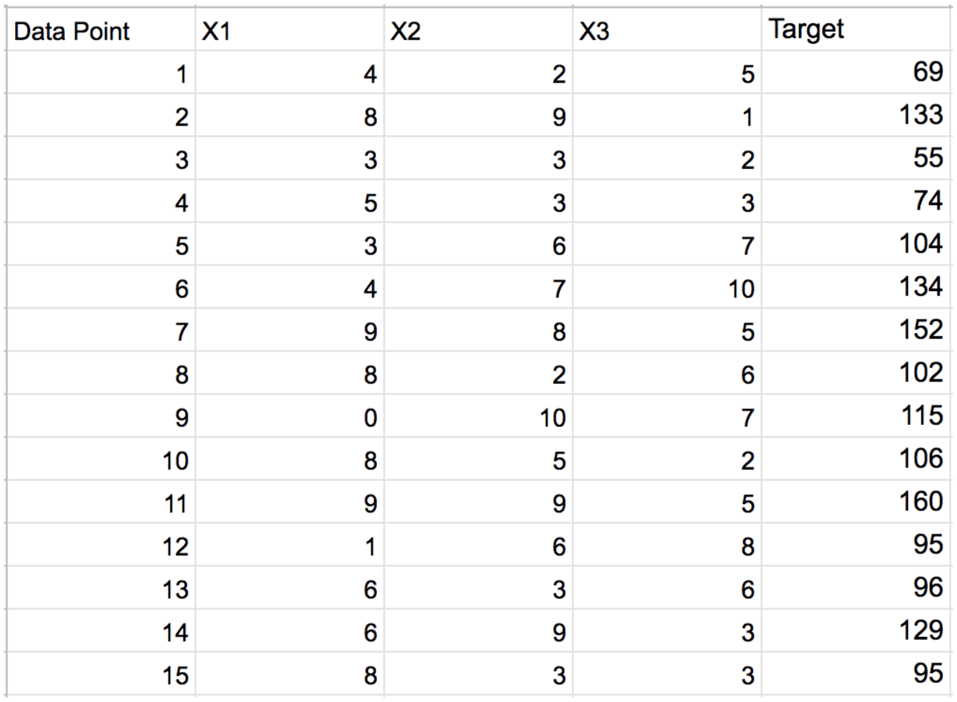
Next, we see an example of the delta rule for a one layer neural network. Consider a model that has a target prediction of,

$$y = 7x_1 + 8x_2 + 5x_3 \,, \tag{12}$$

where the weights are the coefficient of the input, x_i. A one layer neural network can learn the model in Equation 12 of the form,

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 \tag{13}$$

The goal is to learn the weights of the model, which are 7, 8, and 5, using training data. A set of 250 training data points is generated using random numbers between 0 to 10 for the inputs x_1, x_2, x_3. This provides a set of target values, which the model can learn from. The following table is a visual representation of a segment of the full dataset, in which only 15 of the 250 total training data points are shown. Note that the target is calculated by Equation 13.



Data Point	X1	X2	X3	Target
1	4	2	5	69
2	8	9	1	133
3	3	3	2	55
4	5	3	3	74
5	3	6	7	104
6	4	7	10	134
7	9	8	5	152
8	8	2	6	102
9	0	10	7	115
10	8	5	2	106
11	9	9	5	160
12	1	6	8	95
13	6	3	6	96
14	6	9	3	129
15	8	3	3	95

Figure 4: Example Training Dataset

The delta rule in Equation 11 can be applied to learn the weights, w_1, w_2, w_3. This is done by iterating over the 250 training data points. With each iteration, the weight, w_i, is learning the model parameters using the weight updating equation, Equation 11, applied from the delta rule. For example, in the first iteration we have the learning rate, $\alpha = 0.005$, with the randomly initialized weights to be $w_1 = 0.3, w_2 = 1.0, w_3 = 0.4$. Our model prediction using this configuration we get,

$$\hat{y} = 0.3 \times x_1 + 1.0 \times x_2 + 0.4 \times x_3\,. \tag{14}$$

Our first observation is $(x_1, x_2, x_3, y) = (4, 2, 5, 69)$ as seen from the first row on the table in Figure 12. Substituting this into Equation 14,

$$\hat{y} = 0.3 \times 4 + 1.0 \times 2 + 0.4 \times 5 = 5.2$$

This prediction of $\hat{y} = 5.2$ is far off the target value 69. This is not surprising as our initial weights were selected at random. The error is,

$$(y - \hat{y}) = (69 - 5.2) = 63.8\,.$$

We can use this error term to update the weights by applying the delta rule from Equation 11,

$$w_{new} = w_{old} - \alpha \times e_i \times x_i$$

Substituting the values above, each new weight value becomes,

$$w_1^1 = w_1^0 - \alpha \times e_1 \times x_1 = 0.3 - 0.005 \times 63.8 \times 4 = 1.576$$

$$w_2^1 = w_2^0 - \alpha \times e_1 \times x_2 = 1.0 - 0.005 \times 63.8 \times 2 = 1.638$$

$$w_3^1 = w_3^0 - \alpha \times e_1 \times x_3 = 0.4 - 0.005 \times 63.8 \times 5 = 1.995$$

where w_i^j is the ith weight at the jth iteration. We can see the weights are getting closer to our desired values. Repeating the above process for the full 250 data points, the weights converge to the true values. The following figure plots the results of the training session.

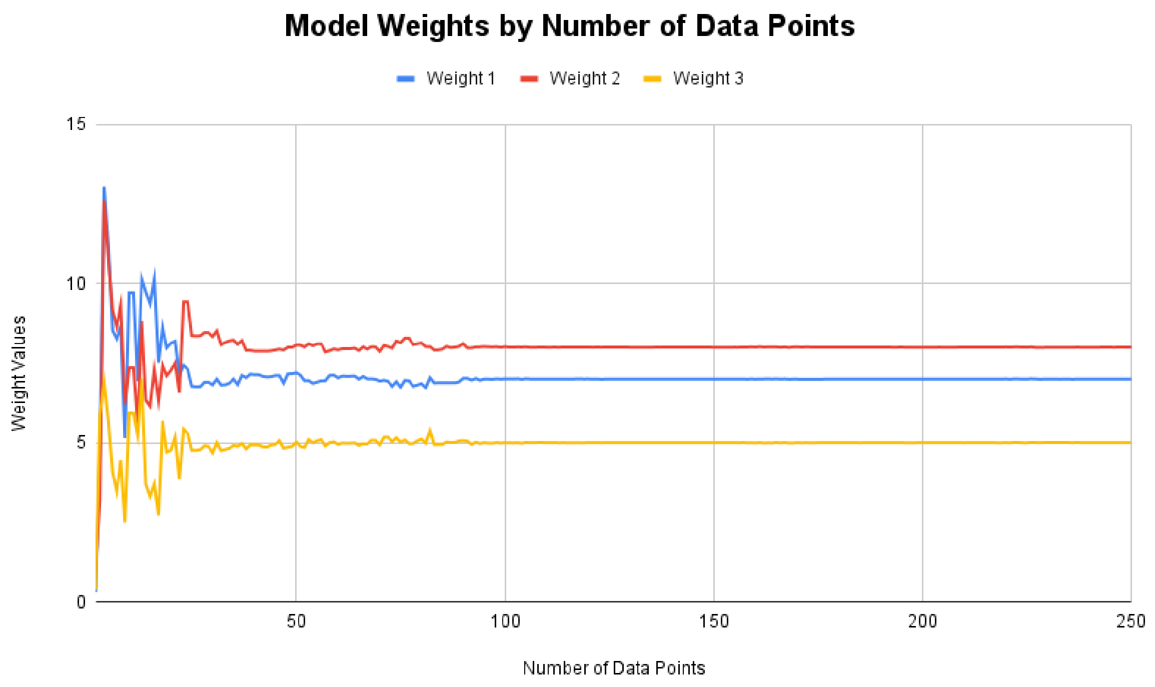


Figure 5: Training Session With $\alpha = 0.005$

It should be noted that one assumption in training the model is specifying the learning rate value, α. Upon experimentation, it was found that not all learning rates make the model weights converge to the true values. For example, the learning rate of $\alpha = 0.0178$ resulted in learned weight values of $(w_1, w_2, w_3) = (6.83, 8.19, 5.36)$. Figure 4 provides the results of this training session.

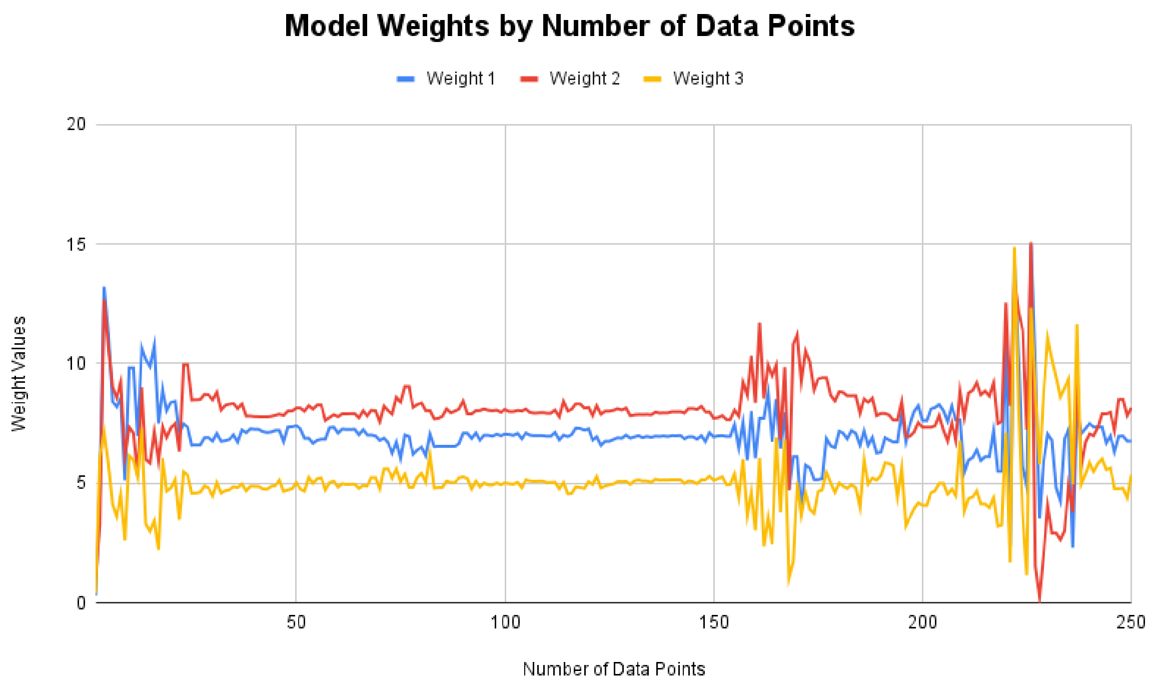


Figure 6: Training Session 2 With $\alpha = 0.0178$

In the above example, the model failed to learn the target weights. However, with a learning rate of $\alpha = 0.0171$, the model's weight values successfully converged to the target weight values, $(w_1, w_2, w_3) = (7.00, 8.00, 5.00)$. Thus, it is interesting to see how hypersensitive the value of the learning rate is when having the model learn the target values. In fact, there is significant research done in understanding the complex fragility of the learning rate and what the optimal learning rate should be for different problems in this field [2].

1.2 Activation Functions

In the previous section, we found the one layer neural network model to be very well suited in learning targets that are based off of linear functions using the Delta Rule. However, in most real world applications, the target model is not linear. For example, if we were interested in learning a model that distinguishes between two classes (0 vs. 1, cat vs. dog, etc.), then the linear model will not do a good job [29].

One method to learn non-linear models is to modify our simple neural network architecture from Figure 2.1 with the introduction of an activation function. This can be done by doing a one-to-one transformation ($\mathbb{R} \to \mathbb{R}$) on the model's predictions.

Definition 1.3 (Activation Functions) *An activation function can be defined as*

$\hat{y} = f(z)$ *where* z *is the dot product of the inputs and weights,*

$$z = \vec{w} \cdot \vec{x} = \sum_{i}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

An example of an activation function is a linear activation function, which can be defined as,

$$f(z) = z \tag{15}$$

This can be interpreted as no transformations being applied to the outputs of the network. Recall that there had been no transformations applied to the output of the network in our previous work. So, implicitly, our networks have actually been using a linear activation function all along.

Another example of an activation function is a sigmoid activation function, which can be used to have a network learn a classification task containing two outputs $(0, 1)$. The equation of the activation function is defined as follows.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{16}$$

The figure below plots the sigmoid activation function:

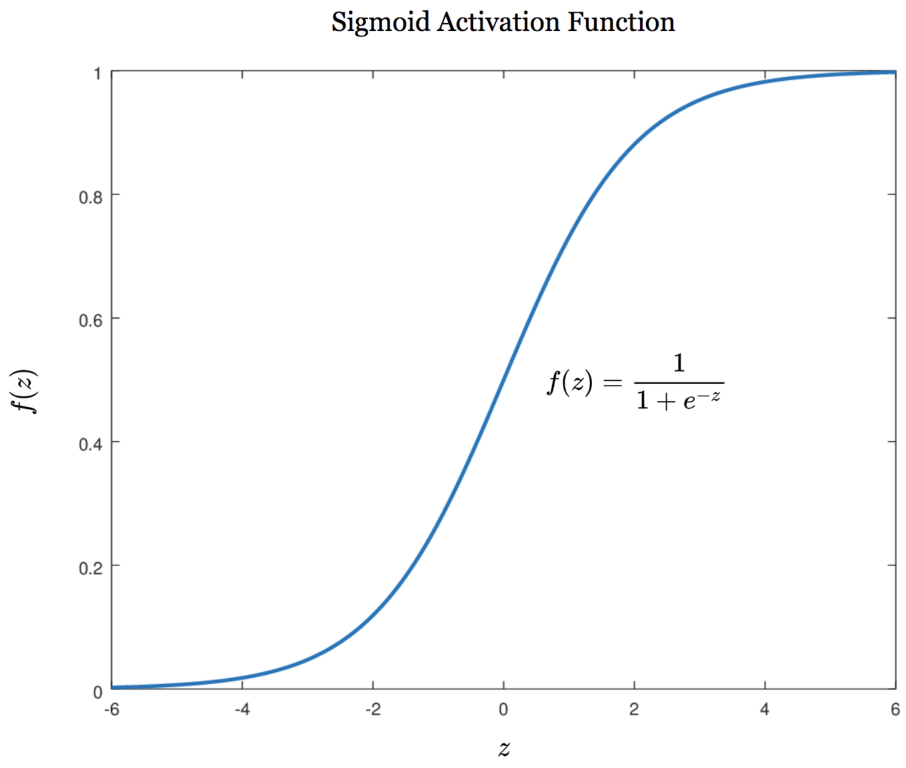


Figure 7: Sigmoid Activation Function

The sigmoid function has a range over the interval $f(z) \in [0, 1]$. The function allows a neural network with many inputs to have a target output value between the range $[0, 1]$. This would not have been possible with a linear activation function.

We can round the predicted values coming from the sigmoid function to give a binary classification output of $\{0, 1\}$. The figure below plots the regions of the function that correspond to a classification of 0 and 1 rounding up from a threshold of 0.5.

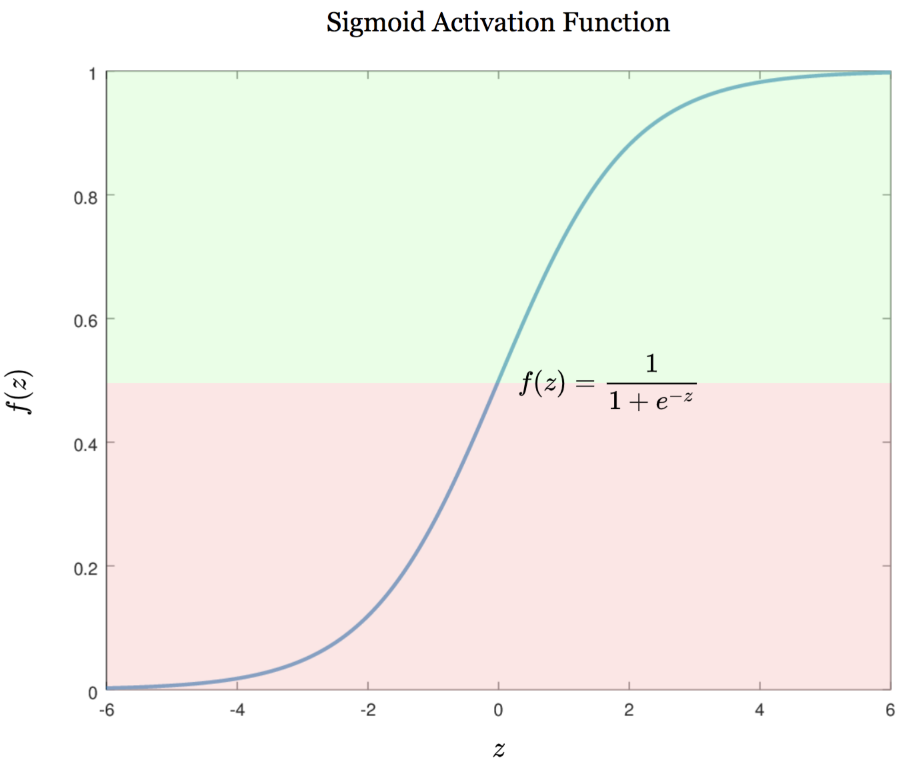


Figure 8: Sigmoid Activation Function Range

In Figure 8 the green highlighted region corresponds to a classification of 1 and the red highlighted region corresponds to a classification of 0. However, finding a set of weights to learn a model that will successfully predict a binary classification becomes increasingly challenging as the derivation is longer and more complex.

1.2.1 Gradient Descent with Sigmoid Activation Function

The following section derives the gradient descent learning algorithm for a neural network with a sigmoid activation function. Recall an $n-$layer neural network with a sigmoid activation function has a prediction function defined as:

$$\hat{y} = f(z) = \frac{1}{1+e^{-z}} \tag{17}$$

where z is the dot product of your inputs and weights:

$$z = f(w) = \vec{w} \cdot \vec{x} = \sum_{i=1}^{n} w_j \times x_i = w_1x_1 + w_2x_2 + \ldots + w_nx_n \tag{18}$$

Similar to deriving the delta rule in Section 2.1.3, the goal is to find a set of weights $\vec{w} = \{w_1, w_2, \ldots, w_n\}$ that successfully learns the binary classification problem. In other words, our prediction, $\hat{y} \in \{0, 1\}$, is equal to the target y for all inputs x. The only difference between the linear activation function and a sigmoid activation function is there is an additional transformation $f(z)$ taking place at the output layer that allows for the target predictions of the binary classification to become either 0 or 1.

Let us solve for the gradient $\frac{\partial e}{\partial w}$ to find our weight updating equation, which we can do using the gradient descent from Equation 7 and the error function defined in Equation 4. Substituting the error into the gradient we get,

$$\frac{\partial e}{\partial w} = \frac{\partial}{\partial w}[\frac{1}{2}(y - \hat{y})^2]$$

$$\frac{\partial e}{\partial w} = (y - \hat{y})(-\frac{\partial \hat{y}}{\partial w}) = -e\frac{\partial \hat{y}}{\partial w}$$

The challenge here is to find the partial derivative of the prediction, $\hat{y}$, with respect to the weights, w, since the prediction is now a sigmoid activation function. We can apply chain rule to get the following,

$$\frac{\partial \hat{y}}{\partial w} = \frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w}$$

We know from the delta rule that,

$$\frac{\partial z}{\partial w} = \frac{\partial[w_1x_1 + w_2x_2 + \ldots + w_nx_n]}{\partial w},$$

which can be simplified to,

$$\frac{\partial z}{\partial w_i} = x_i \, ,$$

for a given input i. This can be done since there is only one component of z that is a variable when considering the derivative with respect to one input. The rest of the equation is assumed as a constant in the derivative.

To find $\frac{\partial \hat{y}}{\partial z}$ we need to derive the sigmoid activation function $\hat{y}$ with respect to z ,

$$\hat{y} = f(z) = \frac{1}{1 + e^{-z}} \, ,$$

and $\frac{\partial \hat{y}}{\partial z}$ can be rewritten as the following.

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial f(z)}{\partial z} = \frac{\partial}{\partial z}[\frac{1}{1+e^{-z}}]$$
$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial}{\partial z}(1+e^{-z})^{-1}$$

Simplifying the partial derivative, we get,

$$\frac{\partial \hat{y}}{\partial z} = -1(1+e^{-z})^{-2}(e^{-z})(-1) = \frac{e^{-z}}{(1+e^{-z})^2}$$

This derivative can be further simplified to an elegant solution. We split the equation into parts,

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial z} &= \frac{e^{-z}}{(1+e^{-z})(1+e^{-z})} \\
&= \frac{e^{-z}}{1+e^{-z}} \times \frac{1}{1+e^{-z}} \\
&= \frac{e^{-z}+(1-1)}{1+e^{-z}} \times \frac{1}{1+e^{-z}} \\
&= \frac{1+e^{-z}-1}{1+e^{-z}} \times \frac{1}{1+e^{-z}} \\
\frac{\partial \hat{y}}{\partial z} &= [\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}] \times \frac{1}{1+e^{-z}}
\end{aligned}$$

By substituting $f(z) = \frac{1}{1+e^{-z}}$ the equation becomes,

$$\frac{\partial \hat{y}}{\partial z} = [\frac{1+e^{-z}}{1+e^{-z}} - f(z)] \times (f(z))\,,$$

which becomes,

$$f'(z) = (1-f(z))f(z)\,.$$

This can be substituted back into our partial derivative $\frac{\partial \hat{y}}{\partial w}$ to get,

$$\frac{\partial \hat{y}}{\partial w_i} = \frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w} = [(1-f(z))f(z)] \times x_i$$

This result can be substituted back into our partial derivative $\frac{\partial e}{\partial w}$ to get,

$$\frac{\partial e}{\partial w_i} = -e\frac{\partial \hat{y}}{\partial w_i} = -e \times (1-f(z))f(z) \times x_i$$

Finally, this can be substituted back into our gradient to get our weight updating rule,

$$w_{new} = w_{old} - \alpha \frac{\partial e}{\partial w_i}$$

$$w_{new} = w_{old} + \alpha \times e \times (1 - f(z))f(z) \times x_i \tag{19}$$

where $\alpha, e, f(z), x_i$ are constants. Equation 19 is the weight updating equation for a neural network with a sigmoid activation function. This allows us to train a neural network of n inputs to learn a set of weights to successfully predict a binary classification task.

2 Computer Vision

An infamous use case of neural networks is in the field of computer vision. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a competition hosted yearly in which researchers across more than fifty institutions build and evaluate models for image classification [24]. From 2005 to 2011 the error rate of classifying the images in the competition with the best models was approximately 26% [18].

In 2012 we saw the first use of a neural network in the field of computer vision. A team of University of Toronto students entered the competition with a neural network algorithm called 'AlexNet' that revolutionized the field. The development of neural network algorithms used in computer vision tasks brought the error rate of the ILSVRC competition models down to approximately 0%, which completely solved the problem. Neural networks in computer vision then became the golden standard of algorithms used in a variety of tasks including object detection, image classification, identity recognition, optical character recognition and self driving cars [9].

One use case we will explore in this section is Optical Character Recognition (OCR). We will use an example from the Modified National Institute of Standards and Technology dataset (MNIST) which contains over 60,000 images of 28×28 grey scaled images of hand written digits. We will train a neural network algorithm that will leverage the equations summarized in section 2 to successfully predict a digit given an image.

2.1 MNIST Dataset

The following figure is shows of 9 of the 60,000 handwritten digits that are in the MNIST Dataset. We are looking to create a model that can learn to correctly classify them.

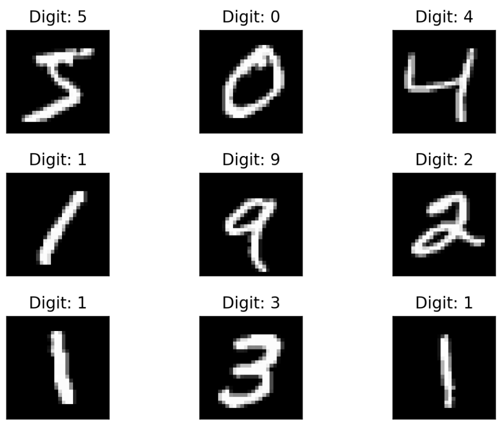


Figure 9: Example of 9 Images from MNIST Training Set

Since a neural network can only take numerical inputs, the concept of passing an image is only doable if the inputs can be represented as numerical values. The method of transcribing images to their numerical representations is called "encoding" [26]. This can be done by turning the pixels of an image into a vector of numbers between 0 and 1 within the concept of a gray-scale/monochrome image. Note that a pixel that is lighter has a numerical value closer to 0 and a darker one has a value closer to 1 create a pixel that is darker, in which 1 and 0 corresponds to a black and white pixel respectively [20][27].

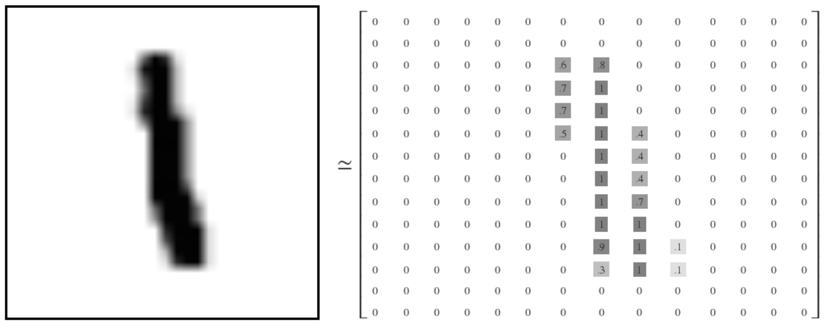


Figure 10: Example of MNIST Data Encoding

Image encodings similar to the ones in Figure 3.2 will be put into a neural network with a sigmoid activation function, described in the previous section, and will be used by the model. The MNIST dataset we are using in this paper provides the images encoded.

2.2 Neural Network Architecture

A neural network with a sigmoid activation function, is a binary classifier [12]. Recall binary classifiers have two output labels 1 and 0 (yes versus no) [8][22]. We can create a neural network algorithm that contains 10 output nodes, in which each node utilizes a sigmoid activation function. Each node corresponds to the classification of a single digit (0 to 9).

In the output layer, 10 nodes are present because the possible predictions for a digit ranges from 0 to 9, each node for one of the predictions. For example, if the model outputs a prediction of 1 at node 7 for a given image, the model suggests that the character in the image is 7. Figure 11 summarizes the neural network architecture of this model.

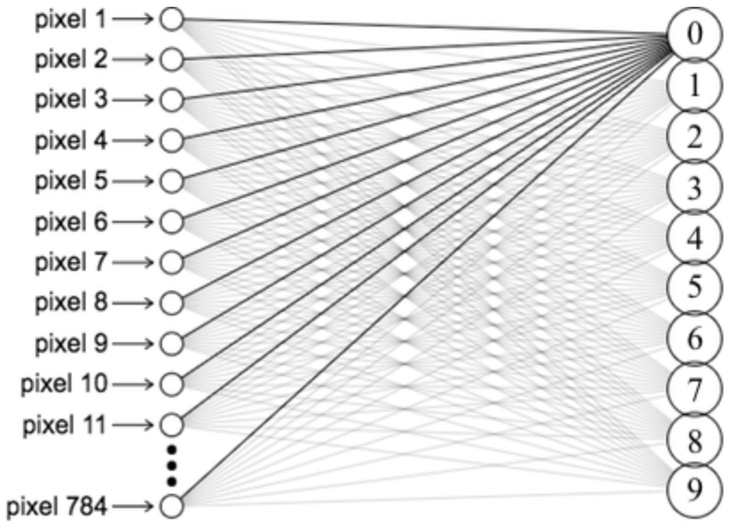


Figure 11: Neural Network Architecture

Each output follows a sigmoid activation function as defined by Equation 17 where the z is defined as,

$$z = \vec{w} \cdot \vec{x} = \sum_{i=1}^{784} w_i x_i = w_1 x_1 + w_2 x_2 + \ldots + w_{784} x_{784} \tag{20}$$

To reiterate, the goal of the network is to classify the digits presented in the images. This is done by training the set of weights using the gradient descent algorithm from Equation 19. Recall that the weights are initialized randomly, so at first, it results in very incorrect predictions. Through the use of gradient descent, the accuracy of predictions will increase, meaning the model will learn to correctly label the handwritten character over time. The accuracy can be defined as number of correct predictions over number of total predictions.

2.3 Training the Neural Network Model

We train a neural network on the architecture from Figure 3.3 on a set of 9,000 different images from the MNIST dataset. Our training accuracy and training loss can be summarized by Figure 3.4 and Figure 3.5.

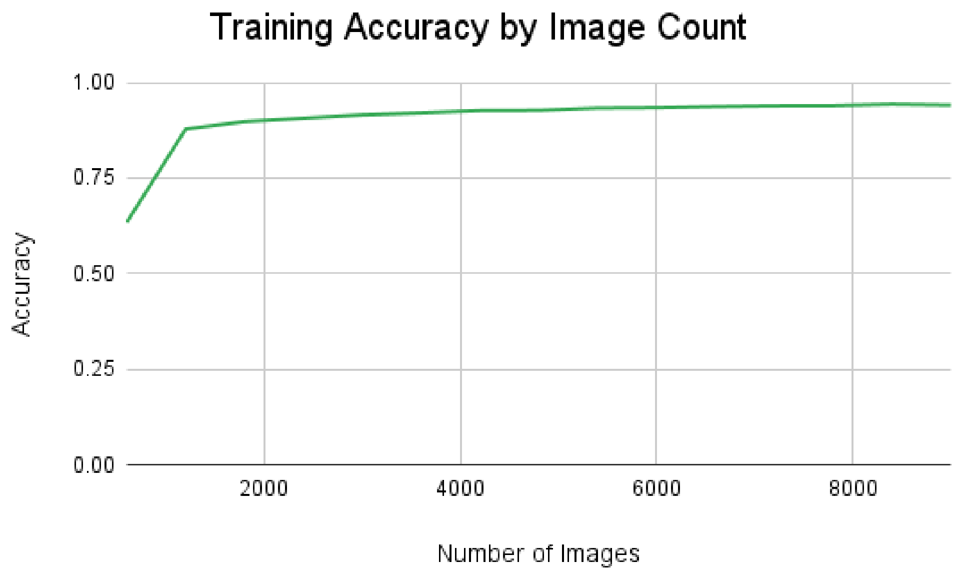


Figure 12: Training Accuracy by Image Count

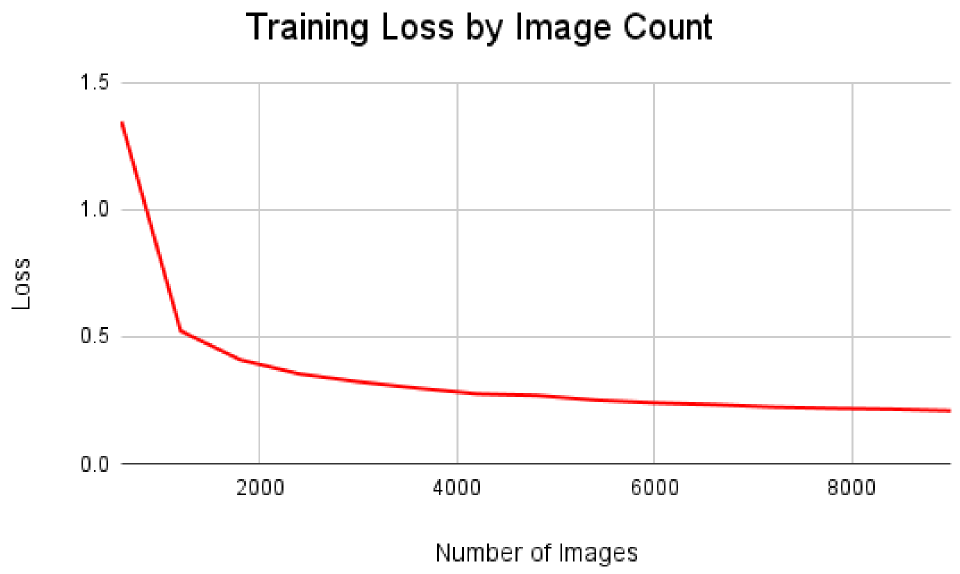


Figure 13: Training Loss by Image Count

We can see that the training loss decreases and training accuracy increases as more images are presented to the model. This is happening primarily due to the gradient descent algorithm, which focuses on tuning the model weights to minimize the error over the number of images presented. We conduct another test on the model in which we evaluate the accuracy of the model recognizing the handwritten character digits on a set of 10000 new images that the model has not seen or been training on before. We were able to successfully classify 9392 of those images correctly, giving the model a testing accuracy of 93.92%.

Overall, this model has been trained well, but we could further increase the accuracy if we were to provide it more images. However, it may not be possible to achieve 100% accuracy as some images may be impossible to classify due to poor writing quality. The same error could also happen if a human was to classify the images, which is often referred to as human error. We present an example of 9 images that were labelled correctly by the model after training.

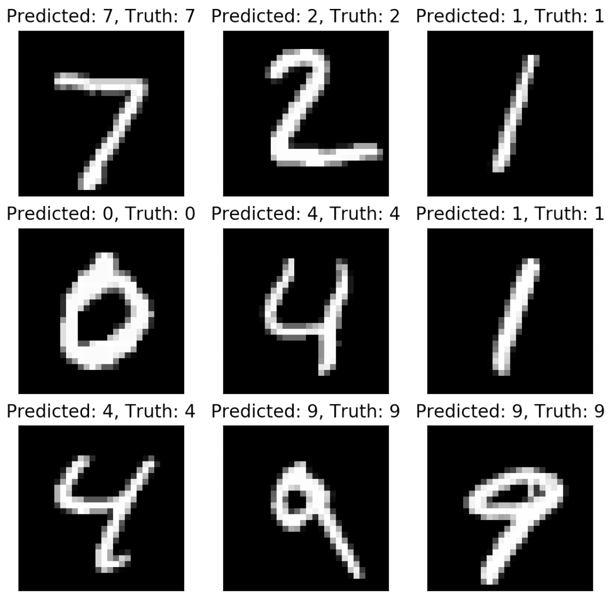


Figure 14: Correct Predictions

We also present an example of 9 images that were incorrectly labelled by the model after training.

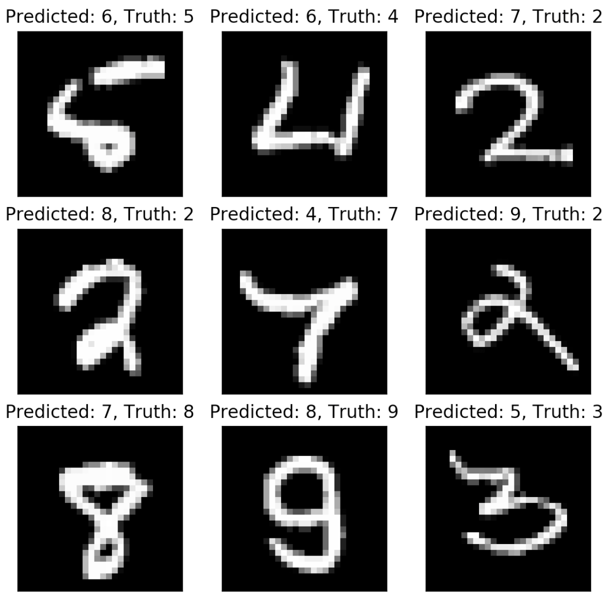


Figure 15: Incorrect Predictions

It can be seen from Figure 15 that some of these digits are poorly written, for example the 4, 5, and 7. Both humans and the model would find these images very difficult to predict.

The neural network architecture leveraged the gradient descent algorithm that we derived hitherto was proven to be very successful in this classification task. There are many other different use cases using a similar or different architecture that can be applied to various domains. For example, this architecture can be extended to a more advanced version, known as Convolutional Neural Networks, which is used by Tesla's self-driving cars [4].

3 Conclusion

The aim of this paper was to make the basic concepts of neural networks accessible to students who may not have the necessary prerequisite knowledge to learn it on their own. Using a mathematical background, the inner workings of different neural network architectures were discovered. We began by defining the simple neural network architecture, as well as its corresponding error from its predictions. We trained a model such that the errors from the predictions from the model were minimized, which was done by gradient descent. Our first use case of applying the gradient descent algorithm was to train a model in learning a linear equation. We then applied the gradient descent algorithm on the simple neural network, which gave us the delta rule. One thing we discovered in the process of training the model was that the learning rate is hypersensitive. Not all values of α solved the problem.

Next, we extended the architecture to be capable of learning problems that have binary values as their targets (yes versus no). This was done by introducing the sigmoid activation function, which ranges between 0 and 1 (1 =yes, 0 =no). Training a neural network to learn the binary classifier became increasingly difficult as applying the gradient descent algorithm was no longer trivial. With the help of calculus (primarily the chain rule), we were able to derive the weight updating equation for a binary classifier neural network model. Using this equation, we were able to create a model that can learn to classify encoded images of handwritten digits. We were able to achieve an accuracy of 93% on a set of 10,000 images from training our model.

This paper summarizes extremely fundamental concepts in the field of machine learning. For example, there has been many different activation functions that have been studied, which can apply to various different domains (Rectified Linear Unit, Tanh, etc). There are many neural network architectures, such as, Convolutional Neural Networks and Recurrent Neural Networks, that can be utilized across different applications, like computer vision and natural language processing. More recently, there has been state of the art neural network architectures, like generative adversarial networks, that enable an AI model to create pictures, videos, and sentences. Though these topics have not been discussed in this paper, we have provided a foundation for students to embark into the many different problems and domains machine learning has to offer.

Works Cited (APA)

1. Algorithmia. (2021, April 28). Introduction to Loss Functions. Retrieved September 7, 2021, from https://algorithmia.com/blog/introduction-to-loss-functions
2. Attoh-Okine, N. O. (1999). Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance. *Advances in Engineering Software, 30*(4), 291-302. doi:10.1016/s0965-9978(98)00071-4
3. Baheti, P. (n.d.). *12 types of neural networks activation functions: How to choose?* 12 Types of Neural Networks Activation Functions: How to Choose? Retrieved October 15, 2021, from https://www.v7labs.com/blog/neural-networks-activation-functions#non-linear.
4. Barla, N. (2021, August 25). Self-Driving Cars With Convolutional Neural Networks (CNN). Retrieved September 12, 2021, from https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn
5. Britannica. (n.d.). Gradient. Retrieved September 12, 2021, from https://www.britannica.com/science/gradient-mathematics
6. Brownlee, J. (2021, January 21). *How to choose an activation function for deep learning*. Machine Learning Mastery. Retrieved August 27, 2021, from https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/.
7. Brownlee, J. (2019, October 22). Loss and Loss Functions for Training Deep Learning Neural Networks. Retrieved August 28, 2021, from https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/
8. Brownlee, J. (2020, August 19). 4 Types of Classification Tasks in Machine Learning. Retrieved November 2, 2021, from https://machinelearningmastery.com/types-of-classification-in-machine-learning/
9. Chandra, R. (2020, June 19). Neural Networks: Applications in the Real World. Retrieved August 14, 2021, from https://www.upgrad.com/blog/neural-networks-applications-in-the-real-world/
10. Du, S., Lee, J., Li, H., Wang, L., & Zhai, X. (2019, May 24). *Gradient descent finds global minima of Deep Neural Networks*. PMLR. Retrieved December 12, 2021, from http://proceedings.mlr.press/v97/du19c.html.

11. Fulcher, J. A. (1990, January 1). *Foundations of Neural Networks: Guide books*. Foundations of neural networks | Guide books. Retrieved August 6, 2021, from https://dl.acm.org/doi/abs/10.5555/103991.
12. Gavrilova, Y. (2020, August 05). Machine Learning: Types of Classification Algorithms. Retrieved November 14, 2021, from https://serokell.io/blog/classification-algorithms
13. Gershenson, C. (2003, August 20). *Artificial Neural Networks for beginners*. arXiv.org. Retrieved July 29, 2021, from https://arxiv.org/abs/cs/0308031.
14. Guichard, D. (n.d.). Retrieved September 17, 2021, from https://www.whitman.edu/mathematics/calculus_online/section05.01.html#:~:text=A local maximum point on,' (x,y).&text=A local extremum is either a local minimum or a local maximum
15. Gurney, K. (1997). *An introduction to neural networks*. CRC Press.
16. Heaton, J. (2011). *Introduction to the Math of Neural Networks (Beta-1)*. Heaton Research, Inc.
17. IBM Cloud Education. (n.d.). What is Gradient Descent? Retrieved August 4, 2021, from https://www.ibm.com/cloud/learn/gradient-descent
18. IMAGENET. (n.d.). ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Retrieved August 3, 2021, from https://www.image-net.org/challenges/LSVRC/
19. Janocha, K., & Czarnecki, W. M. (2017, February 18). *On loss functions for deep neural networks in classification*. arXiv.org. Retrieved September 8, 2021, from https://arxiv.org/abs/1702.05659.
20. JavatPoint. (n.d.). DIP Types of Images - Javatpoint. Retrieved July 7, 2021, from https://www.javatpoint.com/dip-types-of-images
21. Jose, M. (1965, June 01). Non-Convex Loss Function. Retrieved July 28, 2021, from https://stats.stackexchange.com/questions/279292/non-convex-loss-function
22. Mishra, U. (n.d.). Binary and Multiclass Classification in Machine Learning. Retrieved October 15, 2021, from https://www.analyticssteps.com/blogs/binary-and-multiclass-classification-machine-learning
23. Ruder, S. (2017, June 15). *An overview of gradient descent optimization algorithms*. arXiv.org. Retrieved September 15, 2021, from https://arxiv.org/abs/1609.04747.

24. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., . . . Fei-Fei, L. (2015, April 11). ImageNet Large Scale Visual Recognition Challenge - International Journal of Computer Vision. Retrieved October 2, 2021, from https://link.springer.com/article/10.1007/s11263-015-0816-y?sa_campaign=email/event/articleAuthor/onlineFirst#citeas
25. Saha, S. (2018, December 17). *A comprehensive guide to Convolutional Neural Networks - the eli5 way*. Medium. Retrieved September 5, 2021, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.
26. Techopedia. (2011, August 18). What is Encoding? - Definition from Techopedia. Retrieved August 12, 2021, from https://www.techopedia.com/definition/948/encoding
27. TechTarget. (2010, May 20). What is grayscale? - Definition from WhatIs.com. Retrieved November 12, 2021, from https://whatis.techtarget.com/definition/grayscale
28. Tetko, I. V., Kůrková Věra, Karpov, P., & Theis, F. (2019). *Artificial Neural Networks and Machine Learning - Icann 2019: Workshop and special sessions 28th international conference on artificial neural networks, Munich, Germany, September 17-19, 2019, Proceedings* (Vol. 4). Springer International Publishing.
29. Yolcu, U., Egrioglu, E., & Aladag, C. H. (2012, December 12). A new linear & nonlinear artificial neural network model for time series forecasting. Retrieved July 16, 2021, from https://www.sciencedirect.com/science/article/abs/pii/S0167923612003557?casa_token=wvV6ONGhLQYAAAAA:yJkJrDbveOfxW-d6nEKEAeLOSNrxcVelqqjRUrY_RxcvEmZ-ObpcqlH4sWvXNpBSZdbTzDpz
30. YouTube. (2017). *Gradient descent, how neural networks learn | Chapter 2, Deep learning*. *YouTube*. Retrieved September 12, 2021, from https://www.youtube.com/watch?v=IHZwWFHWa-w.
31. Zhao, H., Gallo, O., Frosio, I., & Kautz, J. (2018, April 20). *Loss functions for neural networks for image processing*. arXiv.org. Retrieved August 15, 2021, from https://arxiv.org/abs/1511.08861.