How Artificial Intelligence Works: Using the Gradient Descent Algorithm

Rachel Glowniak

Faculty Mentor: Tina Romansky, M.S.

MA 398: Senior Seminar

Instructor: Elizabeth Tripp, Ph.D.

Fall 2023

Contents

Abstract

The recent commercialization of affordable and simplified tools has brought artificial intelligence out of the movies and research labs and into our homes and classrooms. Using Alexa, Siri, and ChatGPT has been simplified for the masses, but the algorithms behind these applications are steeped in mathematics and computer science. This paper will focus on gradient descent, an example of an optimization algorithm. The foundation of the algorithm, as well as its applications and limitations, will be explored. Finally, we will present a classic example of the gradient descent algorithm. To further analyze the method, the algorithm has been implemented in code and studied for efficiency.

1 Introduction

In this paper, we will be exploring the field of artificial intelligence. Artificial intelligence is a general term that can be applied to any system that learns.

Definition 1. ***Artificial intelligence*** *(AI) is the science and engineering of making intelligent machines, especially intelligent computer programs [2].*

We will be focusing on a specific type of AI called machine learning.

Definition 2. ***Machine learning*** *is a branch of AI and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy [1].*

Machine learning applies to any hardware or software system that continuously improves the accuracy of the results. These machine learning systems consist of algorithms and a neural network.

Definition 3. ***Neural networks*** *simulate the way the human brain works, with a large amount of linked processing nodes. Neural networks are good at recognizing patterns and play an important role in applications including natural language translation, image recognition, speech recognition, and image creation [1].*

The term neural network describes a machine learning system that models how the human brain learns, in that it makes functional connections between

various neurons. Neural networks are some of the most crucial components in any machine learning system, since they execute the action of taking in data and being able to decipher it correctly. Within the neural networks are algorithms that are used to make this learning happen. Sometimes multiple algorithms are used in one network. These algorithms contain high level mathematical features which make them complex but also allow the computer to improve its accuracy. First, the neural networks are provided with training data sets as input values for these algorithms. The accuracy of the output is measured with a cost function.

Definition 4. *The* ***cost function*** *measures the difference, or error, between the actual y and predicted y at its current position [3].*

This improves the machine learning model's efficacy by providing feedback to the model so that it can adjust the parameters to minimize the error and find the local or global minimum. The network will run through its training data each time a change is made to one of the functions parameters. After each iteration with the training data set, we will again quantify the error of our results with the cost function. Then by refining the output generating function against known input/output pairs, the function 'learns' to operate on novel data to an acceptable tolerance of error. In our example that we will see later on, we will use the cost function to help us get more accurate results from our neural network. We will also find out how to use gradient descent to help us minimize the cost function.

2 How Does A Machine Learn?

Students learning a new skill will learn from their errors and modify their process, but how does a machine learn? It does so by modifying its processing function, then evaluating its error from input to output until it can process novel data to a minimum degree of error [1]. This structure of machine learning is called a neural network. There is a set of data consisting of input and output pairs that is used for training the network. Each data point in the training data set is subtly different so that the network can be tested to see how well it understands what inputs should produce the same or different output. This is because the function will not be one to one, meaning one output could have multiple inputs. We will use this type of data to train the network, which can take multiple iterations to do. After each piece of data has ran though the network, we measure the accuracy of the output with the cost function. This means there will be a cost output for each data point in our training data set. After this, the average cost is taken to see how well the network is running. Based on these numbers, tweaks can then be made to the neural network to better its accuracy. This concept of changing parts of the neural network is machine learning, since after each iteration the average cost should be less than before [1]. This same process can be used in any machine learning system that consists of a neural network. These systems can vary in complexity, but all have the same basic mathematical components.

3 The Gradient

The gradient, symbolized by ∇, is a direction vector that is associated with a function. To find the gradient of a function, we first evaluate all of the function's partial derivatives. We then put those partial derivatives into a vector. We can see this with with a two variable example, using the function $f(x, y) = x^2 + xy$:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x}(x^2+xy) \\ \\ \frac{\partial}{\partial y}(x^2+xy) \end{bmatrix} = \begin{bmatrix} 2x+y \\ \\ x \end{bmatrix}$$

This same concept of taking the gradient of a two variable function can applied to a function with any amount of variables. The number of variables will determine the number of partial derivatives which will also determine the number of rows in our vector. The gradient vector has a very important property.

Theorem 3.1. *The gradient is the direction of steepest ascent. Thus, the negative gradient is the direction of steepest descent.*

Proof. Let $D_u f(x_0, y_0) \neq 0$, the directional derivative of $f(x, y)$ at the point (x_0, y_0) in the direction of the unit vector.

$$D_u f(x_0, y_0) = \nabla f(x_0, y_0) \cdot \overrightarrow{u} \tag{1}$$

$$= ||\nabla f(x_o, y_0)||\, ||\overrightarrow{u}|| \cos(\theta) \tag{2}$$

$$= ||\nabla f(x_o, y_0)|| \cos(\theta) \tag{3}$$

where θ is the angle between $\overrightarrow{u}$ and $\nabla f(x_0, y_0)$. Since we want the minimum, we need $\theta = \pi$, since that is where $\cos(\theta)$ is at its minimum.

$$= ||\nabla f(x_0, y_0)|| \cos(\pi) \tag{4}$$

$$= ||\nabla f(x_0, y_0)||(-1) \tag{5}$$

$$= -||\nabla f(x_0, y_0)|| \tag{6}$$

Thus, $\overrightarrow{u}$ and $\nabla F(x_0, y_0)$ are in the opposite directions, meaning the directional derivative is the minimized when $\overrightarrow{u}$ is in the direction of $-\nabla f(x_0, y_0)$. Therefore the negative gradient $(-\nabla)$ is the direction of steepest descent. □

Using the gradient to find a minimum of a multi-variable function will be necessary to minimize error in the application developed later in the paper.

3.1 Requirements

To be able to take the gradient of a function, the function has to be differentiable and convex [6]. To be differentiable, a function must be continuous as well as smooth. This means the graph can not have any discontinuities, must not have sharp corners, and can not have any vertical asymptotes. If a function has breaks, the gradient will not be able to be found, since the point at which we are taking the gradient at will have no point that is directly next to it. This will make the gradient part of the algorithm fail, yet the algorithm itself will still run.

This will be managed in the software through appropriate error handling. This is analogous to dividing by zero: the division is not flawed, the denominator is.

For gradient descent to be applicable to a multi-variable function, the function also needs to be convex.

Definition 5. *For a uni-variate function to be* ***convex****, the line segment connecting two function's points lays on or above its curve (it does not cross it) [6].*

For a single variable function to be convex, its second derivative must exist and be greater then or equal to zero. The Hessian Matrix is the multi-variable analogue of computing the second derivative [6].

Definition 6. *The* ***Hessian matrix*** *of a multi-variable function $f(x_1, x_2, \ldots, x_n)$, organizes all second partial derivatives into a matrix. It is a matrix with functions as entries. In other words, it is meant to be evaluated at some point $(a_1, a_2, \ldots, a_n)$ [8].*

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} & \cdots & \frac{\partial^2 f}{\partial x \partial n} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} & \cdots & \frac{\partial^2 f}{\partial y \partial n} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} & \cdots & \frac{\partial^2 f}{\partial z \partial n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial n \partial x} & \frac{\partial^2 f}{\partial n \partial y} & \frac{\partial^2 f}{\partial n \partial z} & \cdots & \frac{\partial^2 f}{\partial n \partial n} \end{bmatrix}$$

For a multi-variable function to be convex, its Hessian matrix must be positive semi-definite over the domain of the function [6]. This means all of the eigenvalues must be non-negative and the matrix must be symmetric [7].

Definition 7. *A matrix is* ***symmetric*** *if and only if A is $n \times n$ and $A = A^T$.*

A^T is the transpose matrix. You can define A^T by: $\left[A^T\right]_{ij} = [A]_{ji}$, meaning the ij^{ih} element of A^T is the ji^{th} element of A. This means the first column of A will become the first row of A^T, and the first row of A will become the first column of A^T. For our example, we can also note *Clairault's Theorem.*

Theorem 3.2. *Suppose that $f(x, y)$ is defined near (a, b). If the function f_{xy} and f_{yx} are continuous, then $f_{yx} = f_{xy}$*

Thus, by Clairault's theorem, the Hessian Matrix is symmetric as long as all second order partial derivative are continuous. We now have to see if all eigenvalues are non-negative. To find the eigenvalues, we have to solve the formula $det(H_f - (\lambda \cdot I_n)) = 0$, where H_f is the Hessian matrix and I is the identity matrix of the same size. Then, we calculate the determinant of that matrix, which we

see in this sequence below [8].

$$H_f - (\lambda \cdot I_n) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} & \cdots & \frac{\partial^2 f}{\partial x \partial n} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} & \cdots & \frac{\partial^2 f}{\partial y \partial n} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} & \cdots & \frac{\partial^2 f}{\partial z \partial n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial n \partial x} & \frac{\partial^2 f}{\partial n \partial y} & \frac{\partial^2 f}{\partial n \partial z} & \cdots & \frac{\partial^2 f}{\partial n \partial n} \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 & \cdots & 0 \\ 0 & \lambda & 0 & \cdots & 0 \\ 0 & 0 & \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} - \lambda & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} & \cdots & \frac{\partial^2 f}{\partial x \partial n} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} - \lambda & \frac{\partial^2 f}{\partial y \partial z} & \cdots & \frac{\partial^2 f}{\partial y \partial n} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} - \lambda & \cdots & \frac{\partial^2 f}{\partial z \partial n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial n \partial x} & \frac{\partial^2 f}{\partial n \partial y} & \frac{\partial^2 f}{\partial n \partial z} & \cdots & \frac{\partial^2 f}{\partial n \partial n} - \lambda \end{bmatrix}$$

We will demonstrate how to solve $det(h_f - \lambda I_n) = 0$ in the 3-dimensional case. Here, we have

$$det \begin{vmatrix} \frac{\partial^2 f}{\partial x^2} - \lambda & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} - \lambda & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} - \lambda \end{vmatrix} = 0$$

Below, we demonstrate the process of calculating the determinant using the first column. Note that we may use any column or row in this process and will arrive

with the same answer.

$$\left(\frac{\partial^2 f}{\partial x^2}-\lambda\right)\begin{vmatrix}\frac{\partial^2 f}{\partial y^2}-\lambda & \frac{\partial^2 f}{\partial y\partial z}\\ \frac{\partial^2 f}{\partial z\partial y} & \frac{\partial^2 f}{\partial z^2}-\lambda\end{vmatrix}-\frac{\partial^2 f}{\partial y\partial x}\begin{vmatrix}\frac{\partial^2 f}{\partial x\partial y} & \frac{\partial^2 f}{\partial x\partial z}\\ \frac{\partial^2 f}{\partial z\partial y} & \frac{\partial^2 f}{\partial z^2}-\lambda\end{vmatrix}+\frac{\partial^2 f}{\partial z\partial x}\begin{vmatrix}\frac{\partial^2 f}{\partial x\partial y} & \frac{\partial^2 f}{\partial x\partial z}\\ \frac{\partial^2 f}{\partial y^2}-\lambda & \frac{\partial^2 f}{\partial y\partial z}\end{vmatrix}=0$$

$$\begin{gathered}\left(\tfrac{\partial^2 f}{\partial x^2}-\lambda\right)\left[\left(\tfrac{\partial^2 f}{\partial y^2}-\lambda\right)\left(\tfrac{\partial^2 f}{\partial z^2}-\lambda\right)-\left(\tfrac{\partial^2 f}{\partial y\partial z}\right)\left(\tfrac{\partial^2 f}{\partial z\partial y}\right)\right]-\\ \tfrac{\partial^2 f}{\partial y\partial x}\left[\left(\tfrac{\partial^2 f}{\partial x\partial y}\right)\left(\tfrac{\partial^2 f}{\partial z^2}-\lambda\right)-\left(\tfrac{\partial^2 f}{\partial x\partial z}\right)\left(\tfrac{\partial^2 f}{\partial y\partial z}\right)\right]+\\ \tfrac{\partial^2 f}{\partial z\partial x}\left[\left(\tfrac{\partial^2 f}{\partial x\partial y}\right)\left(\tfrac{\partial^2 f}{\partial y\partial z}\right)-\left(\tfrac{\partial^2 f}{\partial x\partial z}\right)(\,\tfrac{\partial^2 f}{\partial y^2}-\lambda\right)\Big]=0\end{gathered}$$

We then can then simplify to solve for our eigenvalues, which will be our three λ values. Note that the number of eigenvalues is determined by the dimension of the matrix. When we solve for the eigenvalues, if they are all non-negative, then that covers the first of the two parts of being positive semi-definite [7]. Since the dimension of our matrix was $n \times n$ and it satisfies $A = A^T$, the matrix is symmetric, so this covers the second part of being positive semi-definite [7]. In this case, our function will be convex. This process of finding the eigenvalues will be the same for all matrices as long as they are $n \times n$. If our function is both differentiable and convex, then we are able to use the gradient descent algorithm [6].

3.2 Gradient Descent

We will be using gradient decent to help us find the local minimum of a function that has multiple inputs [3]. For single variable functions, this is quite simple: we take the derivative and set it equal to zero. However, when we add more variables, finding the minimum gets more complex. In this case, we also have to consider a multidimensional direction instead of just left and right. To do this, by **Theorem 3.1**, we have to use the negative gradient of the function [5]. We do this by simply multiplying our vector of partial derivatives by a scalar of negative one.

To describe how gradient descent works, we will consider the single variable case first. For single variable functions, the slope of the tangent line can be positive, negative, zero, or undefined. For our purpose, we will just be looking at the cases where the slope of the tangent line is positive or negative. If we happen to pick a point where the slope is zero, we can stop there, and that point is our local minima. If we choose a point that is undefined, we select another random starting point. When the slope is positive at a point, if we want to approach the minimum of the function, we will move slightly to the left of our starting point; if it is negative, we will move to the right. We iterate the process of finding the direction of the slope and moving the point in the corresponding direction [5]. Eventually, we will approach a local minimum of the function where the slope is 0.

For multi-variable functions, we will apply a similar process. We will start off by picking some arbitrary point on the function. We will then take the negative gradient of the function. The negative gradient of the function, determines the direction to move towards a next point. Unlike the single variable function, we can't just move our point to the left or right. Instead, we will add the vector that contains our arbitrary point to the vector that contains the negative gradient of our function at that arbitrary point. This will give us a new point on the function that is slightly closer to the nearest local minimum [5]. Using the same two variable function example seen earlier on page 4 and an arbitrary starting point (a, b), we get

$$-\nabla f(x,y) = -1 \begin{bmatrix} \frac{\partial}{\partial x}(x^2+xy) \\ \frac{\partial}{\partial y}(x^2+xy) \end{bmatrix} = -1 \begin{bmatrix} 2x+y \\ x \end{bmatrix} = \begin{bmatrix} -2x-y \\ -x \end{bmatrix}$$

This gives us the direction of steepest descent at a general point (x, y). We will now plug in our point (a, b). This will give us a specific direction:

$$-\nabla f(a,b) = \begin{bmatrix} -2a-b \\ -a \end{bmatrix}$$

Next, we will add our point (a, b) to this directional value. This will give us a new point on the function, that is closer to the local minimum.

$$\begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} -2a-b \\ -a \end{bmatrix} = \begin{bmatrix} a+(-2a-b) \\ -b+(-a) \end{bmatrix}$$

We will repeat this process until our directional value is equal to the zero vector, $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$. This will mean we have found a local minimum of the function. The local minimum will be at the final point (a, b) at which the direction vector became the zero vector.

4 Example of the Gradient Descent Algorithm

Let us consider an example where we want to read image files of hand written numbers 0-9 and use the gradient descent algorithm to find out what ways to manipulate the neural network so that the symbolic number in text is the same as the number that was hand written [9]. Each number has hundreds of different hand written versions. These different versions will be the training data. We will run all of our training data through the network after each iteration of the algorithm. We will use the gradient descent algorithm to help us minimize the cost function, which is the goal for this algorithm. There is also an example of the code in the appendix, which can be used for this application to create a neural network.

4.1 How Does This Work?

The numbers 0-9 are hand written, and they are each put on a 28 by 28 grid [9]. So there are 784 cells in total. Each square on the grid is called a neuron. Each neuron is given a value from 0-1 based on how shaded the neuron is [9]. For this example, we will be looking at one version of a hand written number.

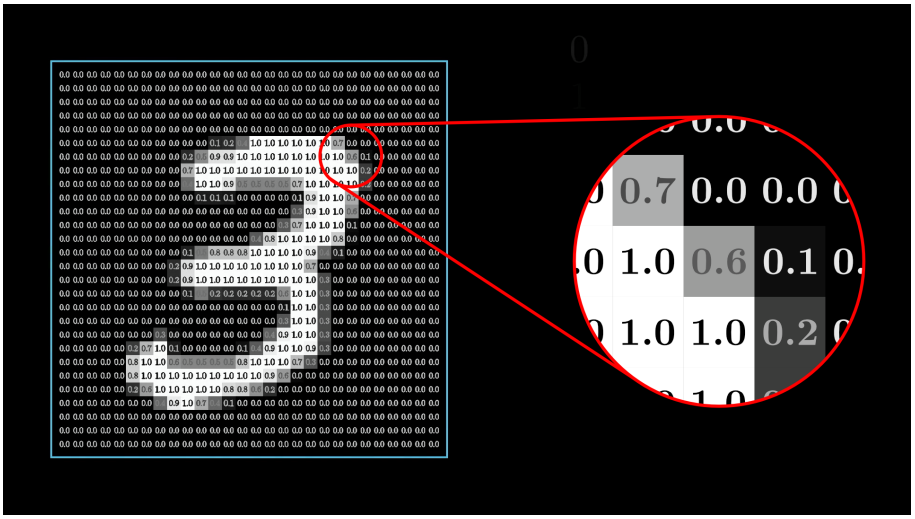

Figure 1: Grid with Shadedness Values [10]

The 784 neurons each with a given value of shadedness are then put into a vector and make up the first layer of the neural network. Each neuron from the first layer is connected to each of an arbitrary amount of neurons in the next layer, and so on for each layer after, until you get to the last layer, which is the output layer [9]. This layer will have 10 neurons, one for each possible hand written number. For this example, we will have the first layer consist of the 784 neurons,

then the second and third layer will both have 16 neurons, and then the final output layer will have 10 neurons [9].

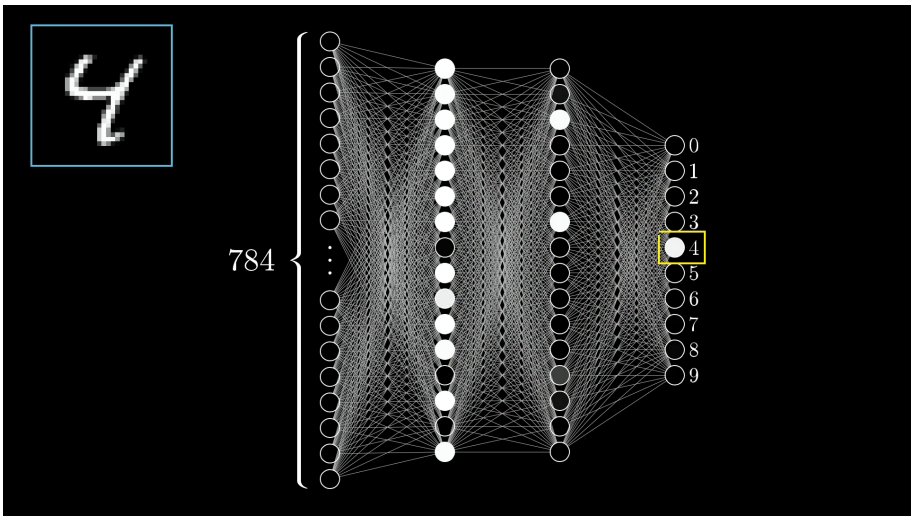

Figure 2: Trained Neural Network [9]

In Figure 2, the network has produced a perfect result. It gave the value 1 to the neuron associated with the number 4 and the value 0 to all other neurons. We will use the function $f(\vec{x}) = \sigma(x_1w_1 + x_2w_2 + x_3w_3 + ... + x_nw_n + b)$ to get from layer to layer [9]. This will give us the value for each neuron for every layer after the first layer, since the values for each neuron in the first layer is taken directly from the grid.

Here, x_i are the variables representing the values at each neuron in the prior layer, w_i are the weights, and b is the bias. Each neuron will have a version of this function associated with it, with different values for each weight and bias.

This means we will have 42 formulas to compute, since we have two layers of 16 and one layer of 10. To figure out what weights and biases to use in each of our formulas, we first need to find out how many different weights and biases we have. To find the amount of weights, we will compute the formula below where $n = 4$ is the number of layers in this network:

$$\sum_{i=1}^{n-1} \textit{(Number of Neurons in layer i)(Number of Neurons in next layer)}$$

$$= (784 \times 16) + (16 \times 16) + (16 \times 10)$$

$$= 12,544 + 256 + 160$$

$$= 12,960 \text{ weights}$$

To find the number of biases, we will add how many neurons there are in each layer except the input layer:

$$16 + 16 + 10 = 42 \text{ biases}$$

We will then add the number of weights and the number of biases together to get the total number of parameters:

$$12,960 + 42 = 13,002 \text{ parameters}$$

The bias is just some constant that we add on to the end of our function. The weights, w_i, are how much of that specific variable we want to contribute to the

function. This is why we have to change these weights for the network to run more accurately. These weights determine how much of the prior neuron we want for the neuron in the next layer. The sigma function returns a value between 0 and 1 and is shown below:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Theorem 4.1. *For all $x \in \mathbb{R}$, the sigma function will always be greater then zero and less then one.*

Proof. We know that e^{-x} will always be greater then zero. This ensures that $\frac{1}{1+e^{-x}}$ will always be positive. We also know that, since e^{-x} is always greater then zero, $1 + e^{-x}$ is always greater then 1. This means $\frac{1}{1+e^{-x}}$ will always be less then one. Therefore:

$$0 < \frac{1}{1 + e^{-x}} < 1$$

□

4.2 Applying The Cost Function

We start training the neural network by randomizing the weights and biases for each function. As expected, when we run the program, it performs poorly, as shown in Figure 3. The output gives us a combination of different values that are associated with each neuron, instead of one output with the value of 1 and the rest 0, as we saw with the trained network in Figure 2.

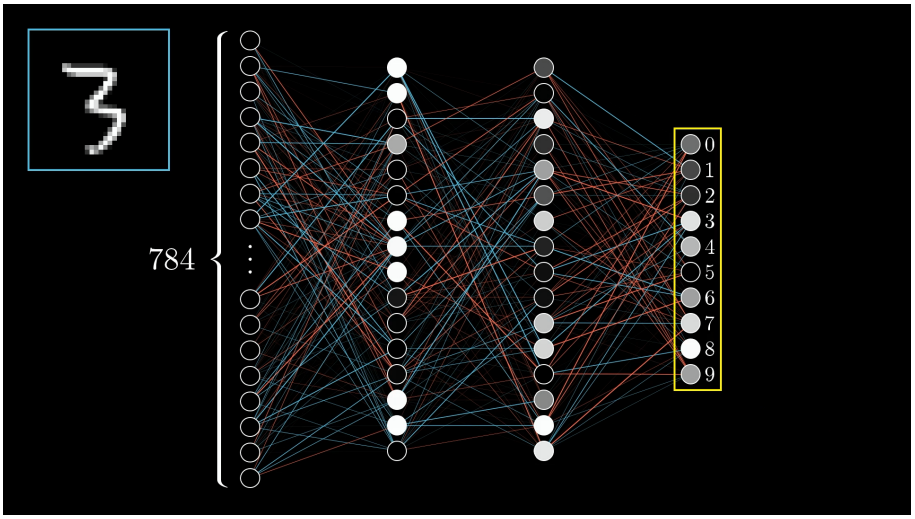

Figure 3: Untrained Neural Network [9]

We will fix this problem by using the cost function [9]. Once we get all ten outputs from the formula, we subtract each predicted output value from its actual output value. Then, we will square each difference and add them all together. For the predicted output value, we use 1 for the neuron associated with the hand written number we expected and 0 for all the other 9 values. For example, if 3 was the correct output for a handwritten number, the cost function would be as shown below, where $f(x)_n$ represents the actual output value from each neuron in the output layer:

$$c(x) = (f(x)_0-0)^2+(f(x)_1-0)^2+(f(x)_2-0)^2+(\mathbf{f(x)_3 - 1})^\mathbf{2}+(f(x)_4-0)^2+(f(x)_5-0)^2+$$

$$(f(x)_6 - 0)^2 + (f(x)_7 - 0)^2 + (f(x)_8 - 0)^2 + (f(x)_9 - 0)^2$$

When the cost $c(x)$ is small, the program is more accurate; when it is large, it is less effective. Our end goal is to get $c(x) = 0$. We will repeat the process of running a new training image though our network, then computing the cost function, for each different hand written number that we have in our training data. We will then find the average, where n is the amount of handwritten images in the training set and $c(x)_i$ is the cost of each input/output pair i from the training data.

$$\text{Average Cost} = \frac{c(x)_1 + c(x)_2 + c(x)_3 + ... + c(x)_m}{m}$$

We want to try and find a way to minimize this cost function so the neural network can run more accurately. We will do this by changing the parameters in each function. [9].

4.3 Using Gradient Descent

To recap, we have 13,002 parameters and we are trying to find the best value for these parameters so that we minimize the cost function. In a single variable function, we can minimize a function by finding the local minimum with the first derivative test. Recall that this is similar in multi-variable functions, except we have to use the negative gradient:

$$\nabla f(x, y) \to -\nabla f(x, y)$$

To use the gradient in this application, we will first set up a column vector with all of our initial randomized parameter values in it. $\overrightarrow{W}$ will represent this column vector and w_i will represent the i^{th} parameter value: [9].

$$\overrightarrow{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{13002} \end{bmatrix}$$

Next, we will compute the gradient of the cost function, C, as a function of our 13002 weights and biases:

$$\nabla C(y_1, y_2, y_3, ..., y_{13002}) = \begin{bmatrix} \frac{\partial}{\partial y_1} C(y_1, y_2, y_3, ..., y_{13002}) \\ \frac{\partial}{\partial y_2} C(y_1, y_2, y_3, ..., y_{13002}) \\ \vdots \\ \frac{\partial}{\partial y_{13002}} C(y_1, y_2, y_3, ..., y_{13002}) \end{bmatrix}$$

We will then plug in our parameters (w) into ∇C. The value we get will be represented by c_i:

$$\nabla C(\overrightarrow{W}) = \begin{bmatrix} \frac{\partial C}{\partial w_1}(w_1) \\ \frac{\partial C}{\partial w_2}(w_2) \\ \vdots \\ \frac{\partial C}{\partial w_{13002}}(w_{13002}) \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{13002} \end{bmatrix}$$

Now, since we need the negative gradient, we will multiply the vector by the scalar -1:

$$-\nabla C(\overrightarrow{W}) = -1 \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{13002} \end{bmatrix} = \begin{bmatrix} -c_1 \\ -c_2 \\ \vdots \\ -c_{13002} \end{bmatrix}$$

We will then add the vectors $\overrightarrow{W}$ and $-\nabla C(\overrightarrow{W})$ together. This will give us new parameters represented by n_i,

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{13002} \end{bmatrix} + \begin{bmatrix} -c_1 \\ -c_2 \\ \vdots \\ -c_{13002} \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_{13002} \end{bmatrix}$$

We will plug these new parameters into the functions in the neural network, and run all of the training data though the neural network again using these new parameter vales [9]. We will compute the value of the cost function after each input from the training data set. After finding the cost from each input, we can then compute the average cost and can compare this new average cost from the new updated neural network to the average cost with the previous parameters. By using gradient descent, the average cost should decrease from the first attempt with random parameters. After seeing the new average cost, if the average cost has not achieved a specified standard, then we will repeat the process: compute

the new negative gradient and add it to the current values of the parameters to get the new, updated parameters to use for the network. This process will be repeated until the average cost function is at the desired value for accuracy.

5 Conclusion

The gradient descent algorithm does have some flaws that are worth mentioning. Sometimes, a function may not meet the requirements needed to be able to calculate the gradient. This means some alternative method would have to be used to optimize the cost function and improve the efficiency of the neural network. One way to do this is by using back propagation [9]. This is a different algorithm that does not use the gradient that can be used to train a neural network. Another flaw with gradient descent is that the processing time can be high, particularly when working with high degree functions, as in our example. This is why there are different variations of gradient descent including batch, mini batch, and stochastic [3]. In our example, we used mini batch gradient descent. This is because we updated our parameters after we ran through all the training data, instead of after each input/output pair in our training data set. If we did that, it would be considered stochastic gradient descent [4]. The gradient descent algorithm is one very small but very important piece of what goes into creating a machine learning system.

References

[1] What is machine learning?. IBM. (n.d.). https://www.ibm.com/topics/machine-learning

[2] What is Artificial Intelligence (AI)?. IBM. (n.d.-a). https://www.ibm.com/topics/artificial-intelligence

[3] What is gradient descent?. IBM. (n.d.). https://www.ibm.com/topics/gradient-descent

[4] Gradient descent in Machine Learning: A basic introduction. Built In. (n.d.). https://builtin.com/data-science/gradient-descent

[5] Khan Academy. (n.d.). Gradient descent (article). Khan Academy. https:// www.khanacademy.org/ math/multivariable-calculus/applications-of- multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent

[6] Kwiatkowski, R. (2023, October 11). Gradient descent algorithm a deep dive. Medium. https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

[7] https://www.cse.iitk.ac.in/users/rmittal/prev_course/s14/notes/lec11.pdf. (n.d.).

[8] Khan Academy. (n.d.-b). The hessian matrix multivariable calculus (article). Khan Academy. https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/ quadratic-approximations /a/the-hessian

[9] YouTube. (2017). YouTube. Retrieved December 3, 2023, from https://www.youtube.com/watch?v=IHZwWFHWa-w.

[10] But what is a neural network?.3Blue1Brown.(n.d.).https://www.3 blue1brown .com/lessons/neural-networks

[11] Stewart, J., Cleeg, D., & Watson, S. (2020). Calculus. Cengage Learning.

[12] Chatgpt. (n.d.). https://chat.openai.com

Appendix

Following is a software implementation of a neural network modeled off of of the example detailed in the paper [12]. It consist of an input layer with 784 nodes, two interior layers with 16 nodes, and an output layer with 10 nodes. The program is trained using the MNIST (Modified National Institute of Standards and Technology) data set, which contains handwritten image files of the numbers 0-9. It is able to take in those files, as well as identify the hand written digits, and classify them. This code will also asses the performance, quantifying its accuracy with a metric labeled "Test Accuracy".

```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "code",
      "execution_count": 2,
      "metadata": {
        "id": "oqDNHWgXcVu_"
      },
      "outputs": [],
      "source": []
    },
    {
      "cell_type": "code",
      "source": [
        "import tensorflow as tf\n",
        "from tensorflow.keras.datasets import mnist\n",
        "from tensorflow.keras import layers, models\n",
        "\n",
        "# Load the MNIST dataset\n",
        "(x_train, y_train), (x_test, y_test) = mnist.load_data()\n",
        "\n",
        "# Preprocess the data\n",
        "x_train = x_train.reshape((60000, 784)).astype('float32') /
255\n",
        "x_test = x_test.reshape((10000, 784)).astype('float32') /
255\n",
        "\n",
        "# Define the architecture of the neural network with sigmoid
activation function\n",
        "model = models.Sequential([\n",
        "    layers.Dense(128, activation='sigmoid',
input_shape=(784,)),\n",
        "    layers.Dense(16, activation='sigmoid'),\n",
        "    layers.Dense(16, activation='sigmoid'),\n",
        "    layers.Dense(10, activation='softmax')\n",
        "])\n",
        "\n",
        "# Compile the model\n",
        "model.compile(optimizer='adam',\n",
        "              loss='sparse_categorical_crossentropy',\n",
```

```
        "              metrics=['accuracy'])\n",
        "\n",
        "# Train the model with the MNIST training data using batch
gradient descent\n",
        "batch_size = 32\n",
        "epochs = 5\n",
        "steps_per_epoch = x_train.shape[0] // batch_size\n",
        "model.fit(x_train, y_train, epochs=epochs,
batch_size=batch_size, steps_per_epoch=steps_per_epoch, verbose=1)\n",
        "\n",
        "# Evaluate the model on the MNIST test data\n",
        "test_loss, test_accuracy = model.evaluate(x_test, y_test,
verbose=0)\n",
        "print(\"Test Accuracy:\", test_accuracy)"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "PSC_sjfDdBL_",
        "outputId": "670e3941-7106-46fe-a1cb-808e2a950c75"
      },
      "execution_count": 3,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Epoch 1/5\n",
            "1875/1875 [==============================] - 8s 4ms/step -
loss: 1.0070 - accuracy: 0.7843\n",
            "Epoch 2/5\n",
            "1875/1875 [==============================] - 8s 4ms/step -
loss: 0.2684 - accuracy: 0.9380\n",
            "Epoch 3/5\n",
            "1875/1875 [==============================] - 8s 4ms/step -
loss: 0.1673 - accuracy: 0.9576\n",
            "Epoch 4/5\n",
            "1875/1875 [==============================] - 6s 3ms/step -
loss: 0.1230 - accuracy: 0.9682\n",
            "Epoch 5/5\n",
            "1875/1875 [==============================] - 8s 4ms/step -
loss: 0.0955 - accuracy: 0.9757\n",
            "Test Accuracy: 0.9679999947547913\n"
          ]
        }
      ]
    }
  ]
}
```


Figure 4: Example Code