Homework 5 Final Draft

Dante Criscio

December 14, 2021

1 Neural Networks

Neural networks consist of many **artificial neurons**, so to understand what a neural network is, it is first important to understand these artificial neurons, how they work, and how they are connected. Neurons output a single, real value between 0 and 1. They take, as an input, the output of several other neurons. The way in which these neurons compute their output from all the inputs that they are receiving can seem complicated at first, but is really meant to simply model a somewhat linear system.

Let $x = (x_1, x_2, \ldots, x_n)$ be an n dimensional vector which represents the inputs to a given neuron. Again, each x_i is simply a real value between 0 and 1. The first thing that we do is compute a linear combination of all these x_i values, so let $w = (w_1, w_2, \ldots, w_n)$ be an n dimensional vector, where each $w_i \in \mathbb{Z}$. We first compute the dot product, $(w \cdot x)$. Each w_i is known as a **weight**, and this dot product is meant to indicate the idea that our neuron can weigh certain inputs above others.

We also introduce a **bias** value b, where $b \in \mathbb{Z}$. We want this b value to reflect how "easy" it is for our neuron to output a larger value (closer to 1), where very positive values of b make it easier to do this and very negative values

of b make it more difficult. So we then compute $(w \cdot x) + b$. Lastly, we wanted our neuron to output a value strictly between 0 and 1, but at the moment it can output any real value. So we, compute $\sigma((w \cdot x) + b)$, where $\sigma(z)$ is known as the sigmoid function and defined by:

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

Its definition is less important than its graphical behavior, which is designed to squeeze all real numbers between an output range of 0 and 1.

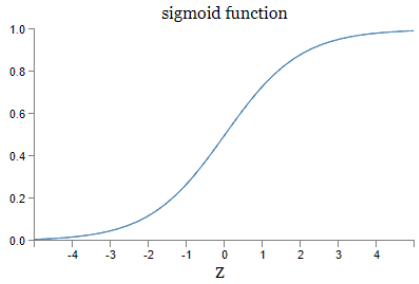

Figure 1: Sigmoid Function. Source: [1]

So, we can finally say that our neurons simply compute $\sigma((w \cdot x) + b)$. Our neural network is then a series of neurons which are broken down into multiple layers, where each neuron takes inputs from every neuron in the layer before it.

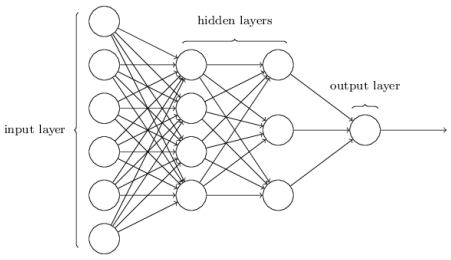

Figure 2: Neural Network. Source: [1]

If the problem we wanted this network to solve was to return a 1 if an image

was of a dog and 0 if the image was not a dog, the output layer would be just that, the input layer would represent the values of each pixel in the image, and the hidden layers somehow "calculate" our output. These neural networks often include far more neurons per level and can include many more layers. Training a neural network, simply refers to selecting values of w and b for each neuron such that the network does what we want it to do [1].

2 Gradient Descent

In general, neural networks are trained by having a large amount of training data which is fed to the network. The network learns from this by gradually tweaking its w and b values until it does its job very well with the training data. First, we define a **cost function**, which is meant to represent how far off the neural network was from returning the correct outputs for the training data:

$$C(w, b) = \frac{1}{2n} \sum_{i=1}^{n} ||y_i - a_i||^2.$$

In the equation, w and b refer to large vectors that contain all the weights and biases of the network and n is the number of training examples. y_i is our network's output for a given training example, and a_i is the correct output for that example (note that these are both vectors because the output layer can have any number of output neurons).

The goal of training is to find a way to minimize the cost function. It is important to note that our neurons simply represent continuous functions, so the cost function itself is a continuous function of many inputs. Let v be a vector which represents all of these inputs (weights and biases) such that $v = (v_1, v_2, \ldots, v_m)$. Minimizing the cost function is the same as finding a local minimum, which we can do using a technique known as **gradient descent**.

Let ∇C denote the gradient vector of the cost function, where $\nabla C = \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)$. Calculus tells us that, for a small change to our inputs, known as Δv, the resulting change to the output, known as ΔC, can be approximated as $\Delta C \approx (\nabla C \cdot \Delta v)$. Now, gradient descent refers to a technique where we choose $\Delta v = -\eta \nabla C$, where η is just a small, positive integer which guarantees that our value for Δv will be small enough that our approximation holds true.

We choose Δv in this way because we then have that $\Delta C \approx -\eta ||\nabla C||^2$, which guarantees the change to our cost function will be negative. So, if we repeatedly choose Δv in this way, our cost function will decrease until we find a minimum. In fact, it can be proven that $\Delta v = -\eta \nabla C$ is the value of Δv which decreases C as much as possible, using the Cauchy-Schwarz Inequality [1].

3 Cauchy-Schwarz Inequality

For those unfamiliar, the Cauchy-Schwarz Inequality is crucial theorem in linear algebra.

Theorem 1 (Cauchy-Schwarz Inequality)**.** *For any vectors u and v which belong to some inner product space:*

$$(u \cdot v)^2 \leq (u \cdot u)(v \cdot v).$$

Note that I chose to show the theorem using the dot product, but it does apply to any inner product space.

Taking the square root of both sides we find a corollary to this theorem.

Corollary 1. *For the same vectors u and v:*

$$|(u \cdot v)| \leq ||u||||v||,$$

where $||u||$ denotes the norm of u. Additionally, we only have equality when u and v are linearly dependent [2].

Now, we can use this to prove that the optimal choice for minimizing $\Delta C \approx (\nabla C \cdot \Delta v)$ is $\Delta v = -\eta \nabla C$. First note that η was only introduced to guarantee that Δv is a small value, specifically such that $||\Delta v|| = \epsilon$, where ϵ is known as the size constraint. So, when choosing the vector Δv we essentially have no control over the size of the vector, we can only choose the direction of the vector.

Next, recognize that we are trying to minimize $(\nabla C \cdot \Delta v)$. By the corollary to the Cauchy-Schwartz Inequality, the maximum of this value can be found when Δv is linearly dependent to ∇C. But, we previously said that we can only choose the direction of the vector, so we actually have a maximum when $\Delta v = \eta \nabla C$. It then follows that we have a minimum when $\Delta v = -\eta \nabla C$.

In conclusion, the Cauchy-Schwarz Inequality can be used to prove that a certain choice for Δv is optimal for reducing the cost function of a neural network. This is a very important conclusion in the world of machine learning, because neural networks can generally take a long time to train. As such, it is important to have confidence that the mathematics behind each step of the process is sound.

References

[1] M. A. Nielsen, *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/: Determination Press, 2015.

[2] W. contributors, "Cauchy–schwarz inequality," 2021, last accessed 08 October 2021. [Online]. Available: https://en.wikipedia.org/wiki/Cauchy%E2%80%93Schwarz_inequality