MATHEMATICS AND MACHINE LEARNING

ARDEN RASMUSSEN

Abstract. Machine Learning is a rapidly expanding subject of much research. There are many different approaches to the subject of machine learning, and many more which are still being researched. This paper provides a short introduction to two methods that can be utilized for machine learning. We will focus specifically on the mathematics of these methods, and how they are can be used for construction artificial intelligence. We will focus on the methods of *neural networks* and *genetic algorithms*. These two methods can be used in complement together to develop advanced algorithms.

1. Introduction

2. Neural Networks

A neural network is a computer model that is constructed to emulate the biological process of neural networks in brains. These models have been a key point for a lot of modern research, and breakthroughs in artificial intelligence. This section will attempt to construct an understanding of a basic type of neural network. This type of network is formally classified as a *Multi Layer Perceptron*.

2.1. Structure.

2.1.1. *Neuron.* The most basic component of a neural network is the *neuron*. A single neuron takes input from many sources and produces a single output value. A single neuron takes inputs from other neurons or from external sources. For each input value the neuron assigns a weight associated with how important that value is relative to the other inputs. Then the neuron applies a function f, to the weighted sum of the input values.

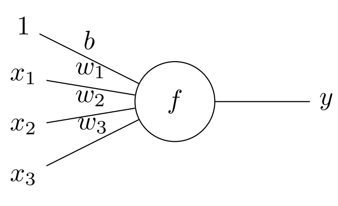

Figure 1. A single neuron acting on inputs with provided weights w_i.

In figure 1, we see that the neuron takes in four inputs, $x_1, \ldots, x_3, 1$. Then for each input there is an associated "importance" to that input with respect to the other inputs $w_1, \ldots, w_3, b$. This value is the weight of that input. The neuron takes the inputs and the weights of the inputs and applies the function f on the weighted sum of the inputs

$$f(w_1x_1 + w_2x_2 + w_3x_3 + b).$$

We can then generalize this formula for any number of input neurons to be

$$f\left(\sum_{i=1}^{N} w_i x_i + b\right). \tag{1}$$

The output of this neuron is either passed along to other neurons as inputs, or is the output of our network, and is given to the user.

The function f that the neuron applies to the weighted sum is called the *activation function*. The activation function is a non-linear function used to introduce non-linearity into the neural network. Without the non-linearity then the network would be just a bunch of linear operations, which could be represented as a single matrix, by the introduction of the non-linearity we enforce that there must be many steps. This will increase the ability of the network to do more complex operations in the future. We will go into further detail on activation functions in a different section.

There are three main *types* of neurons that we will consider.

Input Neurons: Input neurons take information from external inputs, such as image data, and then passes that data on to the other neurons in the network. The input neurons do no computation, they just pass on their information to later neurons.

Hidden Neurons: Hidden neurons have no connection directly to input or output data. They only have what is provided to them from their previous layer, and preform the computation that was explained in section **??**. Then the

Date: April 2, 2019.

output value is passed on, either to more hidden neurons or to the output neurons.

Output Neurons: The output neurons are responsible to taking the values passed to them by the previous layer of neurons and processing the values into an understandable interpretation, such as a probability. They then return that data to the user.

We will uses these three types of neurons to construct the neural network.

2.1.2. *Activation Function.* Activation functions are a method to cause some non-linearity. This importance comes into play when we have multiple neurons connecting into one another. Since each neuron individually is linear, then this connection of neurons will also be linear. That would mean that it could all be modeled as a single neuron. This is the reason as to why we need to impose the activation functions, to cause non-linearity in between the different neurons, so that the function as a whole is not linear. This means that each neuron can be trained to be important in the network.

There are a number of common functions that are used for the activation functions. *Sigmoid*, *tanh*, *ReLU*, and other variations on *ReLU*. We will describe each of these possible activation functions in more detail below. Through usage, it has been found that a good default is to use *ReLU*.

Sigmoid: The sigmoid activation function is defined as

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

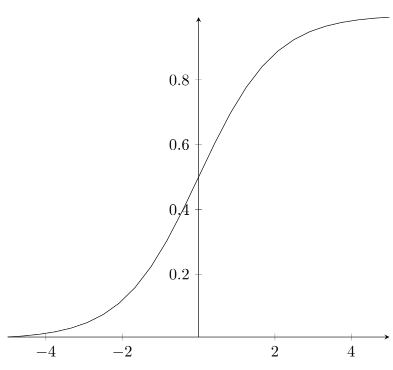

FIGURE 2. $\frac{1}{1+e^{-x}}$

The sigmoid function takes any value input to it and squashes it into the range of $(0, 1)$. However, there are some issues that arise with the sigmoid activation function. If the input is small or large, then the output becomes saturated, and the gradient of the function at those points is zero. This becomes an issue later when using back propagation. The other issue is that the output is not centered at zero, this will again cause issues later with back propagation.

tanh: The tanh activation function is similar to the sigmoid function, but is constructed to be zero centered. It is defined as

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

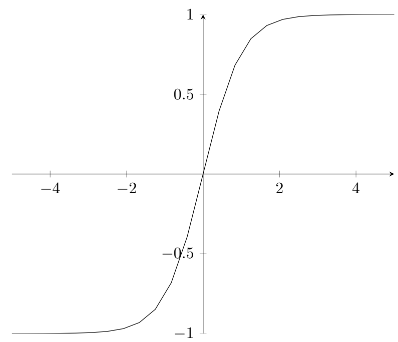

FIGURE 3. $\frac{e^x-e^{-x}}{e^x+e^{-x}}$

The tanh function has the same issue as with the sigmoid function in that it can become saturated at large or small values, thus taking the gradient to zero. However, this does solve the issue with the zero centering. The output of this function will be zero centered.

ReLU: The ReLU is an acronym for Rectified linear unit. This is commonly the default activation function. It is defined as

$$f(x) = \max(0, x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}.$$

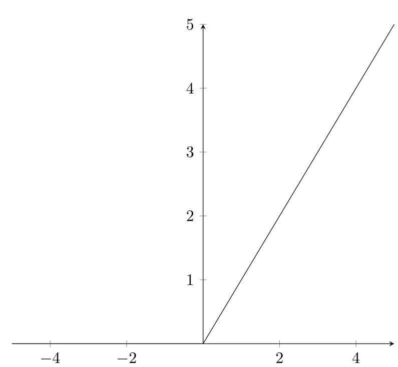

FIGURE 4. $\max(0, x)$

The ReLU function runs into many of the same issues as the other activation functions. First it is possible to the neuron to "die" if the input is less than zero, then the derivative will be zero, and that neuron is effectively dead. This can be very bad in training. It is also not zero centered, just like the sigmoid function. However both of these issues are usually left to the neural network to deal with, then the speed increase that is provided by the simplicity of the function makes this significantly better than the other possibilities. This is so much faster because there are no computationally expensive computations in the function, so it allows the network to execute much faster. It has also been shown that using ReLU allows networks to converge faster than other activation functions.

2.1.3. *Layer.* Neural networks are organized into layers. Each layer contains a single type of neuron, and can only interact with the layer before it and the one after it.

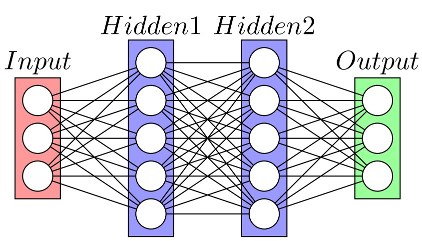

FIGURE 5. Example structure of a two layer neural network, with the four layers highlighted in different colors.

In figure 5 an example of how each layer of neurons interact with one another can be seen. Note how each neuron of the previous layer is now an input for every neuron of the current layer. We call this type of network a "two layer" network even though it clearly has four layers. This is because the input and output layers must always be present, and so it is not important if we count them or not, thus the important number is the number of hidden layers in the network.

2.1.4. *Network.*

2.2. **Forward Propagation.**

2.3. **Learning.**

2.3.1. *Gradient Descent.*

2.3.2. *Back Propagation.*

2.4. **Summary.**

3. Genetic Algorithms

3.1. **Structure.**

3.1.1. *Initialization.*

3.1.2. *Evaluation.*

3.1.3. *Selection.*

3.1.4. *Recombination.*

3.1.5. *Mutation.*

3.1.6. *Replacement.*

3.2. **Integration.**