How Do Neural Networks Work

If you've been paying attention to developments in technology, you've probably heard the words machine learning. This field has risen to prominence due to companies like Google and Facebook applying machine-learning solutions to solve some of their more challenging problems. The solutions have been so effective that leading researches like Andrew Ng, a Stanford professor, have gone as far as to call machine learning the "next electricity." A key building block in these solutions is something called a **Neural Network**.

Neural Networks have made it possible for SpaceX to recover and reuse rocket boosters. They've defeated the human champion in Go. They've also made it possible for you to use your voice to instruct your phone (and sometimes it even listens!). The way in which Neural Networks changed technology makes them seem almost magical. However, the way that they work is rather simple, and while they're incredibly powerful, they are not going to become sentient and take over humanity. In this document we'll explain how Neural networks work and try to show why they aren't going to destroy the world.

Some prerequisites for this document would be a basic understanding of linear algebra and calculus 3. However, if the reader doesn't know or has a fuzzy recollection he or she should still be able to get a general idea of how a neural network works.

Machine Learning

Before diving into neural networks it is useful to understand some the fundamental ideology behind machine learning.

In computer science, a majority of thought revolves around **algorithms**. An algorithm is a procedure that transforms an input into an output, hopefully to solve some sort of problem. Computer science researchers are interested in discovering efficient algorithms and proving their correctness. Unfortunately there are some very complex problems for which no known efficient algorithms are available. This is where machine learning is useful. Machine learning researchers look at many instances of valid input output pairs and let the computer discover the relationship between the input and output. Rather than think about the steps needed to transform the input to the output, they are focused on getting a computer to efficiently discover those steps.

In order to allow the computer to discover the series of steps needed to produce the correct solution, the steps are represented as some sort of function that transforms the input. These functions are also known as **models** and are very useful to us because they allow us to quantify how far off we are from correctly answering each problem instance, which in turn allows for adjustments to make the model more correct.

Model Architecture

In this section we'll take a look at what neural networks actually look like. First we will provide a metaphor that is useful for understanding the inspiration for the model. Then we will dive more into the math of how the model actually works

Similarity to the Brain

The setup for neural networks takes inspiration from neurons in the brain and can be helpful to understand through this analogy. Your brain takes in information through various nerves that are connected to neurons. As you can see in Figure 1, the model takes some input in the form of neurons. When they receive a signal in your brain, neurons process it and decide whether or not to fire along synapses to pass the information. Likewise, the neural network transforms the input and passes it along weighted synapses to another layer of neurons. The next layer neurons take in all of the values from the first layer than decide whether or not to fire and pass its value along the synapses to the next layer. This process is repeated for an arbitrary number of layers until we end up with an output.

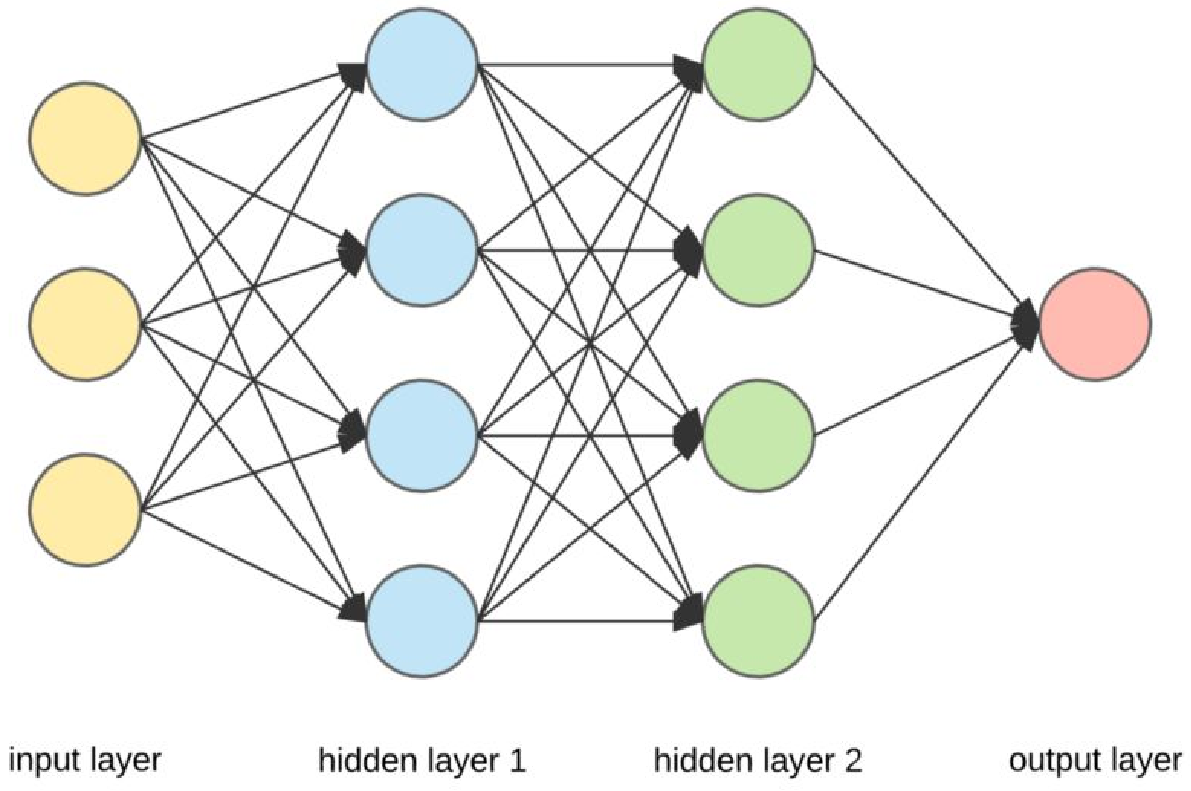

Figure 1: Example Neural Network Architecture

Prediction and Math

Though the brain metaphor is helpful for understanding the basic structure of a neural network, it is not a perfect analogy. If we really want to understand **prediction**, the process of going from input to output, we'll need to introduce some math. We can think of the input as a **vector**. For our purposes, a vector is just an ordered group of numbers. For example, this could be the pixel values in an image. The weighted synapses are then taking a **linear combination** of the input neurons. A linear combination simply involves taking a sequence of weights $w_1, \dots, w_n$ and our input sequence $x_1, \dots, x_n$ and getting a single output of $w_1x_1 + w_2x_2 + \dots + w_nx_n$. Because this is done for each neuron in the next layer, we can put all of the synapse weights going to the next layer into a matrix W_1. Then if the input is a vector x, we can exploit the properties of matrix multiplication to express the input to the second layer of neurons as $W_1 x$.

Inside each of the neurons in the next layer we apply a nonlinear activation function. This functions transforms the value the neuron received from the first layer, mimicking the process of deciding whether or not a neuron will fire. One popular activation function is Rectified Linear Units, which is defined simply as f(x) = max(x, 0). This essentially means the neuron will fire with the value it receives if the value is greater than 0. To put all of this together, the output of the model in Figure 1 would be $f(x) = W_3 \max(0, W_2 \max(0, W_1 x))$. In addition, Figure 2 shows how a sample input is manipulated into an output. While this is a decent amount of linear algebra, I'm sure you'll agree that the result of three matrix multiplications is hardly a super advanced solution that is capable of humanity's destruction.

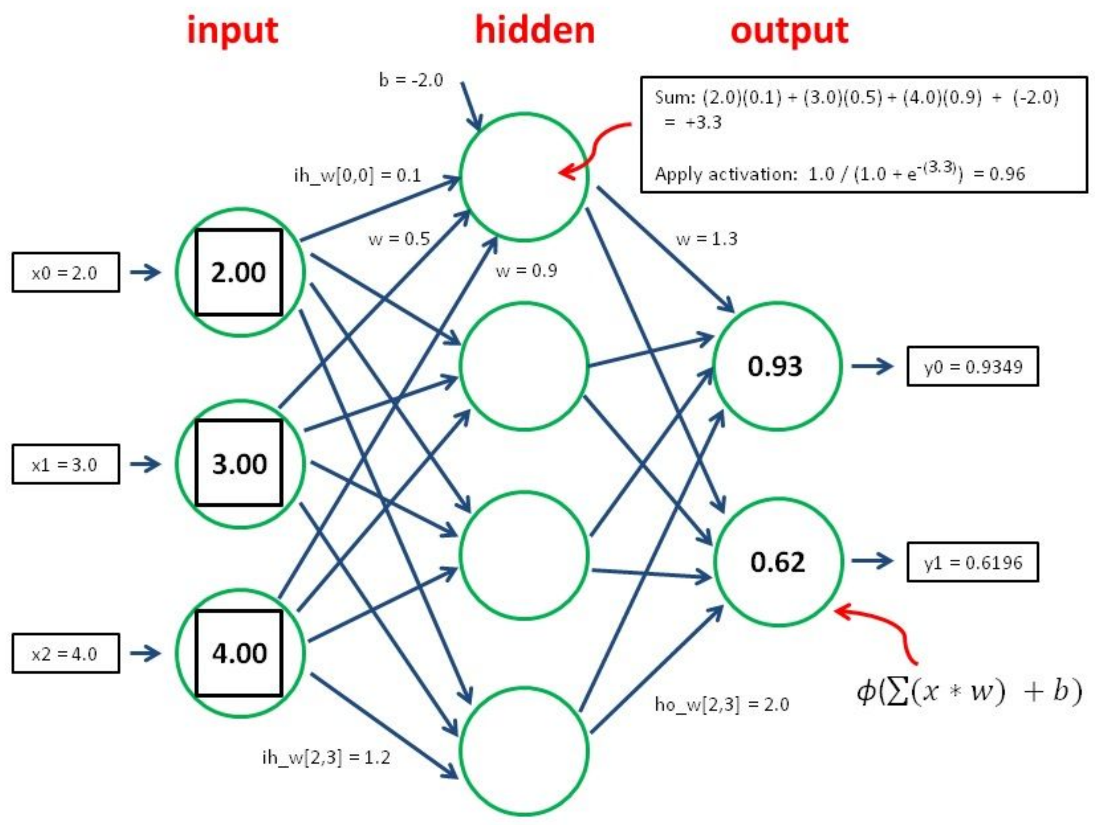

Figure 2: An example of how input is manipulated mathematically and is transformed into an actionable prediction.

Training and Evaluation

It is easy enough to understand how a neural network goes from input to the output. However before you can really appreciate how they work you" need to understand how they are trained and evaluated.

Training

In order to learn the correct solution to a problem, we first have to define correctness. This is typically done through an **objective function**. An objective function quantifies the difference between a wrong answer and a right one. This could be as simple as 0 if the model proposes the correct answer and 1 otherwise. It follows from this definition that an ideal model will minimize the value produced by this function.

In machine learning, we are typically interested in **differentiable** objective functions. A differentiable function is a function whose derivative is defined for every input value. The reason why we are interested in differentiability is because we can minimize the objective function (how wrong we are), by finding the **gradient vector** and negating it. The gradient vector is the vector obtained by taking the derivative a function with respect to each coefficient in a vector or matrix. The gradient vector is also the direction of steepest ascent in the function. By negating this vector we now have the steepest direction of descent. Since we have identified the direction we should be changing our weights to minimize the objective function we can now update them using the following rule:

$$W_{t+1}(i) = W_t(i) - \alpha \nabla L(W)$$

Where $W_{t+1}(i)$ and $W_t(i)$ is the weight matrix between the layers *i* and *i+1* after *t+1* and *t* updates, *L* is the objective function, and α is the **learning rate.** The learning rate is a constant between 0 and 1 that you multiply by the gradient to reduce the magnitude of change from each update. In Figure 3, the utility of scaling down the magnitude of change is illustrated where a lack of scaling would overshoot the minimum value of the objective function. The tradeoff you face with a low learning rate is that the model will take more updates to converge to the correct value.

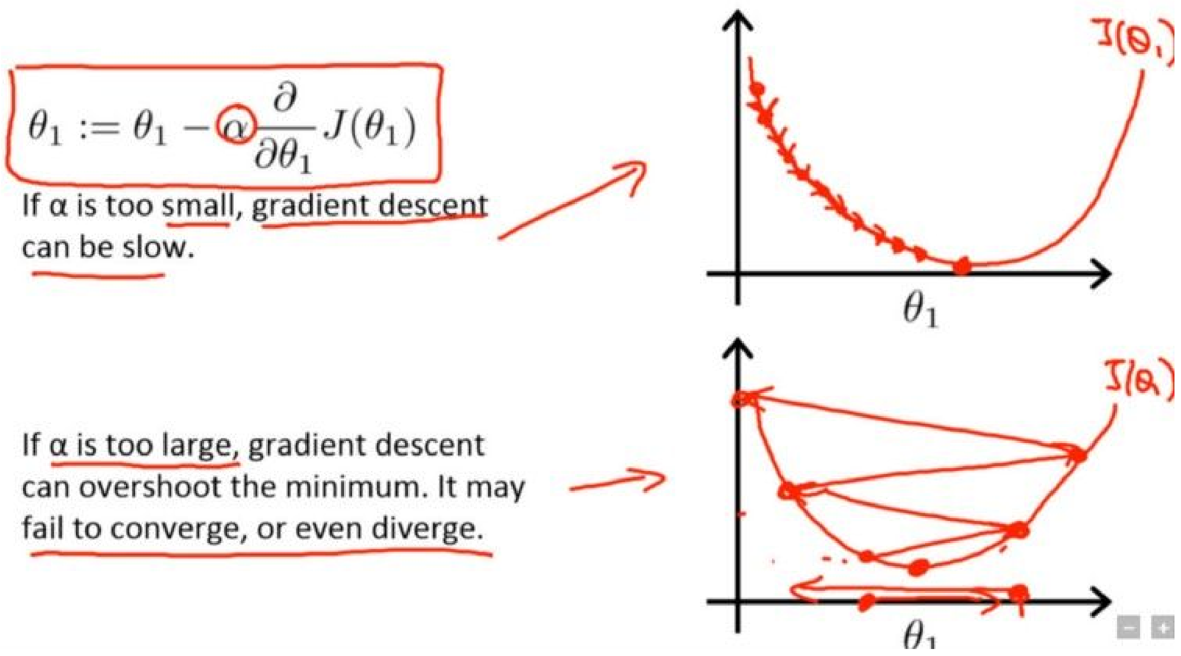
$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$
If α is too small, gradient descent can be slow.
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.
J(θ₁)
θ₁
J(θ₁)
θ₁

Figure 3: Gradient descent with small (top) and large (bottom) learning rates. Source: Andrew Ng's Machine Learning course on Coursera

Evaluation

Once we've trained a model, it follows naturally that we'd like to know how good of a model it is. An important result in machine learning shows that you can actually replicate any function with a neural network meaning that we can obtain a perfect mapping between inputs and outputs. However as you can see in Figure 4, finding a function that is a perfect mapping may not do well on unseen inputs. Because of this, it is important to distinguish between **explanatory** and **predictive** models. Explanatory models are models provide an almost perfect mapping between the provided input and output pairs. On the other hand, predictive models fit the provided input and output pairs relatively well and also do well on input output pairs that have not yet been seen.

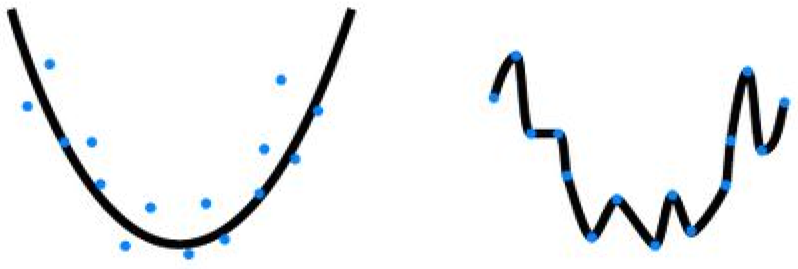

Figure 4: Illustration of how finding a function that maps input to output perfectly may be less ideal than almost mapping inputs to outputs and performing better on unseen data points.

In order to estimate how predictive a certain model is, we use **cross validation**. During cross validation, we take the data we are using to train the model and randomly split it into two sets. Figure 5 illustrates this split. The model is trained on one of the sets and then the performance of the model on the other set can be used to estimate whether or not it is a good predictive model. If there is a large difference between the value of the objective function on the dataset used for training and the hidden dataset then we may conclude that our model is more explanatory then it is predictive. Because cross validation is a good way to measure the effectiveness of a model, it can be used to evaluate different model architectures and learning rates in order to choose the most effective one for a given problem.

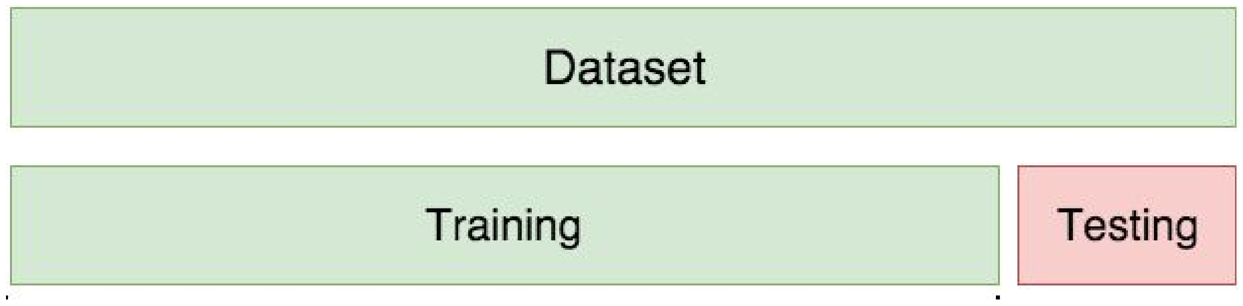

Figure 5: Illustration of how a dataset is split into two sets in order to test the predictive power of a model.

Conclusion

Now that you know how neural networks are trained and evaluated and how they make predictions, you can hopefully see the power in this approach to problem solving. However, there is a significant difference between these models, which are essentially piles of linear algebra, and a self aware artificial intelligence like SkyNet. If you found this interesting, there are many interesting developments using Neural Networks that have helped make them famous including CNNs, RNNs and GANs and you should use these topics as starting points for learning more about Neural Networks.