Neural Networks

Overview of Neural Networks

A neural network is a system in which many similar functions, called *neurons*, are composed together in order to classify inputs, perform some computation, or approximate some function. Neural networks are used today in various machine-learning applications such as handwriting- or speech-recognition, and have several interesting mathematical properties.

Definition of a Neuron

A neuron is a function $f(x_1, x_2, \ldots, x_n) = \phi(\Sigma_{i=1}^{n} w_i x_i)$, where $\{w_1, w_2, \ldots\ w_n\}$ is a set of weights, with each weight w_i corresponding to an input x_i, and where ϕ is the activation function that determines the output of the neuron based on the sum of the products of the weights and inputs. For the sake of brevity we will also notate the weight and input vectors as $\vec{w}$ and $\vec{x}$, respectively.

A neuron could be also thought of as a partial application of ϕ and $\vec{w}$ over the factory function $F(\phi, \vec{w}, \vec{x})$.

There are two popular activation functions, the perceptron and the sigmoid function. The perceptron can be defined as the function

$$\phi_P(x) = \begin{cases} 0 & : x + b \leq 0 \\ 1 & : x + b > 0 \end{cases}$$

where b is a bias that is specific to the neuron.

The sigmoid function can be defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. Though bearing some resemblance to the perceptron, the sigmoid function has the advantage of being both smooth and differentiable. These properties make training a neural network much easier.

Demonstration of the Equivalence of Various Perceptron Networks to Certain Boolean Logic Functions

One attribute of neural networks, specifically perceptron networks, is the ability to compute boolean logic functions. In this section the construction of various logic gates from perceptron networks will be demonstrated. Since modern computers are constructed from various logic gates, predominantly NAND gates, it stands to reason that by composing neural networks that compute the same as the logic gates that a modern computer could be implemented using neural networks[1]; intuitively this leads to the hypothesis that recurrent neural networks are Turing-complete. This will be proved in the next section.

Consider the following network consisting of one perceptron-type neuron, with inputs $\left(\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right)$.
Let $\vec{w} = \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$ and $b = -1$. Compare the behavior of the network to that of the AND function:

x_1	x_2	$\vec{x}\dot{\vec{w}}$	$\vec{x}\dot{\vec{w}} + b$	Output	x_1 AND x_2
0	0	0	-1	0	0
0	1	1	0	0	0
1	0	1	0	0	0
1	1	2	1	1	1

Note how the network is equivalent to the AND function over the inputs for which AND is defined.

Now let $\vec{w} = \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$ and $b = 0$. Compare the behavior of the network to that of the AND function:

x_1	x_2	$\vec{x}\dot{\vec{w}}$	$\vec{x}\dot{\vec{w}} + b$	Output	x_1 OR x_2
0	0	0	0	0	0
0	1	1	1	1	1
1	0	1	1	1	1
1	1	2	2	1	1

[1]It should be noted, however, that most implementations of neural networks do just the opposite: neurons are simulated in software. This is due to the ease of construction of digital circuitry i.e. transistors over something that requires analog signals like a sigmoid neuron.

Now, simply by changing the weights, the network output becomes equivalent to the OR function over the domain of OR.

Let's also consider NAND. NAND is important because it is functionally complete, meaning that all boolean expressions can be expressed by an equivalent combination of NAND. A proof by exhaustion is available in Appendix A. Now let $\vec{w} = \left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right)$ and $b = 2$. Compare the behavior of the network to that of the NAND function:

x_1	x_2	$\dot{\vec{x}}\vec{w}$	$\dot{\vec{x}}\vec{w} + b$	Output	x_1 NAND x_2
0	0	0	2	1	1
0	1	-1	1	1	1
1	0	-1	1	1	1
1	1	-2	0	0	0

Thus a single two-input perceptron can compute NAND.

Computer scientists met with some difficulty when attempting to weight a single perceptron to compute the exclusive-or (XOR) function. In fact, it is impossible to weight a single perceptron to do so (proof left as an exercise to the reader). However, a multilayer perceptron network can be designed to compute XOR.

Statement and Proof of the Turing-completeness of Neural Networks

As was alluded to before, a recurrent neural networks are Turing-complete. The Turing machine is a conceptual machine proposed in 1948 by Alan Turing that can be programmed to compute any possible computer algorithm. Let C be a system of computing. C is said to be Turing-complete if computers of type C are capable of simulating any single-taped Turing machine. Closely related is the idea of Turing equivalence, that a Turing machine can simulate any computer of type C. Then, by the transitive property, if follows that all Turing-equivalent computers can simulate each other.

Statement and Proof of the Universality Theorem

In the previous section it was shown that recurrent neural networks can compute any function that can be computed by an algorithm. More impressively, it is also true that a neural setwork with only a single hidden layer can compute any continuous function to an arbitrary degree of precision. This is known as the Universality Theorem.

Statement and Proof of Correctness for a Training Algorithm for a Perceptron