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

The idea for neural networks is borrowed from the field of biology; the human brain uses cells called neurons for cognition. Neurons are joined with one another via connections called synapses. Neurons create voltages based upon the voltages on certain synapses that act as inputs. Thus when the collective voltage on the input synapses exceeds a certain level the neuron becomes excited and produces a higher voltage on the output synapses.

A neuron is modelled mathematically as 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. $\Sigma_{i=1}^{n} w_i x_i$ calculates the collective weighted input; this total is then passed to the activation function ϕ which uses the collective weighted input to generate an output.

A neuron could be also thought of more generally as a partial application of ϕ and $\vec{w}$ over the function $F(\phi, \vec{w}, \vec{x})$. Such a definition may be useful for implementation of a neural network in software.

There are two popular activation functions, the perceptron and the sigmoid.

Having been invented in 1957 by Frank Rosenblatt, the perceptron was among the first activation functions to be implemented. It can be used as a standalone classifier on linearly seperable data. 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 smooth and thus differentiable. These properties make training a neural network composed of sigmoid neurons much easier then a similar network composed of perceptron neurons.

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

Neural networks can be used to compute boolean logic functions. In this section the construction of various logic gates (devices that compute logical functions) from perceptron networks will be demonstrated. Since traditional computers are constructed from logic gates, predominantly NAND[1] gates, it stands to reason that a neural network could be constructed to simulate a traditional computer;[2] intuitively this leads to the hypothesis that recurrent neural networks are Turing-complete. This will be proved in the section following the current one.

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	$\vec{x} \cdot \vec{w}$	$\vec{x} \cdot \vec{w} + b$	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 OR function:

x_1	x_2	$\vec{x} \cdot \vec{w}$	$\vec{x} \cdot \vec{w} + b$	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] NAND is the composition of NOT and AND.

[2] 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 NANDs. A proof by exhaustion is available in Appendix A. Now let $\vec{w} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$ and $b = 2$. Compare the behavior of the network to that of the NAND function:

x_1	x_2	$\vec{x} \cdot \vec{w}$	$\vec{x} \cdot \vec{w} + b$	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. Let P be a perceptron with two inputs x_1 and x_2, with associated weights w_1 and w_2, respectively, and a bias b. Assume for the sake of contradiction that P properly computes the XOR function. This implies the following behavior:

	x_1	x_2	Output $\begin{cases} 0 & : w_1x_1 + w_2x_2 + b \leq 0 \\ 1 & : w_1x_1 + w_2x_2 + b > 0 \end{cases}$
1)	0	0	0
2)	0	1	1
3)	1	0	1
4)	1	1	0

Then the following must be true:
Line (1) implies that $b \leq 0$.
Line (2) implies that $w_2x_2 > b$.
Line (3) implies that $w_1x_1 > b$.
Line (4) implies that $w_1x_1 + w_2x_2 < b$.

Adding lines (2) and (3) yields $w_1x_1 + w_2x_2 > 2b$.
$b > 0 \implies 2b > b > 0$.
Thus (4) is contradicted and P therefore cannot exist.

$\square$

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, 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, it 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

The power and versatility of neural networks in general was shown in the previous sections; however, what are the capabilities of a single neuron? A perceptron neuron can be used to classify linearly seperable data. In this section a network design and a training algorithm will be stated, and the ability of the network to correctly classify all data in the training set, provided the training set is linearly seperable, will be proven.