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. This paper will introduce the basic workings of neural networks and then examine their theoretical capabilities.

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 modeled 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 separable data. The perceptron can be defined as the function

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

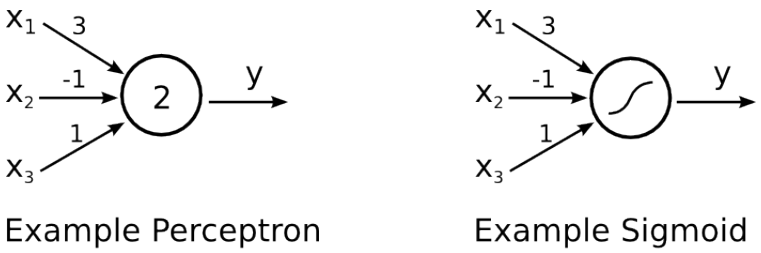

Figure 1: Example perceptron and sigmoid neurons

where b is a bias that is specific to the neuron. Notice that the bias b of the perceptron is sometimes denoted by a number written inside neuron if displayed in an image.

The sigmoid function can be defined as

$$\phi_S(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

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.

Example. *Consider the neurons in Figure 1. Note that Let* $x_1 = 3, x_2 = 2, x_3 = -0.5$. *Both of the neurons will have the same aggregate input passed to their activation functions. To determine the aggregate input, multiply the inputs by their respective weights and then sum the products:* $w_1 * x_1 + w_2 * x_2 + w_3 * x_3 = 3 * 3 + -1 * 2 + 1 * -0.5 = 6.5$ *The perceptron will output* 1 *since* $\phi(6.5) = 1$ *(see Equation 1). The sigmoid will then output* $\sigma(6.5) = \frac{1}{1+e^{-6.5}} \approx 0.9985$.

Construction of Neural Networks

More complex behavior can be created by linking the outputs of some neurons into the inputs of other neurons. This is how the human brain is constructed. An example of a neural network is shown in Figure 2. The network pictured is considered a four-layer sigmoid network because each layer besides the input layer consists of sigmoid neurons as denoted by the sigmoid shape inscribed in the representation of each neuron. The input layer does not contains actual neurons but rather is drawn by convention to show the inputs being fed to the second layer. The hidden layers encompass all of the neurons between the input layer and the output layer.

The output layer contains the neurons whose output is interpreted as the output of the network. In the figure shown the output layer only contains one output neuron. It is possible to have more than one output from a neural network; for example, a binary number could be encoded in the outputs.

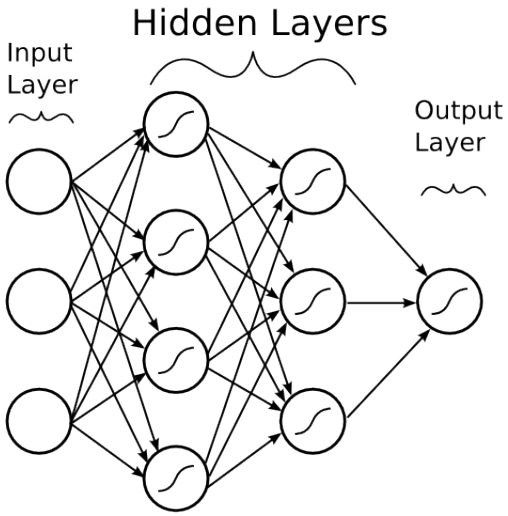

Figure 2: A four-layer neural network of sigmoid neurons.

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 $\binom{x_1}{x_2}$.
Let $\vec{w} = \binom{1}{1}$ and $b = -1$. Compare the behavior of the network to that of the AND function:

[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.

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 $\vec{w} = \binom{1}{1}$ 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

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. Now let $\vec{w} = \binom{-1}{-1}$ 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 $w_2 + b > 0$.
Line (3) implies that $w_1 + b > 0$.
Line (4) implies that $w_1 + w_2 + b \leq 0$.

Lines (2) and (3) imply that $w_1 > -b$ and $w_2 > -b$.
Then $w_1 + w_2 + b > (-b) + (-b) + b = -b \geq 0$.
Combining this with line (4) implies that $w_1 + w_2 + b = 0$.
Adding lines (2) and (3) yield $w_1 + b + w_2 + b = w_1 + w_2 + b + b > 0$.
Then the previous result implies that $w_1 + w_2 + b + b = 0 + b > 0$.
This contradicts line (1).
Therefore the assumption is false and no perceptron can compute XOR.

□

However, a multilayer perceptron network can be designed to compute XOR. One way to demonstrate this is by constructing a network of perceptrons weighted to compute NAND. Then XOR can be computed because $\{NAND\}$ is functionally complete; that is, all boolean functions can be expressed as a composition of NAND.

Definition 1. Let S be a set of truth functions. Then S is functionally complete iff all possible truth functions are definable from S[2].

The following proof borrows heavily from ProofWiki[2].

Proof. Let $\{0, 1\}$ be the set of truth values, with 0 signifying a false value and 1 signifying a true value. Consider the following truth table of binary boolean functions. On the left side are the two inputs x_1 and x_2. Because there are two inputs, each of which can assume two values, there are four possible inputs, yielding $2^4 = 16$ possible outputs. Thus the listing on the right side of the table of binary functions is exhaustive.

For purposes of readability the table has been broken horizontally into two halves.

x_1	x_2	f_F	AND	$(\neg \Rightarrow)$	pr_1	$(\neg \Leftarrow)$	pr_2	XOR	OR
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

x_1	x_2	NOR	$\Leftrightarrow$	$(\neg \text{pr}_2)$	$\Leftarrow$	$(\neg \text{pr}_1)$	$\Rightarrow$	NAND	f_T
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

The following table will demonstrate that the set $S_1 = \{\neg, \Rightarrow, \text{AND}, \text{OR}\}$ is functionally complete. Each possible binary boolean function not in S_1 is listed down the first column. An equivalent expression for each is function is written in the second column. The third column contains the set of boolean functions that have been used so far in the chart without having an equivalent expression given for them.

Function	Equivalent Expression	Functions Used So Far
f_T	$x_1 \Leftrightarrow x_2$	$\{\Leftrightarrow\}$
f_F	$x_1 \text{XOR} x_2$	$\{\Leftrightarrow, \text{XOR}\}$
XOR	$\neg(x_1 \Leftrightarrow x_2)$	$\{\Leftrightarrow, \neg\}$
$\Leftrightarrow$	$(x_1 \Rightarrow x_2)\text{AND}(x_1 \Leftarrow x_2)$	$\{\neg, \Rightarrow, \Leftarrow\}$
$(\neg \text{pr}_1)$	$\neg \text{pr}_1$	$\{\neg, \Rightarrow, \Leftarrow, \text{pr}_1\}$
$(\neg \text{pr}_2)$	$\neg \text{pr}_2$	$\{\neg, \Rightarrow, \Leftarrow, \text{pr}_1, \text{pr}_2\}$
pr_1	$x_1 \text{AND} x_2$	$\{\neg, \Rightarrow, \Leftarrow, \text{pr}_2\}$
pr_1	$x_2 \text{AND} x_2$	$\{\neg, \Rightarrow, \Leftarrow\}$
NAND	$\neg(x_1 \text{AND} x_2)$	$\{\neg, \Rightarrow, \Leftarrow\}$
NOR	$\neg(x_1 \text{OR} x_2)$	$\{\neg, \Rightarrow, \Leftarrow\}$
$(\neg \Rightarrow)$	$\neg \Rightarrow$	$\{\neg, \Rightarrow, \Leftarrow\}$
$(\neg \Leftarrow)$	$\neg \Leftarrow$	$\{\neg, \Rightarrow, \Leftarrow\}$

□

NOTE: This proof is not yet complete.

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 network 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 separable 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 separable, will be proven.

References

[1] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2014

[2] `https://proofwiki.org/wiki/Definition:Logical_NAND`

[3] Simon Haykin, "Neural Networks: A Comprehensive Foundation", Maxwell Macmillan International, 1994

[4] Heikki Hyötyniemi, "Turing Machines are Recurrent Neural Networks", Publications of the Finnish Artificial Intelligence Society, pp. 13-24, `http://lipas.uwasa.fi/stes/step96/step96/hyotyniemi1/`