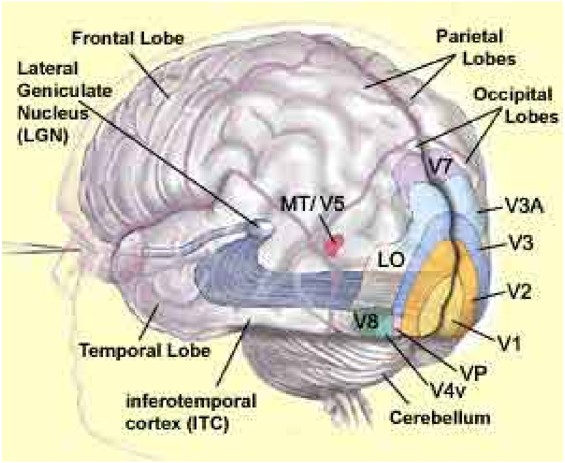
CS 5600/6600: Lecture 1
Artificial Neural Networks (ANNs): Part 1

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

Human Brain's Visual Cortices



source: mybrainnotes.com

Human Brain’s Visual Cortices

In each hemisphere of our brain, we have a primary visual cortex called V1.

V1 contains 140,000,000 neurons with tens of billions of connections between them.

Human vision involves not just V1 but also a series of visual cortices called V2, V3, V4, and V5, each of which does progressively more complex image processing.

Humans are very skillful at making sense of what their eyes show them, all of which is done unconsciously.

Neuron Connectivity

Biological learning systems are built of very complex webs of interconnected neurons.

The human brain is estimated to contain a densely interconnected network of approximately 10^{11} neurons, each connected, on average, to 10^{4} other neurons.

Neuron activity is excited/fired or inhibited through connections to other neurons.

Neuron Switching Speeds

The fastest human neuron switching time is approximately 10^{-3} seconds.

By comparison, the computer switching time is 10^{-10} seconds.

Humans make surprisingly complex decisions suprisingly fast: it takes 10^{-1} seconds to visually recognize one's mother.

Some researchers speculate that information processing capabilities of biological neural systems are highly parallel and representations are distributed over many areas of the brain.

Outline

Simulating Biological Networks and Neurons

Biological networks are simulated with artificial neural networks (ANNs).

ANNs consist of artifical neurons that simulate biological neurons.

Several decades of research on ANNs have produced many types of neurons, of which *perceptrons* and *sigmoid neurons* are the most popular.

McCulloch and Pitts' Theory of Neurons

In 1943, McCulloch and Pitts published their classic paper "A Logical Calculus of the Ideas Immanent in Nervous Activity," where they presented a mathematical theory of neurons. They called neurons *perceptrons*. Five basic assumptions of McCulloch and Pitts' Theory:

1) The activity of the neuron is an "all-or-none" process (either active or passive with no in-between);

2) A certain number of synapses must be exicited within the period of latent addition in order to exercise a neuron at any time; and this number is independent of previous activity of the neuron;

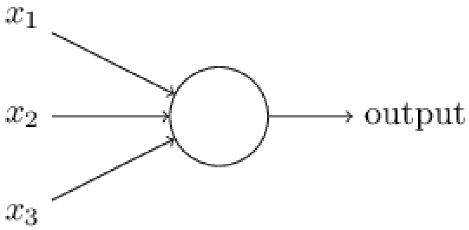
3) The only significant delay within the nervous system is synaptic delay;

4) The activity of any inhibitory synapse absolutely prevents execitation of the neuron at that time;

5) The structure of the net does not change with time.

Perceptron

A perceptron takes several binary inputs x_1, x_2, ... x_n and produces a single binary output.



Perceptron's Output

Connections (aka synapses) from the inputs to the perceptron are governed by weights w_1, w_2, ..., w_n.

Weights are real numbers expressing the importance of the respective inputs to the output.

The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum_j w_j x_j$ is $\leq$ or $>$ than some threshold θ.

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \theta \\ 1 & \text{if } \sum_j w_j x_j > \theta. \end{cases}$$

Simplified Notation for Perceptrons

We can write the weighted sum as the dot product:
$\sum_j w_j x_j = w \cdot x$.

We can also move the threshold θ inside the inequalities and call it *bias* or b. If $b = -\theta$, then

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \leq 0 \\ 1 & \text{if } \sum_j w_j x_j + b > 0. \end{cases}$$

One can think of the bias b as a measure of how easy it is to get the perceptron to output 1, i.e., to *fire*. The more negative the bias, the harder for the perceptron to output 1, and vice versa.

Perceptrons as 1-Layer Networks

Perceptrons have just 2 layers of nodes (input nodes and output nodes) and are often called *single-layer* networks, because they have only one layer of connections.

Weights may be negative, which causes higher positive inputs not to fire.

Positive and negative inputs tend to cancel each other out.

Perceptrons and Logical Functions

In their paper "A Logical Calculus of the Ideas Immanent in Nervous Activity, " McCulloch and Pitts showed that perceptrons can implement the three fundamental logical functions:

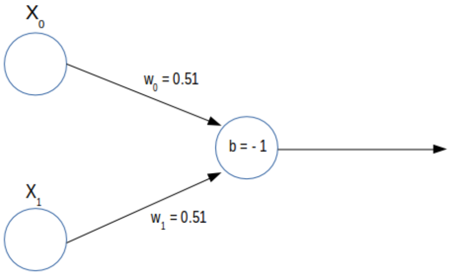
1. AND;
2. OR;
3. NOT.

Why is this important? Because every logical proposition can be represented in terms of AND, OR, and NOT.

Logical AND

X_0	X_1	$X_0 \wedge X_1$
0	0	0
0	1	0
1	0	0
1	1	1

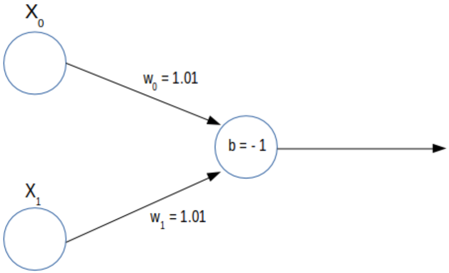
Sample AND Perceptron



Logical OR

X_0	X_1	$X_0 \vee X_1$
0	0	0
0	1	1
1	0	1
1	1	1

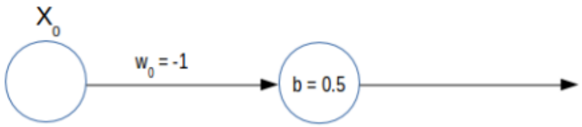
Sample OR Perceptron



Logical NOT

X_0	$\neg X_0$
1	0
0	1

Sample NOT Perceptron



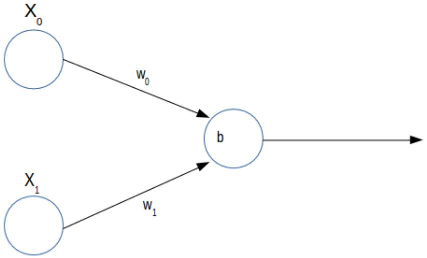
Importance of Logical Perceptrons

Logic is a model of how symbolic thought processing works in the human brain, and has been used as such a model for over 2,000 years (some historians of mathematics argue that this period is closer to 5,000 years).

Since logical formulas can be implemented as perceptron networks, complex behaviors of the human brain can be understood or modeled through perceptron networks.

McCulloch and Pitts' hypothesis: complex behaviors of the human brain can be produced by its cells wired in different ways.

When Does a Binary Perceptron Fire?



If $b \geq 0$, this binary perceptron fires if

$$x_0 w_0 + x_1 w_1 > -b$$

or

$$x_1 > \frac{-w_0}{w_1} x_0 - \frac{b}{w_1}.$$

Geometrical Interpretation of Perceptron's Firing

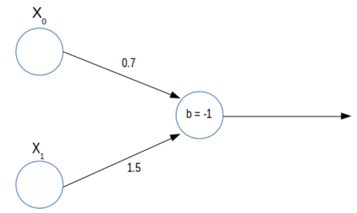
A single binary perceptron represents a line that separates the inputs on the one side of the line from the inputs on the other side of the line.

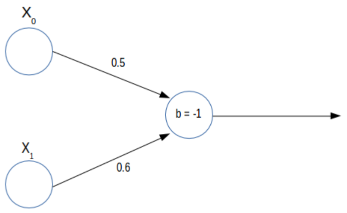
Example: Two AND Perceptrons

Let's consider the binary AND table again and construct two AND perceptrons and see if one is better than the other.

x_0	x_1	$x_0 \wedge x_1$
0	0	0
0	1	0
1	0	0
1	1	1

Example: Two AND Perceptrons

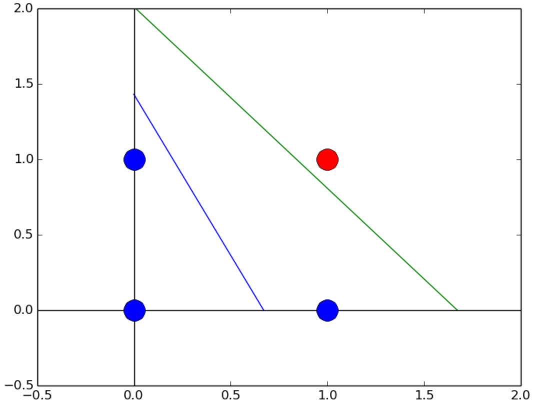




$x_1 = -0.47x_0 + 0.67$ $x_1 = -0.83x_0 + 1.67$

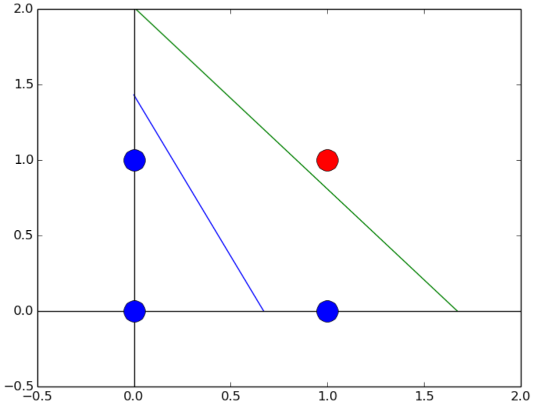
Two AND Perceptron Lines

The blue line corresponds to the left AND perceptron on the previous slide; the green line corresponds to the right perceptron on the previous slide. The blue dots are 0's (i.e., when the perceptrons output 0) and the red dot is 1. Which perceptron is better?



Two AND Perceptron Lines

A moment's reflection should be sufficient to convince you that the perceptron whose behavior is expressed by the green line is much better, because it separates the 0 outputs from the only 1 output.



Logical XOR

The logical XOR function is defined in one of the two ways:

$$x_0 \oplus x_1 \equiv (x_0 \vee x_1) \wedge \neg(x_0 \wedge x_1);$$

$$x_0 \oplus x_1 \equiv (x_0 \wedge \neg x_1) \vee (\neg x_0 \wedge x_1).$$

The truth table for logical XOR is as follows:

x_0	x_1	$x_0 \oplus x_1$
0	0	0
0	1	1
1	0	1
1	1	0

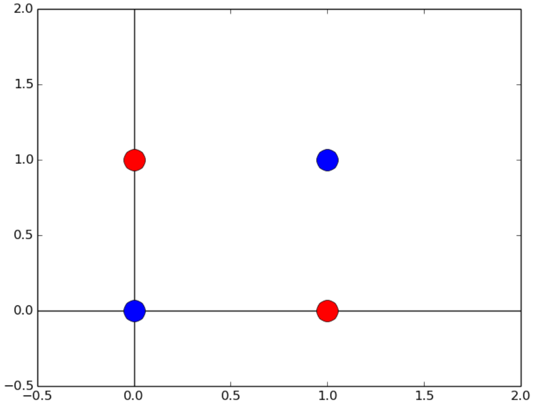
Logical XOR Problem

Is it possible to construct a binary perceptron to model logical XOR?

x_0	x_1	$x_0 \bigoplus x_1$
0	0	0
0	1	1
1	0	1
1	1	0

Logical XOR Problem

Here is a plot for the four logical XOR outcomes. Can we construct a single perceptron to separte the two positive outcomes (reds) and the two negative outcomes (blues)?



Logical XOR Problem for a Single Perceptron

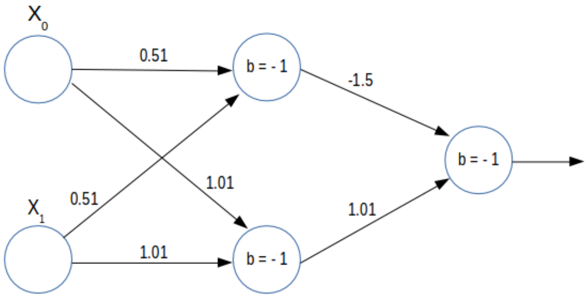
The logical XOR problem cannot be solved with a single binary perceptron, because a single binary perceptron draws just one line in the 2D space, and we need two lines to separate the negatives from the positives.

Question

Can we model logical XOR with more than one binary perceptron?

Answer

Yes, we can. Here's a perceptron that models XOR. However, it's no longer a single layer perceptron.



NAND Gates

NAND (NOT AND) is a logical gate that produces 0 iff all of its inputs are 1's. The function $NAND(a_1, a_2, ..., a_n)$ is logically equivalent to $\neg(a_1 \wedge a_2 \wedge ... \wedge a_n)$.

In theory of computability, NAND is significant, because any boolean function can be implemented as a combination of NAND gates.

Question

Is it possible to simulate NAND gates with perceptrons?

Answer

Yes, it is, because percentrons can implement AND, OR, and NOT.

Since NAND gates are universal to computation, perceptrons are also universal to computation.

It is reassuring, because it tells us that perceptron networks can be as powerful as any other computational device that uses logical circuits.

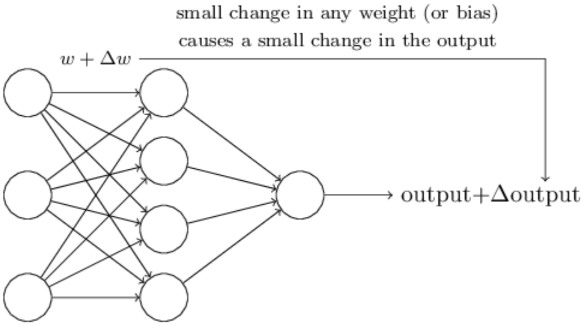
OK, but what's the big deal?

The Big Deal

The big deal is the existence of *learning algorithms* that can automatically tune the weights and biases in networks of perceptrons (and other neurons). This makes such networks radically different from circuits.

When Networks Learn Well

Learning Proportionality Principle: A network of neurons learns well when a small change in a weight/bias causes a small change in the network's output.


small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

Lack of Proportionality in Perceptrons

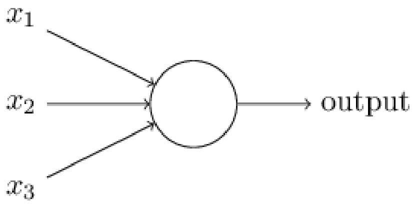
It has been emperically discovered that networks of perceptrons do not have the proprotionality of weight/bias and output.

A new type of neuron, a *sigmoid* neuron, has been introduced to overcome this problem.

Sigmoid neurons are similar to perceptrons but modified so that small changes in weights/biases are likely to cause only small changes in the output.

Sigmoid Neuron

A sigmoid neuron takes several inputs x_1, x_2, ... x_n but, unlike in a perceptron, these inputs are real numbers between 0 and 1. The output is also a real number between 0 and 1, not just 0 or 1, as with a perceptron.



Sigmoid Neuron's Output

The output of a sigmoid neuron with the inputs x_1, x_2, ... x_n, the weights w_1, w_2, ..., w_n, and the bias b is

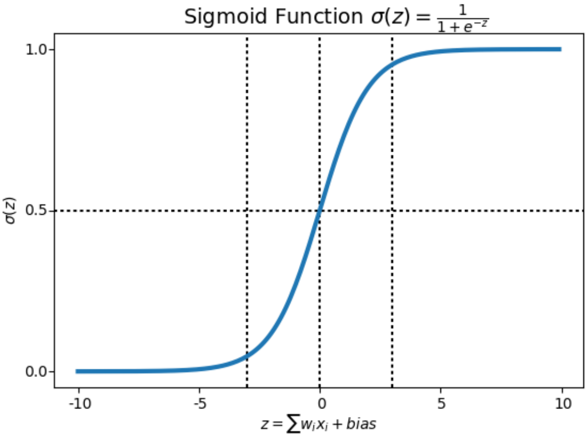
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

or

$$\sigma(w \cdot x + b) = \frac{1}{1 + exp(-\sum_j w_j x_j - b)},$$

where $z = w \cdot x + b$.

Sigmoid Neuron's Output Function (aka Activation Function)



What Do Sigmoid Neurons Buy Us?

The use of sigmoid neurons makes it much more likely that a small change in a weight/bias causes only a small change in output. We can use this property in modifying the weights/biases to get our network to behave more smoothly.

Basic takeaway: The function σ turns sigmoid neurons into smoothed perceptrons.

Question

How can we approximate output change in a sigmoid neuron?

Approximating Output Change in a Sigmoid Neuron

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b,$$

where the sum is over all the weights, w_j, and $\partial output/\partial w_j$ and $\partial output/\partial b$ are partial derivatives of the output with respect to w_j and b, respectively.

$\Delta output$ is a linear function of Δw_j and Δb, which makes it possible to approximate small changes in output via small changes in weights and biases.

A practical reason to use σ is that it has nice differentiation properties.

Sigmoid Activation Function and its Derivative

Here is the sigmoid function, its derivative and derivative approximation:

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

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)).$$

$$\frac{d}{dx}\sigma(x) \approx x(1 - x).$$

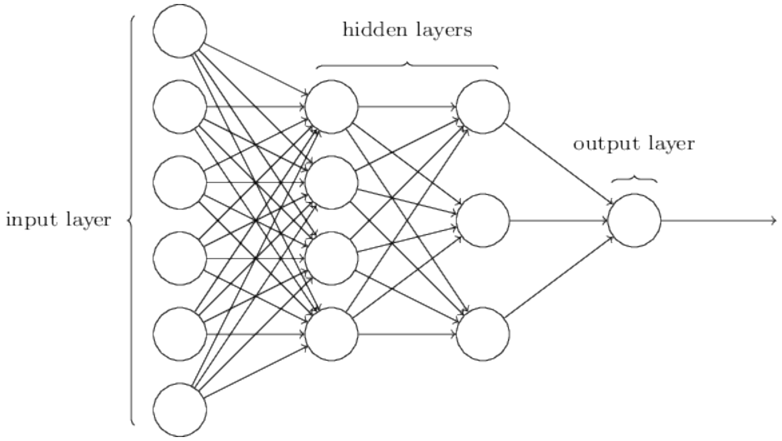
Layers

Neural networks consists of layers. Each layer consists of neurons (either perceptrons or sigmoids).

There can be arbitrarily many layers stacked on each other. The leftmost layer is called the *input* layer.

The righmost layer is called the *output* layer. The layers in between the input layer and the output layer, if there are any, are called the *hidden* layers.

Layers



Terminological Note

Multilayer ANNs are sometimes called *multilayer perceptrons* (MLPs) although they are made up of sigmoid neurons, not perceptrons. The term MLP can be confusing.

Synapses and Neurons

The links between neurons, called *synapses*, indicate which neurons input their outputs to which other neurons. These links have weights.

The job of a neuron is to take the inputs from the synapses and apply its activation function to them.

The job of a synapse is to take the value from the input neuron, multiply the value by its weight, and output the result into the output neuron.

Parameters vs. Hyperparameters

The neural networks differ from each other in terms of their layers, types of neurons, synapses, and activation functions. These are called *hyperparameters*. Hyperparameters cannot be changed.

The parameters of a neural network are weights and biases that can be manipulated and/or learned.

The neural networks also differ from each other by their learning algorithms.

References

1. M. Neilsen. *Neural Networks and Deep Learning*.
2. T.M. Mitchell. *Machine Learning*.