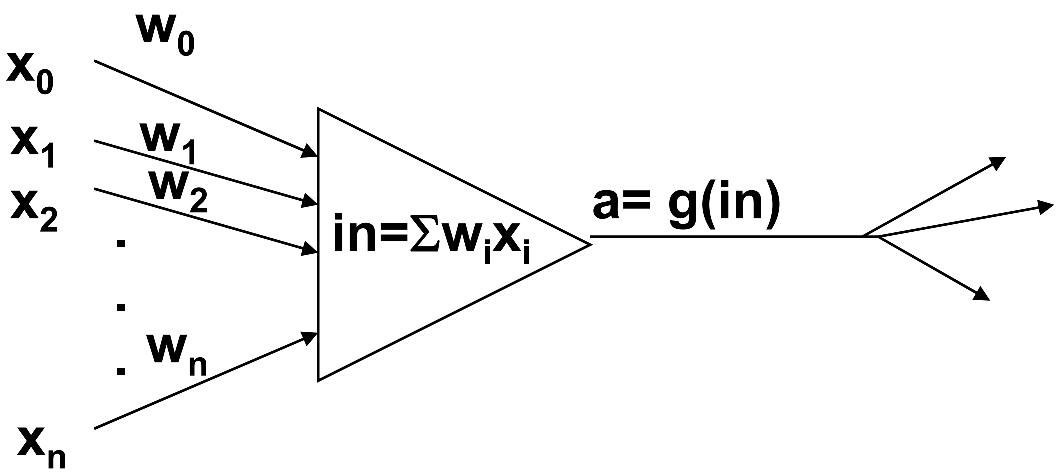
Chapter 20 Section 5 --- Slide Set 2

Perceptron examples

Additional sources used in preparing the slides:
Nils J. Nilsson's book: Artificial Intelligence: A New Synthesis
Robert Wilensky's slides: http://www.cs.berkeley.edu/~wilensky/cs188

A unit (perceptron)



x_i are the inputs w_i are the weights

w_0 is usually set for the threshold with x_0 =-1 (bias)

in is the weighted sum of inputs including the threshold (activation level)

g is the activation function

a is the activation or the output. The output is computed using a function that determines how far the perceptron's activation level is below or above 0

A single perceptron's computation

A perceptron computes $a = g(X \cdot W)$,

where

$in = X.W = w_0 * -1 + w_1 * x_1 + w_2 * x_2 + \ldots + w_n * x_n$,

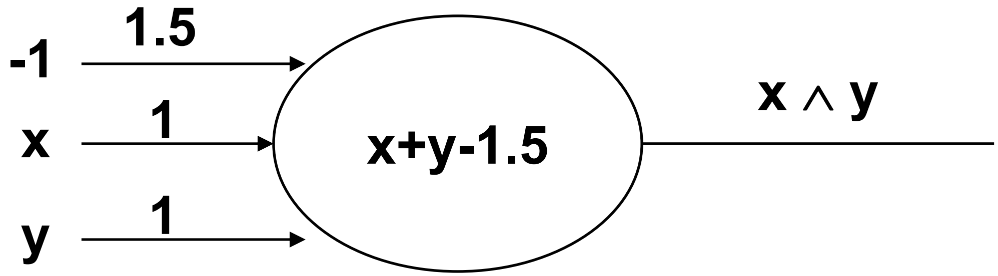
and g is (usually) the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z > 0 \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

A perceptron can act as a logic gate interpreting 1 as true and 0 (or -1) as false

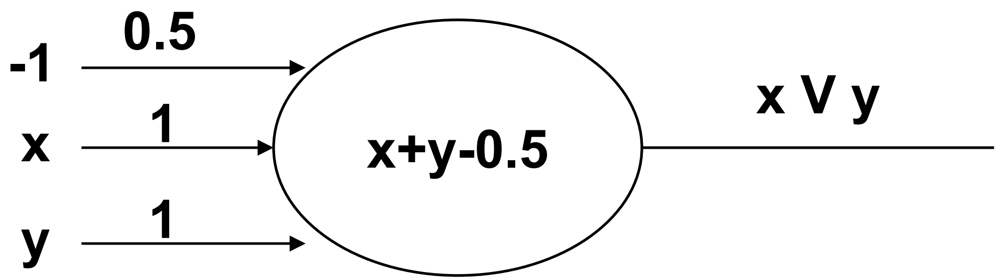
Notice in the definition of g that we are using $z>0$ rather than $z \geq 0$.

Logical function and



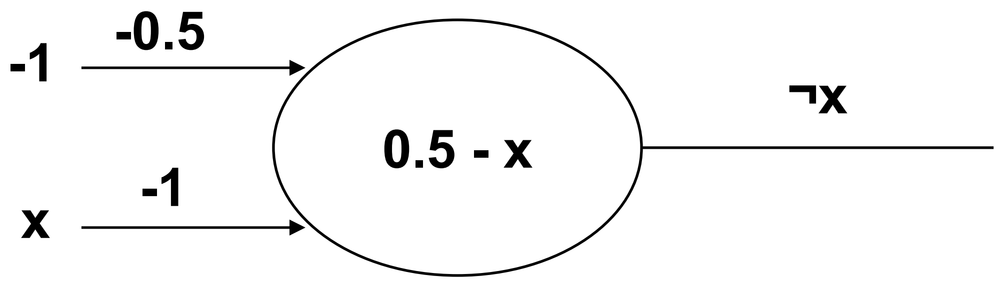
x	y	x+y-1.5	output
1	1	0.5	1
1	0	-0.5	0
0	1	-0.5	0
0	0	-1.5	0

Logical function or



x	y	x+y-0.5	output
1	1	1.5	1
1	0	0.5	1
0	1	0.5	1
0	0	-0.5	0

Logical function not



x	0.5 - x	output
1	-0.5	0
0	0.5	1

Interesting questions for perceptrons

- How do we wire up a network of perceptrons?
  - i.e., what "architecture" do we use?
- How does the network represent knowledge?
  - i.e., what do the nodes mean?
- How do we set the weights?
  - i.e., how does learning take place?

Training single perceptrons

- We can train perceptrons to compute the function of our choice
- The procedure
  - Start with a perceptron with any values for the weights (usually 0)
  - Feed the input, let the perceptron compute the answer
  - If the answer is right, do nothing
  - If the answer is wrong, then modify the weights by adding or subtracting the input vector (perhaps scaled down)
  - Iterate over all the input vectors, repeating as necessary, until the perceptron learns what we want

Training single perceptrons: the intuition

- **If the unit should have gone on, but didn't, increase the influence of the inputs that are on:**
  - **adding the inputs (or a fraction thereof) to the weights will do so.**

- **If it should have been off, but was on, decrease influence of the units that are on:**
  - **subtracting the input from the weights does this.**

- **Multiplying the input vector by a number before adding or subtracting scales down the effect. This number is called the *learning constant*.**

Example: teaching the logical or function

Want to learn this:

Bias	x	y	output
-1	0	0	0
-1	0	1	1
-1	1	0	1
-1	1	1	1

Initially the weights are all 0, i.e., the weight vector is (0 0 0).

The next step is to cycle through the inputs and change the weights as necessary.

Walking through the learning process

Start with the weight vector (0 0 0)

ITERATION 1

Doing example (-1 0 0 0)
The sum is 0, the output is 0, the desired output is 0.
The results are equal, do nothing.

Doing example (-1 0 1 1)
The sum is 0, the output is 0, the desired output is 1.
Add half of the inputs to the weights.
The new weight vector is (-0.5 0 0.5).

Walking through the learning process

The weight vector is (-0.5 0 0.5)

Doing example (-1 1 0 1)
The sum is 0.5, the output is 1, the desired output is 1.
The results are equal, do nothing.

Doing example (-1 1 1 1)
The sum is 1, the output is 1, the desired output is 1.
The results are equal, do nothing.

Walking through the learning process

The weight vector is (-0.5 0 0.5)

ITERATION 2

Doing example (-1 0 0 0)
The sum is 0.5, the output is 1, the desired output is 0.
Subtract half of the inputs from the weights.
The new weight vector is (0 0 0.5).

Doing example (-1 0 1 1)
The sum is 0.5, the output is 1, the desired output is 1.
The results are equal do nothing.

Walking through the learning process

The weight vector is (0 0 0.5)

Doing example (-1 1 0 1)
The sum is 0, the output is 0, the desired output is 1.
Add half of the inputs to the weights.
The new weight vector is (-0.5 0.5 0.5)

Doing example (-1 1 1 1)
The sum is 1.5, the output is 1, the desired output is 1.
The results are equal, do nothing.

Walking through the learning process

The weight vector is (-0.5 0.5 0.5)

ITERATION 3

Doing example (-1 0 0 0)
The sum is 0.5, the output is 1, the desired output is 0.
Subtract half of the inputs from the weights.
The new weight vector is (0 0.5 0.5).

Doing example (-1 0 1 1)
The sum is 0.5, the output is 1, the desired output is 1.
The results are equal do nothing.

Walking through the learning process

The weight vector is (0 0.5 0.5)

Doing example (-1 1 0 1)

The sum is 0.5, the output is 1, the desired output is 1.

The results are equal, do nothing.

Doing example (-1 1 1 1)

The sum is 1.5, the output is 1, the desired output is 1.

The results are equal, do nothing.

Walking through the learning process

The weight vector is (0 0.5 0.5)

ITERATION 4

Doing example (-1 0 0 0)
The sum is 0, the output is 0, the desired output is 0.
The results are equal do nothing.

Doing example (-1 0 1 1)
The sum is 0.5, the output is 1, the desired output is 1.
The results are equal do nothing.

Walking through the learning process

The weight vector is (0 0.5 0.5)

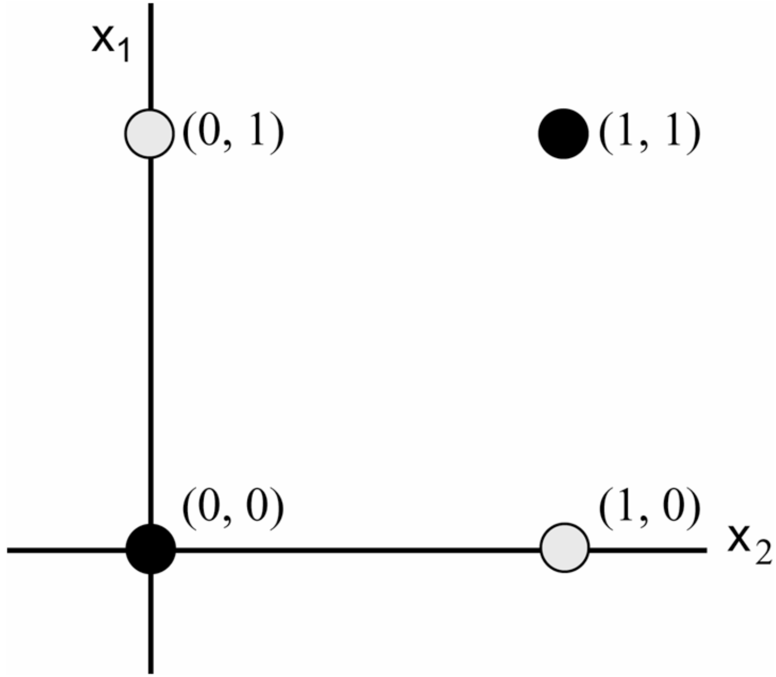
Doing example (-1 1 0 1)
The sum is 0.5, the output is 1, the desired output is 1.
The results are equal, do nothing.

Doing example (-1 1 1 1)
The sum is 1.5, the output is 1, the desired output is 1.
The results are equal, do nothing.

Converged after 3 iterations!

Notice that the result is different from the original design for the logical or.

The bad news: the exclusive-or problem



No straight line in two-dimensions can separate the (0, 1) and (1, 0) data points from (0, 0) and (1, 1).

A single perceptron can only learn *linearly separable* data sets (in any number of dimensions).

The solution: multi-layered NNs