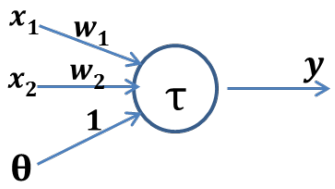
CSE5095 – Research Topics in Big Data Analytics | Professor Rajasekaran
Date: 04/24/2014 | Note taker: Wei Zeng

Artificial Neural Networks (ANN)

- An Artificial Neural Network (ANN) is a directed graph G(V,E)
  V->Processors
  E->Edges with weights
- Perceptron

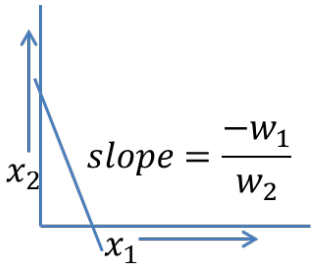


where x_is are the inputs, y is the output, w_is are the weights, θ is the bias constant and τ is the threshold.

If $w_1x_1 + w_2x_2 + \theta \geq \tau$ **then** $y = 1$
else $y = 0$

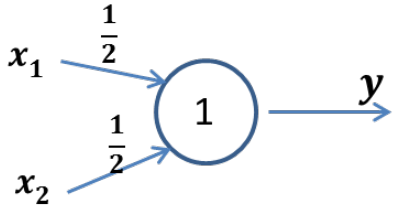
Without loss of generality, let $\tau = 0 \Rightarrow w_1x_1 + w_2x_2 + \theta = 0$

$x_2 = \frac{-w_1x_1-\theta}{w_2} = \frac{-w_1x_1}{w_2} - \frac{\theta}{w_2}$. We can think of a perceptron as a binary classifier. If we can find a straight line that separates the two classes, then this straight line can be used to specify a corresponding perceptron.



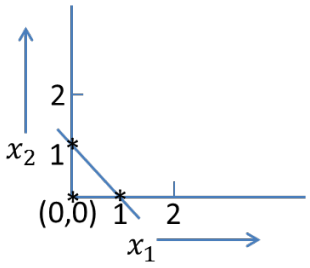
Fact: We can build perceptrons for various Boolean operations.

E.g., consider the Boolean Function AND: $f(x_1, x_2) = x_1 \wedge x_2$

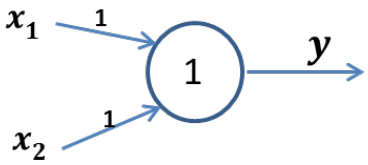


A choice of $w_1 = 1/2, w_2 = 1/2, \theta = 1$ suffices to realize the AND operation. Clearly, there exists a straight line that separates the two sets: {(0,0), (1,0), (0,1)} and {(1,1)}.

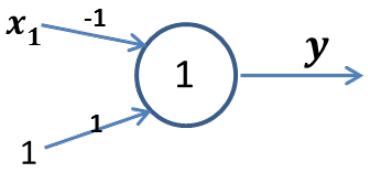
x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1



Consider the Boolean Function OR: $f(x_1, x_2) = x_1 \vee x_2$
A choice of $w_1 = 1, w_2 = 1$, and $\theta = 1$, prescribes a perceptron for the Boolean OR function.



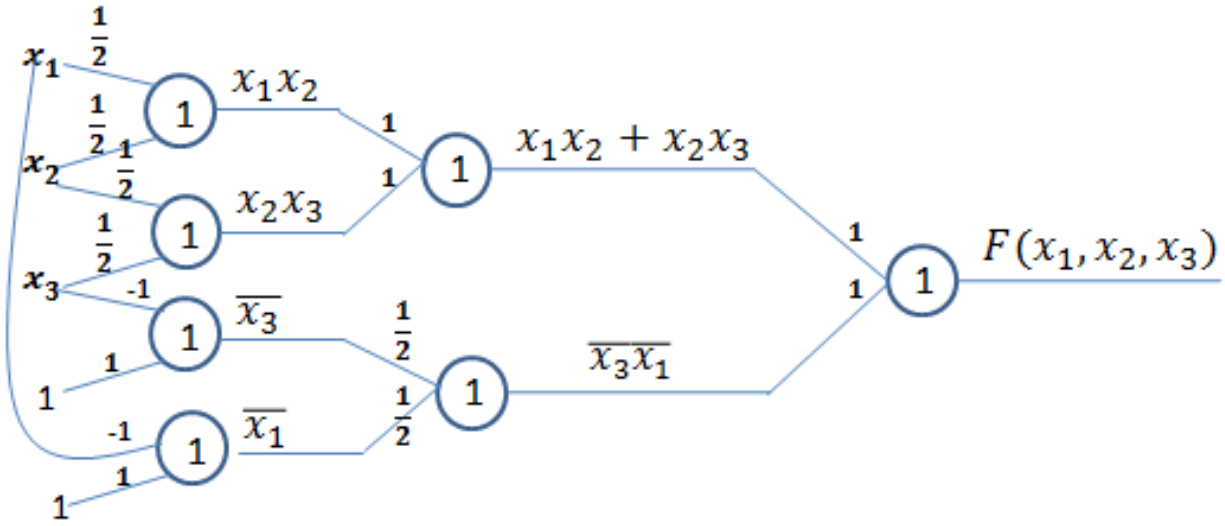
Now consider the Boolean Function NOT: $f(x) = \neg x$



Fact: Any Boolean can be realized with $\wedge, \vee$ and $\neg$.

⇨ For any Boolean function, ∃ an ANN.

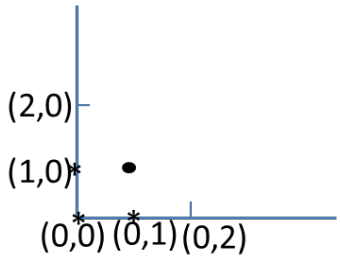
Example: $f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + \overline{x_3x_1}$
We can combine the above perceptrons to construct an ANN for the above function as follows:



- Consider: $f(x_1, x_2) = x_1 \oplus x_2$

  $\oplus$ means XOR

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0



Fact: There is no perceptron to realize XOR. Since there is no straight line that can separate the two sets: {(0,0), (1,1)} and {(0,1), (1,0)}.

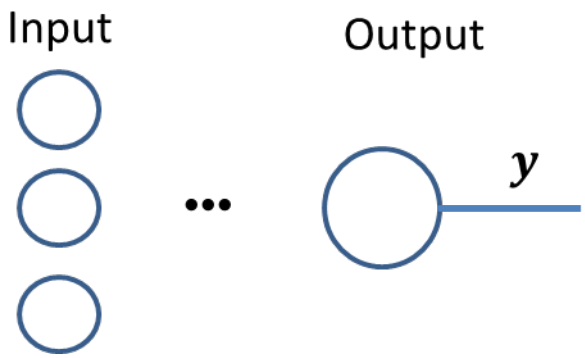
Back Propagation

Delta rules for learning the weights

An ANN can be used to learn concepts. Any learning algorithm is supplied with examples. An example could either be positive or negative. A positive example is nothing but a set of values to the inputs under which the network's output is 1. A negative example is a set of values to the inputs under which the output from the network is 0. (Here we have assumed that there are many inputs to the network and there is a single output that could either be 1 or 0. We can extend these in a natural manner). To learn a concept with an ANN, we first come up with the underlying topology (i.e., graph) for the ANN. Then we have to compute the weights on edges and threshold for nodes. We use the examples for this purpose. A simple strategy for the initial values for the weights could be random values. Without loss of generality, all the threshold values could be chosen to be zero. When we pass every example though the network, the weights get modified. When we process many examples, the hope is that the weights would have converged to the right values corresponding to the concept of interest.

A simple algorithm for adjusting weights is known as back propagation delta rule. For every given example, we know what the inputs and expected output are. We feed the input values, corresponding to an example, to the network and observe the output from the network. If the observed output and the expected output are the same, we don't do any adjustments to the weights. If these two are different we adjust the weights proportional to the difference between the expected and observed outputs.

Thus, for every example, there are two steps. The first step is to compute the output of the network. The second step is to adjust the network parameters as needed.



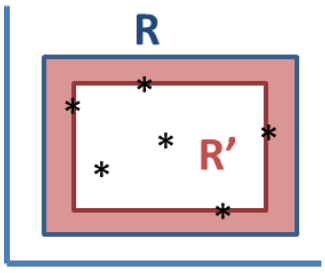
Let y be the expected output, and let y_a be the observed output.
Adjust the weights proportional to the difference between y and y_a.

- Learning time = # of examples × size of the network.

PAC (Probably Approximately Correct) Learning

Let *C* be the concept that we are interested in learning. In general, we learn a concept *C'* that is an approximation to *C*.
The "distance" between *C* and *C'* should be "small" -> Approximation
We should be able to show that $dist(C, C')$ is $\leq \epsilon$ with a probability of $\geq (1 - \delta)$, where ϵ and δ are user specified parameters. The learning time should be a polynomial in $n, \frac{1}{\epsilon}$, and $\frac{1}{\delta}$ where *n* is the number of examples.

Example 1:
Concept: C->An axes aligned rectangle R
Examples: Points within the rectangle
Output: the smallest rectangle R' that includes all the example points



The area in red is the area R' misses.
We want to show that the area missed is very small.

Let ϵ be the difference between R and R' in area.
Let m be the number of samples (i.e., examples).
The difference between R and R' will be $\geq \epsilon$ if all the m samples come from an area of $\leq (1 - \epsilon)$.
Probability of this happening is $\leq (1 - \epsilon)^m$
We want this to be $\leq \delta$.

$$(1-\epsilon)^m = \delta$$
$$\Rightarrow m \log(1-\epsilon) = \log \delta$$
$$\Rightarrow m = \frac{\log\left(\frac{1}{\delta}\right)}{\log\left(\frac{1}{1-\epsilon}\right)}$$

. Thus we get to know how many samples will be enough to satisfy the requirements of the user.

Recall:

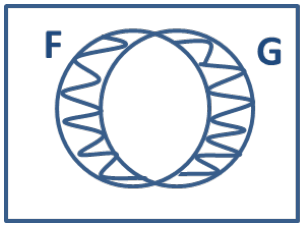
A Boolean formula is in Conjunctive Normal Form (CNF) if it is a conjunction of disjunctions.
Ex. $F(x_1, x_2, x_3) = (x_1 + \overline{x_2})(x_3 + x_1)$

Definition: A *k*-CNF formula is a CNF formula with ≤*k* literals in each clause.
Definition: A monomial is a 1 CNF formula.
Ex. $F = x_1\overline{x_2}x_3\overline{x_4}$

Learning a monomial: From positive examples
Let *F* be on n variables $x_1, x_2, \ldots, x_n$;
Start with $G = x_1\overline{x_1}x_2\overline{x_2} \ldots x_n\overline{x_n}$;
We have m examples.
Each example is an assignment to the variables that satisfies *F*.
For each example do:
If $x_i = 1$ then remove $\overline{x_i}$ from G;
If $x_i = 0$ then remove x_i from G;
After processing all the examples, output G.
Let D be a distribution on the set of all assignments to the n variables.

$$dist(F,G) = \sum_{\substack{\{v: v \Rightarrow F \text{ and } v \not\Rightarrow G \text{ or } \\ v \Rightarrow G \text{ and } v \not\Rightarrow F\}}} D(v)$$

"⇒" means satisfies.



Let m be the number of samples.
$Prob[dist(F,G) > \epsilon]$ for all the m samples is $\leq (1 - \epsilon)^m$.

The number of possible concepts is $\leq 2^{2n}$.

Probability that $[dist(F, G) > \epsilon]$ holds for any of 2^{2n} concepts is $\leq 2^{2n}(1 - \epsilon)^m$.
We want this to be $\leq \delta$

$$2^{2n}(1 - \epsilon)^m = \delta$$
$$2n + m \log(1 - \epsilon) = \log(\delta)$$
$$m = \frac{2n + \log\left(\frac{1}{\delta}\right)}{\log\left(\frac{1}{1-\epsilon}\right)}.$$

Note that the learning time is O(*mn*).