We have a make-up class 405-455 this Friday.

Previous class perceptions

\[ x_1, w_1 \rightarrow \Lambda \rightarrow 0 \]
\[ x_2, w_2 \rightarrow \Lambda \rightarrow 1 \]
\[ x_3, w_3 \rightarrow \Lambda \rightarrow 1 \]

- an example of the class (negative)

\[ x_1w_1 + x_2w_2 + 1w_3 \geq 0 \]
\[ \geq 0 \] 0

- an example of the class

\[ x_1w_1 + x_2w_2 + 1w_3 < 0 \]
\[ < 0 \] 0

\[ x_2 \rightarrow \gamma \rightarrow \text{cr} \]

\[ x_1 \rightarrow \gamma \rightarrow \text{cr} \]

(1,1) should output 1

(0,0) should output 0
AND

\[
\begin{align*}
\exists \theta \\
\theta &> 0 \\
\text{or} \\
\theta &< 0
\end{align*}
\]

XOR

Each perceptron has its bias input.

Layer

Error comes from inputs weights.
Run thru all the examples until there are no errors.

converge: all examples are classified correctly.
sigmoid( \text{lin} ) \\
\text{weights sum}

g(x) = \frac{1}{1 + e^{-x}}

\text{sigmoid}

g'(x) = g(x)(1 - g(x))
CS4811 Neural Network Training Example

Consider the following network. It has two inputs (two entries in each input vector), one hidden layer with two neurons and the output layer with a single neuron. Each neuron has a bias input to allow threshold values other than 0.

Each neuron is connected to all the neurons in the next layer, there are no back connections (this is a feedforward network).

Each neuron has an IN value that represents the weighted sum of its inputs. The neurons in the input layer simply output their input, they don’t have weights coming in. That is why they are drawn as dashed lines. Each neuron also has a D value that represents the delta value that will be used for back propagation. Notice that the nodes in the input layer do not have Ds associated because they don’t have incoming weights.

Let’s assume that this network is going to learn the NXOR function. So there are 4 examples. For simplicity we will not be showing the bias in the examples, we will always assume it is 1 for all the neurons. The training examples are the following:

\[
\begin{align*}
    x1 = 1 & \quad x2 = 0 & \quad \text{desired}=0 \\
    x1 = 0 & \quad x2 = 0 & \quad \text{desired}=1 \\
    x1 = 0 & \quad x2 = 1 & \quad \text{desired}=0 \\
    x1 = 1 & \quad x2 = 1 & \quad \text{desired}=1
\end{align*}
\]
The backpropagation algorithm

The following is the backpropagation algorithm for learning in multilayer networks.

**function** BACK-PROP-LEARNING(examples, network)
**returns** a neural network

**inputs:**
- examples, a set of examples, each with input vector \(x\) and output vector \(y\).
- network, a multilayer network with \(L\) layers, weights \(W_{j,i}\), activation function \(g\)

**local variables:** \(\Delta\), a vector of errors, indexed by network node

**for each** weight \(w_{i,j}\) in network **do**
\(w_{i,j} \leftarrow \) a small random number

**repeat**
**for each** example \((x,y)\) in examples **do**

/* Propagate the inputs forward to compute the outputs. */
**for each** node \(i\) in the input layer **do**
\(a_i \leftarrow x_i\) // Simply copy the input values.

**for** \(i = 2\) to \(L\) **do**
**for each** node \(j\) in layer \(l\) **do**
\(in_j \leftarrow \sum_i w_{i,j} a_i\)
\(a_j \leftarrow g(in_j)\) // Feed the values forward.

**for each** node \(j\) in the output layer **do**
\(\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)\) // Compute the error at the output.

/* Propagate the deltas backward from output layer to input layer */
**for** \(l = L - 1\) to \(1\) **do**
**for each** node \(i\) in layer \(l\) **do**
\(\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]\) // “Blame” a node as much as its weight.

/* Update every weight in network using deltas. */
**for each** weight \(w_{i,j}\) in network **do**
\(w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]\) // Adjust the weights.

until some stopping criterion is satisfied

**return** network

\(\alpha : \) learning constant
\(0.1 \ 0.5\)
$y = x^2$

$x, y$