CS4811 Neural Network Learning Algorithms

From: Stuart Russell and Peter Norvig
Artificial Intelligence a Modern Approach
Prentice Hall Series in Artificial Intelligence, 2003.

Single perceptron learning

The following is a gradient descent learning algorithm for perceptrons, assuming a differentiable activation function g. For threshold perceptrons, the factor $g'(in)$ is omitted from the weight update. NEURAL-NET-HYPOTHESIS returns a hypothesis that computes the network output for any given example. Comments were added to the algorithm listed in Figure 20.21.

function PERCEPTRON-LEARNING(*examples, network*)
returns a perceptron hypothesis

inputs:
    examples, a set of examples, each with input $\mathbf{x} = x_1, \ldots, x_n$ and output y
    network, a perceptron with weights $W_j,\ j = 0, \ldots n$ and activation function g
repeat
    for each e **in** *examples* **do**
        $in \leftarrow \sum_{j=0}^{n} W_j\, x_j[e]$ // Compute the weighted sum.
        $err \leftarrow y[e] - g(in)$ // Compute the error.
        $W_j \leftarrow W_j + c \times Err \times g'(in) \times x_j[e]$ // Adjust the weights.
until some stopping criterion is satisfied
return NEURAL-NET-HYPOTHESIS(*network*)

Note that $x_1, \ldots, x_n$ are the real inputs and x_0 is the bias input which is always -1. We'll take $g'(in)$ to be 1 for simplicity.

The stopping criterion can be a combination of the following:

- Convergence: The algorithms stops when every example is classified correctly.
- Number of iterations: The algorithm stops when a preset iteration limit is reached. This puts a time limit in case the network does not converge.
- Inadequate progress; The algorithm stops when the maximum weight change is less than a preset ε value. The procedure can find a minimum squared error solution even when the minimum error is not zero.

The backpropagation algorithm

The following is the backpropagation algorithm for learning in multilayer networks. Comments were added to the algorithm in Figure 20.25.

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
repeat
  for each e **in** *examples* **do**
    for each node j in the input layer **do** // Simply copy the input values.
      $a_j \leftarrow x_j[e]$
    for $l = 2$ to L **do** // Feed the values forward.
      $in_i \leftarrow \sum_j W_{j,i}\, a_j$ // j refers to the previous layer.
      $a_i \leftarrow g(in_i)$ // i refers to the current layer (l).
    for each node i in the output layer **do** // Compute the error at the output.
      $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
    for $l = L-1$ **to** 1 **do** // Propagate the error backwards.
      for each node j in layer l **do**
        $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$ // "Blame" a node as much as its weight.
        for each node i in layer $l+1$ **do**
          $W_{j,i} \leftarrow W_{j,i} + c \times a_j \times \Delta_i$ // Adjust the weights.
until some stopping criterion is satisfied
return NEURAL-NET-HYPOTHESIS(*network*)

For g, use the hyperbolic tangent: $tanh(x)$. The derivative of $tanh$ is $sech^2$, so use $sech^2(x)$ for g'.

$$tanh(x) = \frac{sinh(x)}{cosh(x)}$$

$$sech(x) = \frac{1}{sinh(x)}$$

.