Perceptron learning

Learn by adjusting weights to reduce error on training set

$x = (x_1, x_2, \ldots, x_n)$

The squared error for an example with input $\mathbf{x}$ and true output y is

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 ,$$

squared difference between what the network computes and what it should be.

Perform optimization search by gradient descent:

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g(\underbrace{\textstyle\sum_{j=0}^{n} W_j x_j}_{in})\right)$$
$$= -Err \times g'(in) \times x_j$$

Search for the best weights, in order to reduce E.

Simple weight update rule:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

E.g., +ve error $\Rightarrow$ increase network output

$\Rightarrow$ increase weights on +ve inputs, decrease on -ve inputs

Very much in the spirit of Samuel.

α is the learning rate.

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**
      $in \leftarrow \sum_{j=0}^{n} W_j \, x_j[e]$
      $Err \leftarrow y[e] - g(in)$
      $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$
  until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS(*network*)

Figure 20.21 The 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.

Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$ for outputs

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$ for hidden nodes

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

back propagation

(Most neuroscientists deny that back-propagation occurs in the brain)