Development of the BP Training Algorithm

Error

The activation value of a hidden unit is given by

\[ b_i = f\left(\sum_h v_{hi}a_h\right) \]

where \( b_i \) is the activation value of a hidden layer unit.

The activation value (calculated) for an output unit is

\[ c_j = f\left(\sum_i w_{ij}b_i\right) = f\left(\sum_i w_{ij}f\left(\sum_h v_{hi}a_h\right)\right) \]

where \( c_j \) is the activation (calculated) value of an output layer unit.

The error or discrepancy between the calculated and desired value of an output layer unit can be defined as:

\[
E[w] = \frac{1}{2} \sum_j [c^k_j - c_j]^2 = \frac{1}{2} \sum_j [c^k_j - f(\sum_i w_{ij}f(\sum_h v_{hi}a_h))]^2
\]

This is a **continuous, differentiable** function and, therefore, we can perform **gradient descent**.

From Hidden to Output

First, let us look at the weight changes on the connections from the \( F_B \), or hidden, layer to the \( F_C \), or output, layer.

\[
\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}} = \eta \sum_j [c^k_j - c_j] f'(\sum_i w_{ij}b_i)b_i = \eta \sum_j d_j b_i
\]

where

\[ d_j = f'(\sum_i w_{ij}b_i)(c^k_j - c_j) \quad (1) \]

In summary, the weight change can be written as

\[ \Delta w_{ij} = \alpha d_j b_i \]

If the threshold function is the sigmoid \( f(x) = \frac{1}{1+e^{-x}} \), then the derivative of this function can be expressed in terms of itself as \( d_j = c_j(1 - c_j)(c^k_j - c_j) \) and so the change in connection weight can be expressed as

\[ \Delta w_{ij} = \alpha[c_j(1 - c_j)(c^k_j - c_j)]b_i \]

1
From Input to Hidden

Now we will look at the weight changes on the connections from the $F_A$, or input layer, to the $F_B$, or hidden, layer. To do this we must differentiate $E[w]$ with respect to $v_{hi}$. Using the chain rule:

$$
\Delta v_{hi} = -\eta \frac{\delta E}{\delta v_{hi}} = -\eta \sum_p \delta b_i \delta v_{hi} \\
= \eta \sum_p \left[ c_j^k - c_j \right] f'(\sum_i w_{ij} b_i) w_{ij} f'(\sum_h v_{hi} a_h) a_h \\
= \eta \sum_p d_j w_{ij} f'(\sum_h v_{hi} a_h) a_h \\
= \eta \sum_p d_i a_h
$$

where

$$
d_i = f'(\sum_h v_{hi} a_h) \sum_i w_{ij} d_j \quad (2)
$$

Thus, the weight change is $\Delta v_{hi} = \beta a_h d_i$; once again, using the logistic threshold function:

$$
d_i = b_i (1 - b_i) \sum_i w_{ij} d_j
$$

and

$$
\Delta v_{hi} = \beta a_h [b_i (1 - b_i) \sum_i w_{ij} c_j (1 - c_j) (c_j^k - c_j)]
$$

In General...

The general form of a weight change is

$$
\Delta w_{pq} = \eta \sum_{patterns} d_{OUTPUT} \times V_{INPUT}
$$

where $d_{OUTPUT}$ depends on the layer

- last layer uses Equation (1)
- all other layers use Equation (2)

and $V_{INPUT}$ represents the appropriate input-end activation
The Threshold Function  

Now let us examine the threshold function in detail. The general form of the logistic function is

\[ f_\beta(x) = \frac{1}{1 + e^{-2\beta x}} \]

where \( \beta \) is a steepness parameter (often \( \frac{1}{2} \) or 1). The derivative of this function is

\[ f'_\beta(x) = 2\beta f(1 - f) \]

Now if the steepness parameter is \( \frac{1}{2} \) then \( f'(x) = f(1 - f) = c_j(1 - c_j) \)

The general form of the hyperbolic tangent function is

\[ f_\beta(x) = tanh\beta x \]

The derivative of this function is

\[ f'_{\beta}(x) = \beta(1 - f^2) \]

If \( \beta = 1 \) then \( f'(x) = (1 - c_j^2) \)

Local Minima
- have not been much of a problem in most cases (empirical evidence)
- often the bottoms of very shallow steep-sided valleys
- to avoid, choose patterns in a random order which generates “useful” noise

Alternative Cost Functions
- can replace the \([c_j^k - c_j]^2\) term in the quadratic cost function by another differentiable function \( F(c_j^k, c_j) \) that is minimized when its arguments are equal
- derive a corresponding update rule
  - only \( d_j \) in the output layer changes
  - all other equations remain unchanged

Newton’s Method
- the Hessian matrix

\[ H_{ij} = \frac{\delta^2 E}{\delta x_i \delta x_j} \]

where the vector \( x \) represents the weight space and specifying \( x \) corresponds to specifying all the weights.
Adaptive Parameters

- hard to choose appropriate learning/momentum rates
- best values at beginning may not be good later on
  - adjust the parameters automatically as learning progresses

Standard Approach

- check if a weight update actually decreases the cost function
  - if it did not (overshot) then reduce $\eta$
  - if several consecutive steps decreases the cost then increase $\eta$
- increase $\eta$ by a constant
- decrease $\eta$ geometrically to allow rapid decay

\[
\Delta \eta = \begin{cases} 
+\alpha & \text{if } \Delta E < 0 \text{ consistently} \\
-\beta \eta & \text{if } \Delta E > 0 \\
0 & \text{otherwise}
\end{cases}
\]

where consistently can be

- based on the last $K$ steps
- weighted moving average of observed $\Delta E$'s

Other Adaptive Schemes

- several learning rates
  - parameters $\eta^p$, one for each pattern $p$
    
  - an $\eta_{pq}$ for each connection $pq$
    

- different learning rates for different architectures
  - $\eta_{pq} \propto \frac{1}{\text{(fan-in of site i)}}$
    
Genetic Algorithm Strategy

- use of a GA to search the weight space without use of any gradient information
  - a complete set of weights is coded in a binary string (chromosome) which has an associated fitness that depends on its effectiveness
  - starting with a random population of strings, successive generations are constructed using genetic operators such as mutation and crossover
  - “fitter” strings are more likely to survive and mate
  - the encoding methods and the genetic operators are crucial to the effectiveness of this technique

- GAs perform a global search and thus are not easily fooled by local minima
  - fitness function does not have to be differentiable
  - but, high computational penalty
    * initial genetic search followed by gradient methods
    * gradient descent step used as one of the genetic operators

Initial Weights

- size of initial random weights is important
  - if too large, then sigmoids saturate quickly and the system becomes stuck in a local minimum (flat plateau) near the starting point

- one strategy is to choose the random weights so that the magnitude of the typical net input to a PE is less than (but not too much less than) unity
  - weights $w_{ij}$ will be of the order $\frac{1}{\sqrt{k_i}}$ where $k_i$ is the number of $j$’s which feed forward to unit $i$ (the fan-in of unit $i$)