Error-correcting learning: Delta rule

Change weights so as to reduce difference between actual output \( a_j \) and target output (denoted \( t_j \))

\[
\Delta w_{ij} = \epsilon (t_j - a_j) a_i
\]

- “Delta”: difference between output and target
- Also called Widrow-Hoff rule, LMS (least mean squared)
- Related to perceptron convergence procedure (Rosenblatt)

- Similar to correlation with error
- Hebb rule: \( \Delta w_{ij} = \epsilon t_j a_i \) (where \( t_j \) is activation “clamped” on the output unit)

Weight changes focus on predictive differences
- Hebbian/correlational learning depends on predictive similarities

Learning on orthogonal patterns (one pass): Delta = Hebb

Delta rule: \( \Delta w_{ij} = \epsilon (t_j - a_j) a_i \) (assume linear units: \( a_j = n_j \); Note: Delta = Hebb if \( a_j = 0 \))

For first pattern \( p_1 \), \( w_{ij} = 0 \) so \( a_j(p_1) = n_j(p_1) = 0 \), and

\[
w_{ij} = \Delta w_{ij} = \epsilon (t_j(p_1) - 0) = t_j(p_1) a_i(p_1)
\]

For \( p_2 \), \( a_j(p_2) = \sum_i a_i(p_2) w_{ij} = \sum_i a_i(p_2) t_j(p_1) a_i(p_1) = t_j(p_1) \sum_i a_i(p_2) a_i(p_1) \sum_i a_i(p_2) a_i(p_1) \) (dot product of \( p_i \) and \( p_o \)).

Since \( p_1 \) and \( p_2 \) are orthogonal, \( \sum_i a_i(p_2) a_i(p_1) = 0 \), so \( a_j(p_2) = 0 \). Thus

\[
\begin{align*}
\Delta w_{ij} & = t_j(p_2) a_i(p_1) \\
w_{ij} & = t_j(p_1) a_i(p_1) + t_j(p_2) a_i(p_2)
\end{align*}
\]

In fact, \( a_j(p) = 0 \) for the first presentation of each training pattern \( p \), so at the end of one sweep through all the patterns:

\[
w_{ij} = \epsilon \sum_p (t_j(p) - a_j(p)) a_i(p) = \epsilon \sum_p t_j(p) a_i(p)
\]

This is just Hebbian learning using targets \( t_j \) as output activations \( a_j \).

Note that the Delta rule is inherently multi-pass
- In general, \( a_j \neq 0 \) on subsequent presentations of each training pattern
- Weight changes due to one pattern affect error on others

Effects of training on response to input patterns

Calculated in terms of changes to activations for pattern \( p' \) caused by training on single pattern \( p \):

\[
\Delta a_j(p') = \sum_i a_i(p') \Delta w_{ij} = \sum_i a_i(p') \epsilon (t_j(p) - a_j(p)) a_i(p) = \epsilon (t_j(p) - a_j(p)) \sum_i a_i(p') a_i(p)
\]

- If \( p \) and \( p' \) are orthogonal, training on \( p \) will have no effect on \( p' \)
- If \( p \) and \( p' \) are not orthogonal, training on \( p \) will affect performance on \( p' \) (weighted by similarity) which may be good (generalization) or bad (interference)

Delta rule as gradient descent in error (linear units)

\[
\begin{align*}
a_j & = \sum_i a_i w_{ij} \\
\text{Error} & = \frac{1}{2} \sum_j (t_j - a_j)^2 \\
\text{Gradient descent:} & \quad \Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}} = -\epsilon \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \\
& = -(t_j - a_j) a_i
\end{align*}
\]

\[
\begin{align*}
\frac{\partial E}{\partial w_{ij}} & = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \\
& = \epsilon (t_j - a_j) a_i
\end{align*}
\]

\[
\Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}} = \epsilon (t_j - a_j) a_i \quad \text{Delta rule}
\]
Delta rule as gradient descent in error (sigmoid units)

\[ n_j = \sum_i a_i w_{ij} \]

\[ a_j = \frac{1}{1 + \exp(-n_j)} \]

Error \( E = \frac{1}{2} \sum_j (t_j - a_j)^2 \)

Gradient descent:
\[ \Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}} \]

\[ \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j} \cdot \frac{\partial a_j}{\partial n_j} \cdot \frac{\partial n_j}{\partial w_{ij}} \]

\[ = -(t_j - a_j) \cdot a_j (1 - a_j) \cdot a_i \]

\[ \Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}} = \epsilon (t_j - a_j) \cdot a_j (1 - a_j) \cdot a_i \]

When does the Delta rule succeed or fail?

Delta rule is optimal
- Will find a set of weights that produces zero error if such a set exists

Need to distinguish “succeed” = zero error from “succeed” = correct binary classification

Guaranteed to succeed (zero error) if input patterns are linearly independent (LI)
- No pattern can be created by recombining scaled versions of the others
  (i.e., there is something unique about each pattern; cf. Hebb: no similarity)
- Orthogonal patterns are linearly independent (LI is a weaker constraint)
- Linearly independent patterns can be similar as long as other aspects are unique

Succeed at binary classification of outputs: Linear separability

Linear separability

Delta rule is guaranteed to succeed at binary classification if the task is linearly separability

- Weights define a plane (line for two input units) through input (state) space for which \( n_j = 0 \)
- Must be possible to position this plane such that all patterns requiring \( n_j < 0 \) are on one side and all patterns requiring \( n_j > 0 \) are on the other side
- Property of the relationship between input and target patterns
- AND and OR are linearly separable but XOR is not

\[ n_j = a_1 w_1 + a_2 w_2 + b_j = 0 \]

\[ a_2 = -\frac{w_1}{w_2} a_1 - \frac{b_j}{w_2} \]

\( (y = a \cdot x + b) \)

XOR

\[ n_j = a_1 w_1 + a_2 w_2 + b_j = 0 \]

\[ a_2 = -\frac{w_1}{w_2} a_1 - \frac{b_j}{w_2} \]

\( (y = a \cdot x + b) \)
**XOR with extra dimension**

XOR task can be converted to one that is linearly separable by adding a new “input”

- Corresponds to a third dimension in state space
- Task is no longer XOR

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

**XOR with intermediate (“hidden”) units**

- Intermediate units can re-represent input patterns as new patterns with altered similarities
- Targets which are not linearly separable in the input space can be linearly separable in the intermediate representational space
- Intermediate units are called “hidden” because their activations are not determined directly by the training environment (inputs and targets)