**Error-correcting learning: Delta rule**

Important distinction (and notation):
- \( t_j \) target of unit \( j \); the (correct) activation specified by the environment (training example)
- \( a_j \) activation of unit \( j \) that results from actually running the network

Note: in Hebb rule, \( a_j \) was specified and so would now be called \( t_j \)
- Hebb rule: \( \Delta w_{ij} = \epsilon (t_j - a_j) a_i \) (where \( t_j \) is activation “clamped” on the output unit)

**Delta rule:** Change weights so as to reduce difference between actual output (\( a_j \)) and target output (\( t_j \)) (“delta” = difference between target and activation)

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

- Similar to correlation with error (i.e., \( t_j - a_j \))
- 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: \( a_j = a_j \))

- 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

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

- Hebb rule with target as output activation

For \( p_2 \), \( a_j[p_2] = \sum_i a_j[p_2] w_{ij} = \sum_i a_j[p_2] (t_j[p_1] a_i[p_1]) = t_j[p_1] \sum_i a_j[p_2] a_i[p_1] \sum_i a_i[p_1] a_j[p_1] \) (dot product of \( p_1 \), \( p_2 \))

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

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

- Hebb rule again

In fact, \( a_j[p_1] = 0 \) for the first presentation of each training pattern \( p \), so at the end of one sweep through all the patterns:

\[
\hat{w}_{ij} = \epsilon \sum_p (t_j[p] - a_j[p]) a_j[p] = \epsilon \sum_p a_j[p] a_j[p] \]

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

Note that the Delta rule is inherently multi-pass (\( a_j \neq 0 \) on subsequent presentations)
- Weight changes caused by 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 \( p \):

\[
\Delta a_j[p'] = \sum_i a_j[p'] \Delta w_{ij}
\]

\[
= \sum_i a_j[p'] \epsilon (t_j[p] - a_j[p]) a_i[p]
\]

\[
= \epsilon (t_j[p] - a_j[p]) \sum_i a_j[p'] a_i[p]
\]

\[
= \epsilon (t_j[p] - a_j[p]) \Delta p(p', p)
\]

- 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)

**Weight space (analogous to state space)**

- High-dimensional space with a dimension for each weight in the network
- Any given possible set of weights corresponds to a particular point in the space (coordinates are weight values)
- Extra dimension for error that results from assigning those weight values, running all the examples, and summing up all the error
- Small changes in weights cause small changes in error: error values form a continuous surface above weight space
Delta rule as gradient descent in error (linear units)

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

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

Gradient descent:
\[ \Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}} \]
\[ = -\epsilon \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \]
\[ = -\epsilon \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \]
\[ = -\epsilon (t_j - a_j) a_i \]

Delta rule as gradient descent in error (sigmoid units)

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

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

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

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

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 separable
- 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

\[ n_j = a_1 w_1 + a_2 w_2 + b_j = 0 \]
\[ a_2 = -\frac{w_1}{w_2} a_1 - \frac{b_j}{w_2} \]
\( y = a x + b \)
XOR

\[ n_j = a_1 w_1 + a_2 w_2 + b_j = 0 \]
\[ a_2 = -\frac{w_1}{w_2} \frac{a_1 - b_j}{w_2} \]
\[ (y = a x + b) \]

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 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>1</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)