Simple Perceptrons
Simple Perceptrons

- Perform supervised learning
  - correct I/O associations are provided
- Feed-forward networks
  - connections are one-directional
- One layer
  - input layer + output layer
Notations

- $N$: dimension of the input vector
- $M$: dimension of the output vector
- inputs $x_j, j=1,...,N$
- real outputs $y_i, i=1,...M$
- weight vectors $w_{ij}, i=1,...,M$, $j=1,...,N$
- activation function $g$

$$y_i = g(\text{net}_i) = g\left(\sum_{j=1}^{N} w_{ij} x_j\right)$$
Perceptron Training

- given
  - input patterns $x^u$
  - desired output patterns $O^u$
  - how to adapt the connection weights such that the actual outputs conform the desired outputs

$$O_i^u = y_i^u \quad i = 1, \ldots, M$$
Simplest case

- two types of inputs
- binary outputs (-1,1)
- thresholding \( \text{sgn}(w \cdot x) = \text{sgn}(w_1 x_1'' + w_2 x_2'' + w_o) = y'' \)
Examples

\[
g(\text{net}) = \text{sgn}(w_1 x_1^u + w_2 x_2^u - 1.5)
\]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$O$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
w_0 & = 1.5 \\
w_1 & = 1 \\
w_2 & = 1
\end{align*}
\]
Linear separability

- Is it at all possible to learn the desired I/O associations?
  - yes, if \( w_{ij} \) can be found such that
    \[
    O_i^{"} = \text{sgn}(\sum_{j=1}^{N} w_{ij}x_j^{"} - w_{i0}) = y_i^{"}
    \]
    for all \( i \) and \( u \)
  - no, otherwise

- Single-layer perceptron is severely limited in what it can learn
Perceptron Learning

- Linear separable or not, how to find the set of weights?
- Using tagged samples
  - closed form solution
  - iterative solutions
Closed Form Solution

\[
\begin{bmatrix}
    x_1^1 & \ldots & x_n^1 & 1 \\
    x_1^2 & \ldots & x_n^2 & 1 \\
    \vdots & \ddots & \vdots & \vdots \\
    x_1^u & \ldots & x_n^u & 1
\end{bmatrix}
\begin{bmatrix}
    w_1 \\
    w_2 \\
    \vdots \\
    w_o
\end{bmatrix}
= 
\begin{bmatrix}
    O_1^1 \\
    O_2^2 \\
    \vdots \\
    O_o^u
\end{bmatrix}
\]

\[AW = B\]

\[W = (A^T A)^{-1} A^T B\]

- Not practical when number of samples is large (most likely case)
Perceptron Learning Rule

- If a pattern is correctly classified, no action
Perceptron Learning Rule (cont.)

- If a positive pattern becomes a negative pattern
Perceptron Learning Rule (cont.)

- If a negative pattern becomes a positive pattern
\[ w^{(k+1)} = \begin{cases} 
 w^{(k)} + cx & w^{(k)} \cdot x < 0, x \in + \\
 w^{(k)} - cx & w^{(k)} \cdot x > 0, x \in - \\
 w^{(k)} & \text{otherwise} 
\end{cases} \]

\[ w^{(k+1)} = \begin{cases} 
 w^{(k)} + cyx & y(w^{(k)} \cdot x) < 0, x \in + or - \\
 w^{(k)} & \text{otherwise} 
\end{cases} \]

- How should \( c \) be decided?
  - Fixed increment
  - Fractional correction
**Perceptron Learning Rule (cont.)**

- Weight is a signed, linear combination of training points
- Use those informative points (those the classifier made a mistake, mistake driven)
- This is VERY important, lead later to generalization to Support Vector Machines
Comparison

- Version space
  - The \((w_1-w_2)\) space of all feasible solutions
- Perceptron learning
  - Greedy, gradient descent that often ends up at boundary of the version space with little space for error
- SVM learning
  - Center of largest imbedded sphere in the version space (maximum margin)
- Bayes point machine
  - Centroid of the version space
Perceptron Usage Rules

- After the weight has been determined
  \[ y = w \cdot x = \left( \sum_{i} \alpha_i y_i x_i \right) \cdot x = \sum_{i} \alpha_i y_i x_i \cdot x \]

- Classification involves inner product of training samples and test samples

- This is again VERY important, lead later to generalization to Kernel Methods
Hebb’s Learning Rule

- Synapse strength should be increased when both pre- and post-synapse neurons fire vigorously.

  - for binary outputs
    
    \[
    w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}
    \]
    
    \[
    \Delta w_{ij} = \begin{cases} 
    2\eta y_i^u x_j^u & \text{if } y_i^u \neq O_i^u \\
    0 & \text{otherwise}
    \end{cases}
    \]
    
    \[
    = \eta (1 - y_i^u O_i^u) y_i^u x_j^u 
    \]
    
    \[
    = \eta (y_i^u - y_i^u O_i^u) x_j^u 
    \]
    
    \[
    = \eta \delta x_j^u
    \]
Case 1: \( O = -1, y = 1 \)  
Case 2: \( O = 1, y = -1 \)
LMS (Widrow-Hoff, Delta)

- Not restricted to binary outputs
- Gradient search

\[
E(w) = \frac{1}{2} \sum_{u} \sum_{i} (O_i^u - y_i^u)^2 = \frac{1}{2} \sum_{u} \sum_{i} (O_i^u - g(\sum_{j=1}^{N} w_{ij} x_j^u))^2
\]

\[
\frac{\partial E(w)}{\partial w_{ij}} = -\sum_{u} (O_i^u - g(\text{net}_i^u)) g'(\text{net}_i^u)x_j^u
\]

\[
W_{ij}^{new} = W_{ij}^{old} + \Delta W_{ij}
\]

\[
= W_{ij}^{old} + \eta \sum_{u} (O_i^u - g(\text{net}_i^u)) g'(\text{net}_i^u)x_j^u
\]
Nothing but Chain Rule

\[
\frac{\partial E(w)}{\partial w_{ij}} = \frac{\partial (O_i^u - y_i^u)^2}{\partial (O_i^u - y_i^u)} \frac{\partial (O_i^u - y_i^u)}{\partial y_i^u} \frac{\partial y_i^u}{\partial net_i^u} \frac{\partial net_i^u}{\partial w_{ij}}
\]

\[
= \sum_u (O_i^u - g(net_i^u)) g'(net_i^u) x_j^u
\]
\[ O = g(\text{net}) = \text{sgn}(w_1 x_1 + w_2 x_2 + b) \]

\[ w_1 = 0.4299 \]
\[ w_2 = -0.2793 \]
\[ b = -0.1312 \]
Final training results  Error vs. training epoch
\[ y = g(\text{net}) = \text{sgn}(w_1x_1 + w_2x_2 + w_3x_3 + b) \]

\[ w_1 = 0.4232 \]
\[ w_2 = -0.7411 \]
\[ w_3 = -0.3196 \]
\[ b = 0.7550 \]
Final training results

Error vs. training epoch
\[ y_1 = g(w_{11}x_1 + w_{12}x_2 + b_1) \]
\[ y_2 = g(w_{21}x_1 + w_{22}x_2 + b_2) \]
Final training results

Error vs. training epoch
Final training results

Error vs. training epoch