Artificial Neural Networks (ANN)

Output $Y$ is 1 if at least two of the three inputs are equal to 1.
Artificial Neural Networks (ANN)

<table>
<thead>
<tr>
<th>X₁</th>
<th>X₂</th>
<th>X₃</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Black box

\[ Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4) \]

where \( \text{sign}(x) = \begin{cases} 1 & \text{if} \ x \geq 0 \\ -1 & \text{if} \ x < 0 \end{cases} \)

Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold \( t \)

Perceptron Model

\[ Y = \text{sign} \left( \sum_{i=1}^{d} w_i X_i - t \right) \]

\[ = \text{sign} \left( \sum_{i=0}^{d} w_i X_i \right) \]
Artificial Neural Networks (ANN)

- Various types of neural network topology
  - single-layered network (perceptron) versus multi-layered network
  - Feed-forward versus recurrent network

- Various types of activation functions \((f)\)

\[
Y = f\left(\sum_{i} w_i X_i\right)
\]
Perceptron

- Single layer network
  - Contains only input and output nodes

- Activation function: \( f = \text{sign}(w \cdot x) \)

- Applying model is straightforward
  \[
  Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)
  \]
  where \( \text{sign}(x) = \begin{cases} 
    1 & \text{if } x \geq 0 \\
    -1 & \text{if } x < 0 
  \end{cases} \)

- \( X_1 = 1, X_2 = 0, X_3 = 1 \Rightarrow y = \text{sign}(0.2) = 1 \)

Perceptron Learning Rule

- Initialize the weights \((w_0, w_1, \ldots, w_d)\)

- Repeat
  - For each training example \((x_i, y_i)\)
    - Compute \(f(w, x_i)\)
    - Update the weights:
      \[
      w^{(k+1)} = w^{(k)} + \lambda y_i - f(w^{(k)}, x_i) x_i
      \]

- Until stopping condition is met
**Perceptron Learning Rule**

- Weight update formula:
  \[
  w^{(k+1)} = w^{(k)} + \lambda \left[ y_i - f(w^{(k)}, x_i) \right] x_i \; ; \; \lambda : \text{learning rate}
  \]

- Intuition:
  - Update weight based on error: \( e = [y_i - f(w^{(k)}, x_i)] \)
  - If \( y=f(x,w) \), \( e=0 \): no update needed
  - If \( y>f(x,w) \), \( e=2 \): weight must be increased so that \( f(x,w) \) will increase
  - If \( y<f(x,w) \), \( e=-2 \): weight must be decreased so that \( f(x,w) \) will decrease

---

**Example of Perceptron Learning**

\[
    w^{(k+1)} = w^{(k)} + \lambda \left[ y_i - f(w^{(k)}, x_i) \right] x_i
\]

\[
    Y = \text{sign} \left( \sum_{i=0}^{d} w_i x_i \right)
\]

\( \lambda = 0.1 \)

<table>
<thead>
<tr>
<th>( X_1 )</th>
<th>( X_2 )</th>
<th>( X_3 )</th>
<th>( Y )</th>
<th>( w_0 )</th>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-0.2</td>
<td>-0.2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>5</td>
<td>-0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>-0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>8</td>
<td>-0.2</td>
<td>0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Epoch</th>
<th>( w_0 )</th>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-0.2</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>-0.2</td>
<td>0</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>-0.4</td>
<td>0</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>-0.6</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>6</td>
<td>-0.6</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
</tr>
</tbody>
</table>
Perceptron Learning Rule

- Since \( f(w,x) \) is a linear combination of input variables, decision boundary is linear.

- For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly.

Nonlinearly Separable Data

\[ y = x_1 \oplus x_2 \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

XOR Data
**Multilayer Neural Network**

- Hidden layers
  - intermediary layers between input & output layers

- More general activation functions (sigmoid, linear, etc)

---

**Multi-layer Neural Network**

- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces

---

**XOR Data**
Learning Multi-layer Neural Network

- Can we apply perceptron learning rule to each node, including hidden nodes?
  - Perceptron learning rule computes error term $e = y - f(w, x)$ and updates weights accordingly
    - Problem: how to determine the true value of $y$ for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - Problem:
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

Gradient Descent for Multilayer NN

- Weight update: $w_j^{(k+1)} = w_j^{(k)} - \lambda \frac{\partial E}{\partial w_j}$
- Error function: $E = \frac{1}{2} \sum_{i=1}^{N} \left( t_i - f(\sum_j w_{ij} x_i) \right)$
- Activation function $f$ must be differentiable
- For sigmoid function:
  $w_j^{(k+1)} = w_j^{(k)} + \lambda \sum_i (t_i - o_i) o_i (1 - o_i) x_{ij}$
- Stochastic gradient descent (update the weight immediately)
**Gradient Descent for MultiLayer NN**

- For output neurons, weight update formula is the same as before (gradient descent for perceptron).

- For hidden neurons:

\[
\delta_{j}^{(k+1)} = \delta_{j}^{(k)} + \lambda o_{j} (1 - o_{j}) \sum_{\forall p \in P} \delta_{p}^{(k)} w_{jp} x_{pi}
\]

Output neurons: \(
\delta_{j} = o_{j} (1 - o_{j}) (t_{j} - o_{j})
\)

Hidden neurons: \(
\delta_{j} = o_{j} (1 - o_{j}) \sum_{\forall k} \delta_{k} w_{jk}
\)

---

**Design Issues in ANN**

- Number of nodes in input layer
  - One input node per binary/continuous attribute
  - \(k\) or \(\log_2 k\) nodes for each categorical attribute with \(k\) values

- Number of nodes in output layer
  - One output for binary class problem
  - \(k\) or \(\log_2 k\) nodes for \(k\)-class problem

- Number of nodes in hidden layer

- Initial weights and biases
**Characteristics of ANN**

- Multilayer ANN are universal approximators but could suffer from overfitting if the network is too large.
- Gradient descent may converge to local minimum.
- Model building can be very time consuming, but testing can be very fast.
- Can handle redundant attributes because weights are automatically learnt.
- Sensitive to noise in training data.
- Difficult to handle missing attributes.

**Recent Noteworthy Developments in ANN**

- Use in deep learning and unsupervised feature learning
  - Seek to automatically learn a good representation of the input from unlabeled data.
- Google Brain project
  - Learned the concept of a ‘cat’ by looking at unlabeled pictures from YouTube.
  - One billion connection network.