CS344: Introduction to Artificial Intelligence
(associated lab: CS386)

Pushpak Bhattacharyya
CSE Dept.,
IIT Bombay

Lecture 33: Error Backpropagation
31st March, 2011
Feedforward Network
Example - XOR

\[ w_1 = 1 \quad w_2 = 1 \]

\[ \theta = 0.5 \]

\[ x_1 x_2 \]

\[ x_1 \]

\[ x_2 \]
Backpropagation on feedforward network
Gradient Descent Technique

- Let $E$ be the error at the output layer

$$E = \frac{1}{2} \sum_{j=1}^{p} \sum_{i=1}^{n} (t_i - o_i)^2_j$$

- $t_i = \text{target output}; \quad o_i = \text{observed output}$

- $i$ is the index going over $n$ neurons in the outermost layer

- $j$ is the index going over the $p$ patterns (1 to $p$)

- Ex: XOR: $p=4$ and $n=1$
Weights in a FF NN

- $w_{mn}$ is the weight of the connection from the $n^{th}$ neuron to the $m^{th}$ neuron
- $E$ vs $\bar{W}$ surface is a complex surface in the space defined by the weights $w_{ij}$
- $-\frac{\delta E}{\delta w_{mn}}$ gives the direction in which a movement of the operating point in the $w_{mn}$ coordinate space will result in maximum decrease in error

\[ \Delta w_{mn} \propto -\frac{\delta E}{\delta w_{mn}} \]
Backpropagation algorithm

- Fully connected feed forward network
- Pure FF network (no jumping of connections over layers)
Gradient Descent Equations

\[ \Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}} \]  
\[ (\eta = \text{learning rate, } 0 \leq \eta \leq 1) \]

\[ \frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}} \]  
\[ (\text{net}_j = \text{input at the } j^{th} \text{ layer}) \]

\[ \frac{\delta E}{\delta net_j} = -\delta j \]

\[ \Delta w_{ji} = \eta \delta j \frac{\delta net_j}{\delta w_{ji}} = \eta \delta j o_i \]
Backpropagation – for outermost layer

\[
\delta_j = - \frac{\delta E}{\delta \text{net}_j} = - \frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta \text{net}_j} (\text{net}_j = \text{input at the } j^{th} \text{ layer})
\]

\[
E = \frac{1}{2} \sum_{p=1}^{m} (t_p - o_p)^2
\]

Hence, \( \delta_j = -(-(t_j - o_j)o_j(1-o_j)) \)

\[
\Delta w_{ji} = \eta(t_j - o_j)o_j(1-o_j)o_i
\]
Backpropagation for hidden layers

$\delta_k$ is propagated backwards to find value of $\delta_j$
Backpropagation – for hidden layers

\[ \Delta w_{ji} = \eta \delta_j o_i \]

\[ \delta_j = - \frac{\delta E}{\delta net_j} = - \frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} \]

\[ = - \frac{\delta E}{\delta o_j} \times o_j (1 - o_j) \]

\[ = - \sum_{k \in \text{next layer}} \left( \frac{\delta E}{\delta net_k} \times \frac{\delta net_k}{\delta o_j} \right) \times o_j (1 - o_j) \]

Hence, \[ \delta_j = - \sum_{k \in \text{next layer}} (-\delta_k \times w_{kj}) \times o_j (1 - o_j) \]

\[ = \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) \]
General Backpropagation Rule

- General weight updating rule:
  \[ \Delta w_{ji} = \eta \delta_j o_i \]

- Where
  \[ \delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer} \]
  \[ = \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i \quad \text{for hidden layers} \]
How does it work?

- Input propagation forward and error propagation backward (e.g. XOR)
Can Linear Neurons Work?

\[ h_1 = m_1(w_1 x_1 + w_2 x_2) + c_1 \]

\[ h_1 = m_1(w_1 x_1 + w_2 x_2) + c_1 \]

\[ Out = (w_5 h_1 + w_6 h_2) + c_3 \]

\[ = k_1 x_1 + k_2 x_2 + k_3 \]
**Note:** The whole structure shown in earlier slide is reducible to a single neuron with given behavior

\[ \text{Out} = k_1x_1 + k_2x_2 + k_3 \]

**Claim:** A neuron with linear I-O behavior can’t compute X-OR.

**Proof:** Considering all possible cases:

[assuming 0.1 and 0.9 as the lower and upper thresholds]

\[ m(w_{1.0} + w_{2.0} - \theta) + c < 0.1 \]

For (0,0), Zero class: \[ \Rightarrow c - m.\theta < 0.1 \]

\[ m(w_{2.1} + w_{1.0} - \theta) + c > 0.9 \]

For (0,1), One class: \[ \Rightarrow m.w_i - m.\theta + c > 0.9 \]
For (1,0), One class: \( m.w_1 - m.\theta + c > 0.9 \)

For (1,1), Zero class: \( m.w_1 - m.\theta + c > 0.9 \)

These equations are inconsistent. Hence X-OR can’t be computed.

**Observations:**

1. A linear neuron can’t compute X-OR.
2. A multilayer FFN with linear neurons is collapsible to a single linear neuron, hence **no a additional power due to hidden layer.**
3. Non-linearity is essential for power.