Outline of this lecture

- Review of Back-Propagation Algorithm
- Generalized Delta Rule
- System Identification: An example
- Introducing Momentum in Back-Propagation
- Adaptive learning rate
Multilayer Feed forward Network

- Has one or more hidden layers of computation units (or nodes).
- In FNN, connections are allowed between any layer to it’s succeeding layer and no connection is allowed between any layer to it’s preceding layer.
- Hidden layer gets input from input layer and gives output to next hidden layer or output layer after internal computation.
Revisit of Back-Propagation

Notations used in the derivation of the algorithm.

- $x$: $p \times 1$ input vector
- $h$: Weighted sum of input stimuli
- $v$: $m \times 1$ output vector of hidden layer
- $g$: Weighted sum of $v_j$
- $y$: $n \times 1$ output vector of output layer
- $w_{ij}$: Weight connecting $i^{th}$ unit of output layer and $j^{th}$ unit of hidden layer
- $w_{jk}$: Weight connecting $j^{th}$ unit of hidden layer and $k^{th}$ unit of input layer
Forward Phase:

Output of the $j^{th}$ neuron of the hidden layer is given by,

$$v_j = \Phi(h_j) = \frac{1}{1 + e^{-h_j}}$$

where $h_j = \sum_{k=1}^{p} w_{jk} x_k$.

Final Response (Actual Output) of the network is given by,

$$y_i = \Phi(g_i) = \frac{1}{1 + e^{-g_i}}$$

where $g_i = \sum_{j=1}^{m} w_{ij} v_j$.
Back-Propagation:

The Error function that we seek to minimize is:

\[ E(t) = \frac{1}{2} (y_i^d(t) - y_i(t))^2 \]

where \( y_i^d \) is the desired response of the \( i^{th} \) unit of the output layer. The update law of the weight is given by Gradient Descent rule,

\[ w_{ij}(t + 1) = w_{ij}(t) - \eta \frac{\partial E(t)}{\partial w_{ij}(t)} \]

where \( \eta \) is the learning rate
Contd...

Updating weights connecting output layer and hidden layer

Computation of $\frac{\partial E}{\partial w_{ij}}$:

$$\frac{\partial E(t)}{\partial w_{ij}(t)} = \sum_{i=1}^{n} \frac{\partial E_i(t)}{\partial y_i} \times \frac{\partial y_i}{\partial w_{ij}(t)}$$

where

$$\frac{\partial E(t)}{\partial w_{ij}(t)} = \frac{1}{2} \times 2(y_i^d - y_i) \times (-1)$$

$$\frac{\partial y_i}{\partial w_{ij}} = \frac{\partial y_i}{\partial g_i} \times \frac{\partial g_i}{\partial w_{ij}}$$
Update Law

Since \( \frac{\partial y_i}{\partial g_i} = y_i(1 - y_i) \) and \( \frac{\partial g_i}{\partial w_{ij}} = v_j \)

\[
\Delta w_{ij} = \eta(y_i^d - y_i)y_i(1 - y_i)v_j
\]

Therefore,

\[
w_{ij}(t + 1) = w_{ij}(t) + \eta(y_i^d - y_i)y_i(1 - y_i)v_j = w_{ij}(t) + \eta\delta_i v_j
\]

where \( \delta_i = y_i(1 - y_i)(y_i^d - y_i) \), is the error back propagated from the output layer.
Updating weights connecting hidden layer and input layer

Computation of $\frac{\partial E}{\partial w_{jk}}$:

$$\frac{\partial E(t)}{\partial w_{jk}(t)} = \sum_{i=1}^{n} \frac{\partial E_i(t)}{\partial y_i} \times \frac{\partial y_i}{\partial w_{jk}(t)}$$

where

$$\frac{\partial E_i}{\partial y_i} = (-1)(y_i^d(t) - y_i(t))$$

$$\frac{\partial y_i}{\partial w_{jk}} = \frac{\partial y_i}{\partial v_j} \times \frac{\partial v_j}{\partial w_{jk}}$$
So,

\[
\frac{\partial E}{\partial w_{jk}} = - \sum_{i=1}^{n} (y_i^d - y_i) y_i (1 - y_i) w_{ij} v_j (1 - v_j) x_k
\]

\[
= -v_j (1 - v_j) x_k \sum_{i=1}^{n} n \delta_i w_{ij}
\]

**Update Law**

\[
w_{jk}(t + 1) = w_{jk}(t) + \eta \delta_j x_k
\]

where \( \delta_j = v_j (1 - v_j) \sum_{i=1}^{n} \delta_i w_{ij} \)
Generalized Delta Rule

This is the generalized gradient descent method to minimize the total output squared error of a multilayer network.

Consider a multilayer network with $L$ layers each having $n$ nodes as shown in the following figure.
Let $W_{i_l,i_{l-1}}$ denotes the synaptic weight connecting the $i^{th}$ neuron of layer $l$ to that of layer $l - 1$. Considering sigmoid function as the activation function for each layer, the weight update law can be written as:

$$W_{i_l,i_{l-1}}(t + 1) = W_{i_l,i_{l-1}}(t) + \eta \delta_i v_{i_{l-1}}$$

where $v_{i_{l-1}}$ is the output of $i^{th}$ neuron of layer $l - 1$.

$$\delta_i^L = (y_{i_L}^d - y_{i_L})y_{i_L}(1 - y_{i_L}) \quad \text{for output layer L}$$

$$\delta_i = v_i(1 - v_i) \sum_{i=1}^{n} \delta_{i_{l+1}} W_{i_{l+1},i_l} \quad \text{for other hidden layers}$$
All the weights can be updated in a similar manner.

- The recursive formula is the key to back-propagation learning.
- It allows the error signal of a lower layer $\delta_{i_{l-1}}$ to be computed as a linear combination of the error signal of the upper layer $\delta_{i_l}$.
- In this manner the error signals are back-propagated through all the layers from the top down.
Consider a four layer network as shown in following figure.

\[ W_{i_1,i_0} \quad W_{i_2,i_1} \quad W_{i_3,i_2} \quad W_{i_4,i_3} \]

**Weight Update Law for Output Layer (4th layer)**

\[ W_{i_4,i_3}(t + 1) = W_{i_4,i_3}(t) + \eta \delta_{i_4} v_{i_3} \]

where, \( \delta_{i_4} = (y_{i_4}^d - y_{i_4})y_{i_4}(1 - y_{i_4}) \)

where \( y_{i_4} \) is the \( i^{th} \) unit of the output layer and \( y_{i_4}^d \) is the corresponding desired output.
Weight Update Law for 3rd layer

\[ W_{i_3,i_2}(t + 1) = W_{i_3,i_2}(t) + \eta \delta_{i_3} v_{i_2} \]

where,
\[ \delta_{i_3} = v_{i_3} (1 - v_{i_3}) \sum_{i=1}^{n} \delta_{i_4} W_{i_4,i_3} \]

Weight Update Law for 2nd layer

\[ W_{i_2,i_1}(t + 1) = W_{i_2,i_1}(t) + \eta \delta_{i_2} v_{i_1} \]

where,
\[ \delta_{i_2} = v_{i_2} (1 - v_{i_2}) \sum_{i=1}^{n} \delta_{i_3} W_{i_3,i_2} \]

It has been considered that the number of neurons in each layer is \( n \).
Weight Update Law for 1st layer

\[ W_{i_1,i_0}(t + 1) = W_{i_1,i_0}(t) + \eta \delta_{i_1} x_{i_0} \]

where, \( \delta_{i_1} = v_{i_1}(1 - v_{i_1}) \sum_{i=1}^{n} \delta_{i_2} W_{i_2,i_1} \)

where \( x_{i_0} \) represents a typical signal in the input layer.
Example: System identification

A practical system (Surge Tank system) has been taken for simulation. Hydraulic transients and pressure changes can be controlled by a surge tank.

The schematic of a surge tank is shown below:

When the pressure increases due to a sudden change in flow from the reservoir, level of the surge tank increases, thus controlling the flow as well as the pressure to the connecting system.
The system model is given as

$$h(t + 1) = h(t) + T \left( \frac{-\sqrt{2gh(t)}}{\sqrt{3h(t)} + 1} + \frac{u(t)}{\sqrt{3h(t)} + 1} \right)$$

\[ t \]: discrete time step
\[ T \]: sampling time
\[ u(t) \]: input fbw, can be positive or negative
\[ h(t) \]: liquid level of the tank (output)
\[ g \]: the gravitational acceleration
Sampling time is taken as 0.01 sec, 150 data have been generated using the system equation. The nature of input $u(t)$ and $y(t)$ is shown in the following figure.
The system is identified from a set of 100 input-output data using a multilayer feed-forward network.

The parameters of the network is given below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of hidden layer</td>
<td>2</td>
</tr>
<tr>
<td>Units in 1st hidden layer</td>
<td>15</td>
</tr>
<tr>
<td>Units in 2nd hidden layer</td>
<td>15</td>
</tr>
<tr>
<td>Number of outputs</td>
<td>1 [target: $h(t + 1)$]</td>
</tr>
<tr>
<td>Number of inputs</td>
<td>2 [(u(t), h(t)]</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Number of I/O data</td>
<td>150</td>
</tr>
<tr>
<td>Activation function of hidden layers</td>
<td>Sigmoid</td>
</tr>
<tr>
<td>Activation function of output layer</td>
<td>linear</td>
</tr>
</tbody>
</table>
Training is done using generalized delta rule. After 20000 epochs, the rms error is found to be < 0.004. Convergence of mean square error is shown in the following figure.
Model Validation

After identification, the model is validated through another set of 100 input-output data which is different from the dataset used for training. The result is shown in the following figure where left figure represents the desired and network output and right figure shows the corresponding input.
Problems in Back-Propagation

Following figure shows the error surface vs weight vector in simple gradient descent back-propagation

Error surface will have only one global minimum for linear systems and can have many local minima for nonlinear systems. Effect of noise in instantaneous update may avoid some local minima. However reaching global minimum is not guaranteed.
Several variations have been suggested
- to improve the convergence speed
- to avoid local minima’s
- for generalization capability
Adding a momentum

Adding a momentum factor means giving a momentum to the learning rate of BPA.

- This allows us to increase the learning rate and to avoid long flat surfaces.

If $\alpha$ is the momentum rate then

$$w(t + 1) = w(t) - \eta \frac{\partial E}{\partial w(t)} + \alpha [w(t) - w(t - 1)]$$

$$(1 - \alpha) \Delta w = -\eta \frac{\partial E}{\partial w(t)}$$

$$\Delta w = - \frac{\eta}{1 - \alpha} \frac{\partial E}{\partial w(t)}$$

The learning rate is thus increased by a factor $\frac{1}{1-\alpha}$ in the flat error surface.
Example: XOR gate

Truth table

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
</tr>
</tbody>
</table>

The weights are learned using gradient descent with momentum. The learning and momentum rates are taken as $\eta = 0.5$ and $\alpha = 0.8$ respectively.
Number of epochs to reach an RMSE of $< 0.0005$ is $\sim 2200$, where as for simple gradient descent it is $\sim 4200$. 
Adaptive learning rate

A heuristic approach to adaptive learning rule:

- A different learning rate is assigned to each adjustable parameter.
- Each learning rate parameter should be allowed to vary from iteration to iteration since the error surface behaves differently along different regions.
- When the derivative of cost function w.r.t. weight has same algebraic sign for several consecutive iterations, the learning rate parameter for that weight should be increased.
- When the algebraic sign of the derivative alternates consecutively for several iterations, the learning rate parameter should be reduced.
If $\Delta w_i^P(t - 1)$ and $\Delta w_i^P(t)$ are the weight changes at time $t - 1$ and $t$ respectively and $\eta_i^P(t - 1)$ is the corresponding learning rate at $t - 1$, then the new learning rate is calculated as follows:

$$
\eta_i^P(t) = \begin{cases} 
\mu \eta_i^P(t - 1) & \text{if } \Delta w_i^P(t)\Delta w_i^P(t - 1) \geq 0 \\
\quad d \eta_i^P(t - 1) & \text{if } \Delta w_i^P(t)\Delta w_i^P(t - 1) < 0
\end{cases}
$$

Constants $\mu$ and $d$ are an increase and a decrease factor respectively.
Example: XOR gate

The same example of learning a XOR gate is presented.

The learning parameters are given below:

- Basic learning rate $\eta$ : 0.5
- Increasing factor $\mu$ : 1.1
- Decreasing factor $d$ : 0.7

Training is done until an RMSE of $< 0.0005$ is achieved.
Number of epochs to reach an RMSE of $< 0.0005$ is $\sim 1300$ using adaptive learning rate, where as for simple gradient descent back-propagation it is $\sim 4200$. 

Performance is $2.49997e^{-008}$, Goal is $2.5e^{-008}$

Performance is $2.48055e^{-008}$, Goal is $2.5e^{-008}$