Supervised Learning in Neural Networks

(Part 3)

Supervised Learning in Neural Networks – using matlab

- The MATLAB® Neural Network Toolbox implements some of the most popular training algorithms, which encompass both original gradient-descent and faster training methods.

- **Batch Gradient Descent (traingd):**
  - Original but the slowest.
  - Weights and biases updated in the direction of the negative gradient.
  - Selected by setting `trainFcn` to `traingd`:
    ```matlab
    net = newff(minmax(p), [3 1], {'tansig', 'purelin'}, 'traingd');
    ```

- **Batch Gradient Descent with Momentum (traingdm):**
  - Faster convergence than `traingd`.
  - Momentum allows the network to respond not only the local gradient, but also to recent trends in the error surface.
  - Momentum allows the network to ignore small features in the error surface; without momentum a network may get stuck in a shallow local minimum.
  - Selected by setting `trainFcn` to `traingdm`:
    ```matlab
    net = newff(minmax(p), [3 1], {'tansig', 'purelin'}, 'traingdm');
    ```

- **Faster Training.**

- The MATLAB® Neural Network Toolbox also implements some of the faster training methods, in which the training can converge from ten to one hundred times faster than `traingd` and `traingdm`.
  - These faster algorithms fall into two categories:
    1. **Heuristic techniques:** developed from the analysis of the performance of the standard gradient descent algorithm, e.g. `traingda`, `traingdx` and `trainrp`.
    2. **Numerical optimization techniques:** make use of the standard optimization techniques, e.g. conjugate gradient (`traincfg`, `traincgb`, `traincgf`, `trainscg`), quasi-Newton (`trainbfg`, `trainoss`), and Levenberg-Marquardt (`trainlm`).
Comparison of Training Algorithms

<table>
<thead>
<tr>
<th>Training Algorithms</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>traingd</td>
<td>Gradient Descent (GD)</td>
</tr>
<tr>
<td>traingdm</td>
<td>GD with momentum</td>
</tr>
<tr>
<td>traingda</td>
<td>GD with adaptive α</td>
</tr>
<tr>
<td>traingdx</td>
<td>GD with adaptive α and with momentum</td>
</tr>
<tr>
<td>trainrp</td>
<td>Resilient Backpropagation</td>
</tr>
<tr>
<td>traincfg</td>
<td>Fletcher-Reeves Update</td>
</tr>
<tr>
<td>traincgp</td>
<td>Polak-Ribière Update</td>
</tr>
<tr>
<td>traincgb</td>
<td>Powell-Beale Restarts</td>
</tr>
<tr>
<td>trainscg</td>
<td>Scaled Conjugate Gradient</td>
</tr>
<tr>
<td>trainbfg</td>
<td>BFGS algorithm</td>
</tr>
<tr>
<td>trainoss</td>
<td>One Step Secant algorithm</td>
</tr>
<tr>
<td>trainlm</td>
<td>Levenberg-Marquardt</td>
</tr>
<tr>
<td>trainbr</td>
<td>Bayesian regularization</td>
</tr>
</tbody>
</table>

Modeling Logical XOR Function

- The **XOR solving problem** using a simple backpropagation network

```matlab
%Solution:
% Define the training inputs and targets
p = [0 0 1 1; 0 1 0 1];
t = [0 0 0 1];
% Create the backpropagation network
net = newff(minmax(p), [4 1], {'logsig', 'logsig'}, 'traingdx');
% Train the backpropagation network
net.trainParam.epochs = 500; % training stops if epochs reached
net.trainParam.show = 1; % plot the performance function at every epoch
net = train(net, p, t);
% Testing the performance of the trained backpropagation network
a = sim(net, p)
>> a = 0.0002  0.0011  0.0001  0.9985
>> t = 0  0  0  1
```