Fundamental Theories and Applications of Neural Networks

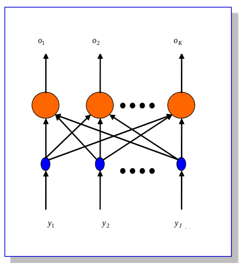
Lecture 3: Multi-layer perceptron

Contents of this lecture

- Review of single layer neural networks.
- Formulation of the delta learning rule of single layer neural networks.
- BP: an extended delta-learning rule for multilayer neural networks.

Review of single layer neural network

- There are J inputs and K outputs.
- The last input is fixed to –1 so that the corresponding weight is the threshold.
- For a given input vector ***y***
  - The *effective input* of the *k-th* neuron is net_k
  - The *actual output* of the *k-th* neuron is o_k
  - The *desired output* of the *k-th* neuron is d_k
  - The *error* to be minimized is E



Reformulation of the delta learning rule

- According to the gradient descent algorithm, the weight from the *j-th* input to the *k-th* neuron should be updated by

$$\boxed{w_{kj}^{new} = w_{kj}^{old} - \eta \frac{\partial E}{\partial w_{kj}} \Big|_{w_{kj}=w_{kj}^{old}}}$$

Gradient of the *error function*

$$E = \frac{1}{2}\sum_{k=1}^{K}(d_k - o_k)^2$$

Note that E is implicitel y a function of net_k, using the chain rule, we can get the partial direvative of E to w_{kj} as follows :

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial(net_k)}\frac{\partial(net_k)}{\partial w_{kj}}$$

where

$$\frac{\partial(net_k)}{\partial w_{kj}} = \frac{\partial(\sum_{j=1}^{J} w_{kj} y_j)}{\partial w_{kj}} = y_j$$

Definition of the *error signal*

If we define the error signal produced by the k - th neuron as follows :

$$\delta_{o_k} = -\frac{\partial E}{\partial(net_k)}$$

we have

$$w_{kj}^{new} = w_{kj}^{old} + \eta\delta_{o_k} y_j$$

where

$$\delta_{o_k} = -\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial(net_k)} = (d_k - o_k) f'(net_k)$$

Equation for updating the weights

Thus, the weight can be updated by

$$w_{kj}^{new} = w_{kj}^{old} + \eta(d_k - o_k) f'(net_k) y_j$$

If we use the unipolar sigmoid function with $\lambda = 1$,

$$f'(net_k) = o_k(1 - o_k)$$

If we use the bipolar sigmoid function with $\lambda = 1$,

$$f'(net_k) = (1 - o_k^2)/2$$

Remarks

- For off-line learning
  - The error should be defined as the **total error** of the network for all training examples.
  - The training examples are used repeatedly until the error becomes small enough.
- The weights of all neurons are updated all together in a synchronous mode.

Program of delta learning rule for single layer neural network

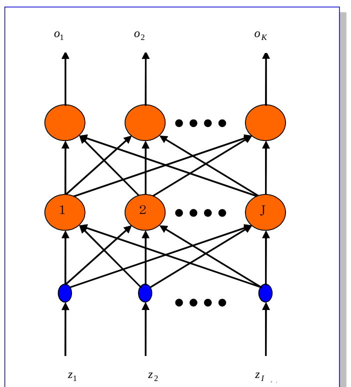
```
Initialization();
while(Error>desired_error){
   for(Error=0,p=0; p<n_sample; p++){
      FindOutput(p);
      for(k=0;k<K;k++){
       Error+=0.5*pow(d[k][p]-o[k],2.0);
      }
      for(k=0;k<K;k++){
         delta=(d[k][p]-o[k])*(1-o[k]*o[k])/2;
         for(j=0;j<J;j++){
            w[k][j]+=eta*delta*y[p][j];
         }
      }
   }
}
```


Results for AND/OR gates

```
Error in the 321-th learning cycle=0.010297
Error in the 322-th learning cycle=0.010263
Error in the 323-th learning cycle=0.010230
Error in the 324-th learning cycle=0.010197
Error in the 325-th learning cycle=0.010165
Error in the 326-th learning cycle=0.010132
Error in the 327-th learning cycle=0.010100
Error in the 328-th learning cycle=0.010068
Error in the 329-th learning cycle=0.010036
Error in the 330-th learning cycle=0.010004
Error in the 331-th learning cycle=0.009973


W[0]:3.520518 3.521593 -3.519444
W[1]:3.520259 3.519185 3.521334
```


Multilayer feedforward neural network



Multilayer feedforward neural network

- The network in the last slide is a three layer perceptron, also called three layer feed forward neural network.
- There are
  - I inputs,
  - J hidden neurons, and
  - K output neurons.
- The last input of each hidden neuron or each output neuron is –1.
- The input can be output of another layer of neurons.
- The output can be input of another layer.

Definition of notations

- z_i : The *i-th* input
- y_j : The output of the *j-th* hidden neuron
- o_k : The output of the *k-th* output neuron
- v_{ji} : The weight from the *i-th* input to the *j-th* hidden neuron
- w_{kj} : The weight from the *j-th* hidden neuron to the *k-th* output neuron

Rule for updating the output neurons

- Weight update for the output neurons can be performed exactly in the same way as for single layer perceptron.
  - For any input, find the output of the hidden neurons, and then the output of the output neurons.
  - The weights of each output neuron can be updated by using the delta learning rule.

Rule for updating the hidden neurons

First, we have

$$v_{ji}^{new} = v_{ji}^{old} - \eta \frac{\partial E}{\partial v_{ji}}$$

Using the chain rule, we can get

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial (net_j)} \frac{\partial (net_j)}{\partial v_{ji}}$$

Error signal produced by the j-th hidden neuron

$$\delta_{y_j} = -\frac{\partial E}{\partial (net_j)}$$

Update the weights of hidden neurons

Note also that $\frac{\partial (net_j)}{\partial v_{ji}} = z_i$, we have

$$v_{ji}^{new} = v_{ji}^{old} + \eta \delta_{y_j} z_i$$

Using the chain rule, we can find the error signal δ_{y_j} as follows:

$$\delta_{y_j} = -\frac{\partial (E)}{\partial y_i} \frac{\partial y_j}{\partial net_i} = f'(net_j) \sum_{k=1}^{K} \delta_{o_k} w_{kj}$$

Update the weights of hidden neurons

Thus, the weights of the hidden neurons can be updated by

$$v_{ji}^{new} = v_{ji}^{old} + \eta \delta_{y_j} z_i$$

If we use the unipolar sigmoid function with $\lambda = 1$,

$$f'(net_j) = y_j(1 - y_j)$$

If we use the bipolar sigmoid function with $\lambda = 1$,

$$f'(net_j) = (1 - y_j^2)/2$$

Comments

- The learning algorithm is usually called the **back propagation** (BP) algorithm because the error signal of the hidden neurons are back propagated from the output layer to the hidden layer(s).
- In some context, the algorithm is also called the extended delta-learning rule.

Summary of the BP algorithm

- Step 1: Initialize the weights.
- Step 2: Reset the total error.
- Step 3: Get a training example z from the training set, calculate the outputs of the hidden neurons and those of the output neurons, and update the total error.
- Step 4: Calculate the error signals as follows:

$$\delta_{o_k} = (d_k - o_k)(1 - o_k^2)/2$$

$$\delta_{y_j} = \left(\sum_{k=1}^{K} \delta_{o_k} w_{kj} \right)(1 - y_j^2)/2$$

Summary of the BP algorithm (cont.)

- Step 5: Update the weights as follows:

$$\boldsymbol{w_{kj}^{new} = w_{kj}^{old} + \eta \delta_{ok} y_j} \quad \text{for } \boldsymbol{k} = 1,2,\cdots,\boldsymbol{K};\ \mathrm{j} = 1,2,\cdots,\boldsymbol{J}$$

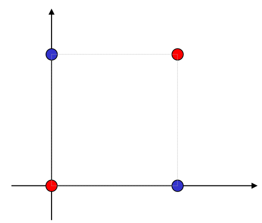
$$\boldsymbol{v_{ji}^{new} = v_{ji}^{old} + \eta \delta_{y_j} z_i} \quad \text{for } \boldsymbol{j} = 1,2,\cdots,\boldsymbol{J};\ \mathrm{i} = 1,2,\cdots,\boldsymbol{I}$$

- Step 6: See if all training examples have been used. If NOT, return to Step 3.
- Step 7: See if the total error is smaller than the desired value. If NOT, return to Step 2; otherwise, terminate.

A simple example:
Solving the XOR problem

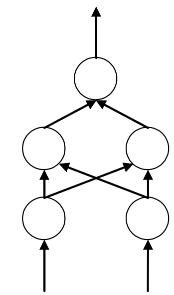
y_1	y_2	o
0	0	0
0	1	1
1	0	1
1	1	0

- The function to be approximated is a 2-variable binary function.
- This problem, although simple, cannot be solved by ANY single layer neural network.



The network structure

- XOR can be solved using a three layer neural network.
- It contains three inputs, with the last one being fixed to –1, two hidden neurons, and one output neuron.
- The problem is to find the correct weights for all neurons, so that for any given input, the correct output can be provided.



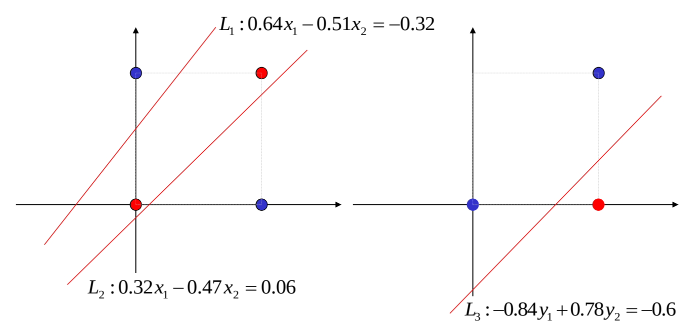
Results of BP

```
................
Error[1073]=0.001008
Error[1074]=0.001003
Error[1075]=0.000998

The connection weights in the output layer:
-0.839925 0.782646 -0.602153

The connection weights in the hidden layer:
0.638360 -0.509153 -0.316935
0.319389 -0.472554 0.068378
```


Physical meaning of the result



To improve the BP algorithm

- The learning constant can be variable
  - If the error is reduced greatly by the current update, the learning rate can be increased.
  - If the error is not reduced, the learning rate can be decreased.
- Momentum method
  - The increment of the weight can be modified by the updating history.

How many hidden neurons to use?

- According to "Multilayer Perceptrons: Approximation order and Necessary Number of Hidden Units" (written by Stephan Trenn, IEEE TNN, Vol. 19, No. 5, 2008, pp. 836-844), for an MLP with one hidden layer, n_0 inputs, and smooth activation function, it achieves approximation order N for all functions only if the number of hidden units is larger than

$$\frac{\binom{N+n_0}{n_0}}{n_0+2}$$

How many hidden Layers to use?

- For high approximation order (>11), two hidden layers should be used instead of one hidden layer. For linear and quadratic approximation, only one hidden layer is needed.
- Here, a function *f* approximates another function *g* with order N if and only if their Taylor polynomials are the same up to the order N. The function to be approximated by the MLP should be sufficiently smooth.
- The numbers given here are relatively conservative because the MLP must approximate ANY function well (to solve a practical problem, we may consider one function or a special set of functions only).

Team Project II

- Make a computer program for the BP algorithm.
- Test your program using the 4-bit parity check problem.
- The number of inputs is 5 (4 plus one dummy input) and the number of output is 1 ([0,1] or [-1,1]).
- The desired output is 1 if the number of ones in the inputs is even; otherwise, the output is 0 or -1.
- Check the performance of the network by changing the number of hidden neurons from 4 to 10, with step-size 2.
- Provide a summary of your results in your report (txt-file).