Major Classes of Neural Networks

Outline

- Multi-Layer Perceptrons (MLPs)
- Radial Basis Function Network
- Kohonen's Self-Organizing Network
- Hopfield Network

Multi-Layer Perceptrons (MLPs)

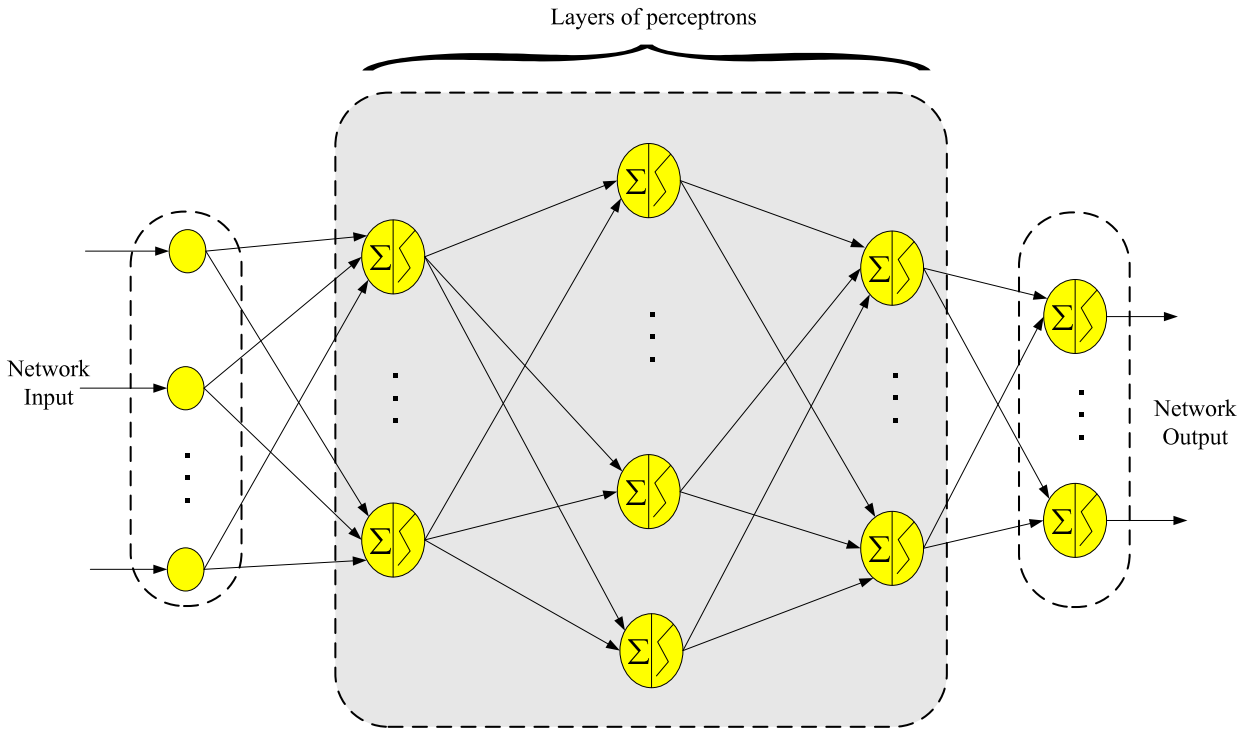
Background

- The perceptron lacks the important capability of recognizing patterns belonging to non-separable linear spaces.
- The madaline is restricted in dealing with complex functional mappings and multi-class pattern recognition problems.
- The multilayer architecture first proposed in the late sixties.

Background (cont.)

- MLP re-emerged as a solid connectionist model to solve a wide range of complex problems in the mid-eighties.
- This occurred following the reformulation of a powerful learning algorithm commonly called the Back Propagation Learning (BPL).
- It was later implemented to the multilayer perceptron topology with a great deal of success.

Schematic Representation of MLP Network



This is the struture of the neural network. It is the most general structure. An input layer is where signals enter the system. These are usually the features of an object. The yellow circles are just units. We call them perceptrons, this is a bit misleading. These perceptrons are different from the ones we saw earlier. The earlier ones create discrete values but they instead (most of the time) they have a smooth, nonlinear activation function. Their activation functions are very differentiable. They have the same structure though. The output layer generates the actual output of the system.

Backpropagation Learning Algorithm (BPL)

- The backpropagation learning algorithm is based on the **gradient descent technique** involving the **minimization of the network cumulative error**.

$$E(k) = \sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$$

  - i represents i-th neuron of the output layer composed of a total number of q neurons.

- It is designed to **update the weights in the direction of the gradient descent of the cumulative error**.

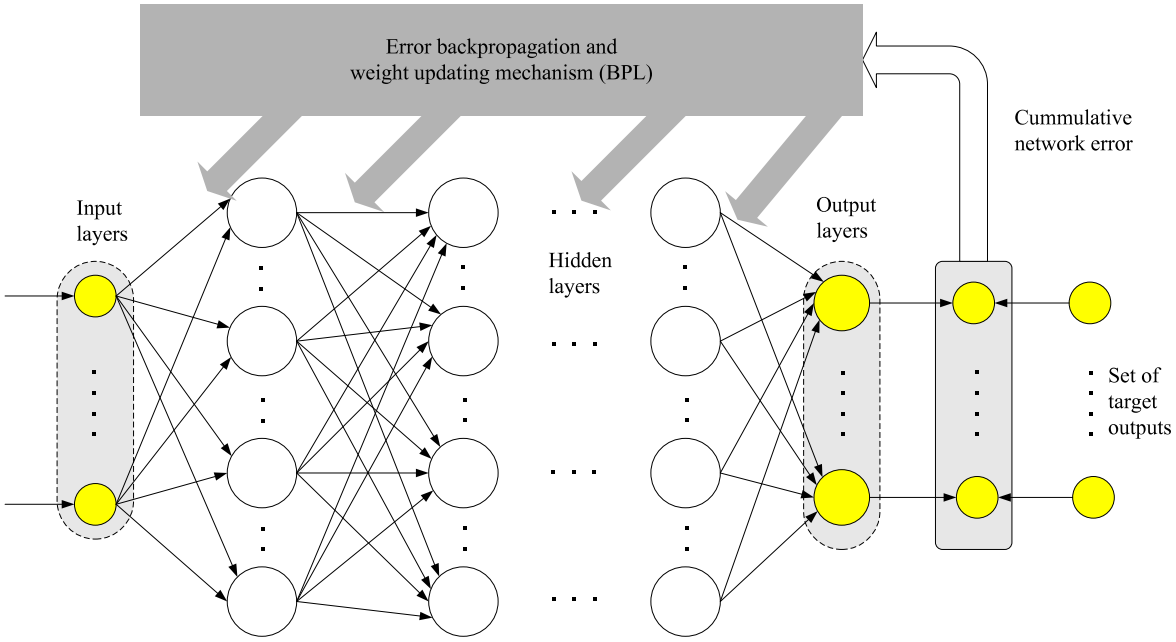
Backpropagation Learning Algorithm (cont.)

A Two-Stage Algorithm

1. First, patterns are presented to the network.
2. A feedback signal is then propagated backward with the main task of updating the weights of the layers connections according to the back-propagation learning algorithm.

BPL: Schematic Representation

- Schematic Representation of the MLP network illustrating the notion of error back-propagation



The multilayer perceptron has many layers to propagate signals. Most of the time, these kinds of models are supervised (we have to teach them).

This is still an optimization problem. We have some weights that we want to optimize our weights so that they have the least error. These kinds of systems have a ton of dimensions to optimize across. There will be very complex structures which make it very hard to actually solve. This is why we tend towards using other systems to get an optima even if its not the global optima.

When a solution is found (weights are known) we use this to find the weights of the next layer. This backpropagates the error signal through the layers backwards untill it reaches the first layer. You keep feeling in training signals until you run out and update all the weights. At this point we have finished one epoch.

Say you have $\overrightarrow{x}(x_1, ..., x_m)$ then you have the output $\overrightarrow{y}(y_1, ..., y_l)$ so we then get the mapping $\overrightarrow{y} = f(\overrightarrow{x})$. So your first training signal has to be a bunch of mappings of x to y. Roughly k signals.

Backpropagation Learning Algorithm (cont.)

Objective Function

- Using the **sigmoid function** as the activation function for all the neurons of the network, we define E_c as

$$E_c = \sum_{k=1}^{n} E(k) = \frac{1}{2} \sum_{k=1}^{n} \sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$$

Here i is the index of the element we are looking at and k is the index of the training set we are currently looking at. This makes an error function. We compute the sum of the square error for each node and propagate it backwards. We keep going backwards until we reach the communication layer (one right after the input layer).

Backpropagation Learning Algorithm (cont.)

- The formulation of the **optimization problem** can now be stated as **finding the set of the network weights** that minimizes E_c or $E(k)$.

Objective Function: Off-Line Training

$$min_w E_c = min_w \frac{1}{2} \sum_{k=1}^{n} \sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$$

Objective Function: On-Line Training

$$min_w E(k) = min_w \frac{1}{2} \sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$$

BPL: On-Line Training

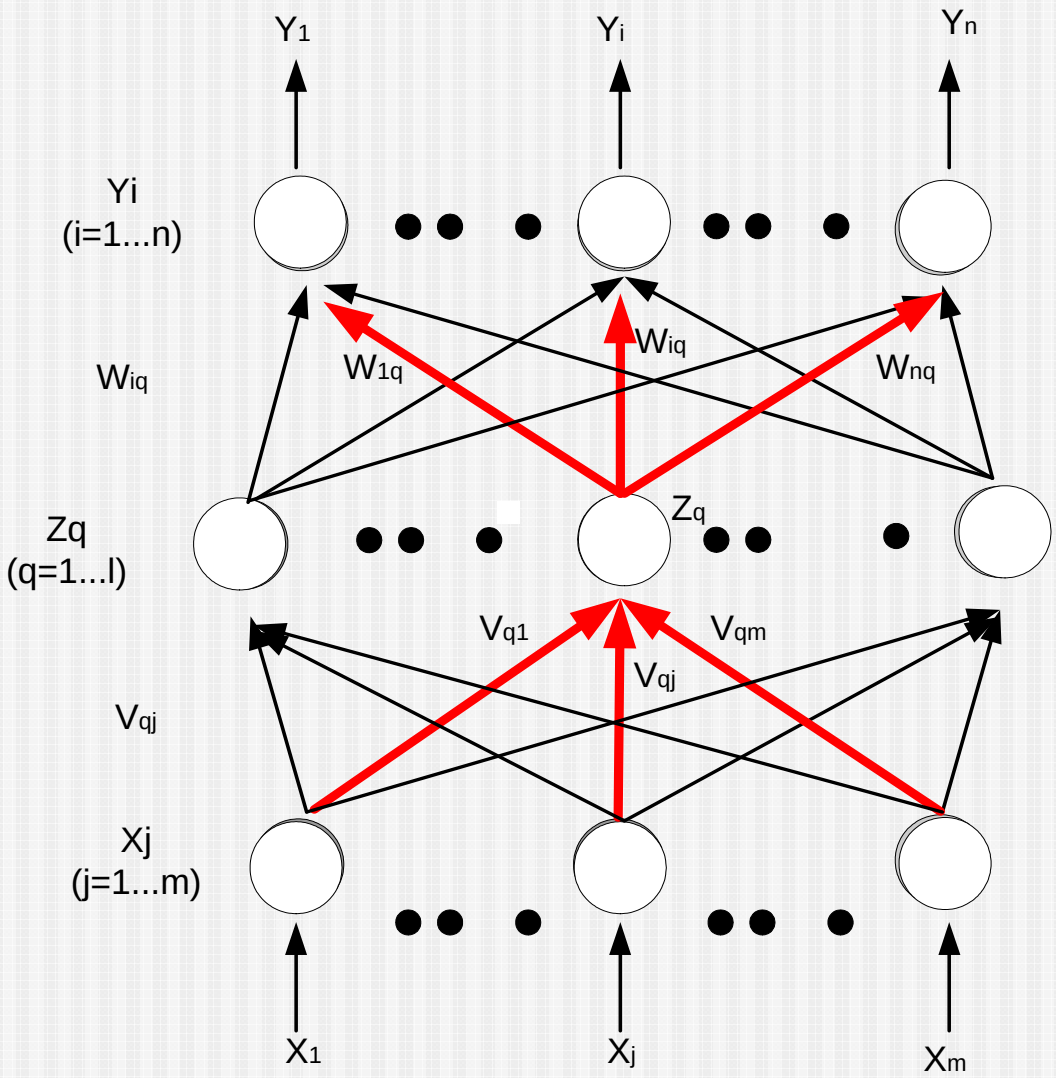
- Objective Function: $min_w E(k) = min_w \frac{1}{2} \sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$

Updating Rule for Connection Weights

$$\Delta w^{(l)} = -\eta \frac{\partial E(k)}{\partial w^l},$$

- l is layer (l-th) and η denotes the learning rate parameter,
- $\Delta w_{ij}^{(l)}$: the weight update for the connection linking the node j of layer $(l-1)$ to node i located at layer l.

Here is the update function for the weights. It looks pretty standard. We work layer by layer.


Y1
Yi
Yn
Yi
(i=1...n)
Wiq
W1q
Wiq
Wnq
Zq
Zq
(q=1...l)
Vq1
Vqm
Vqj
Vqj
Xj
(j=1...m)
X1
Xj
Xm

Here we have input layer x with m nodes in it, middle layer z with l nodes in it, and output layer y with n nodes in it. There are three weights going between x and z labeled with v, there are other ones but we are going to ignore them for simplicity in this example. There are other weights from z to y labeled as w, again there are others that we are just ignoring to make the example easier to go through. The subscript on the notation is the destination node,origin node notation. So how do we find the weights?

$$
\begin{aligned}
a &= \texttt{activation function} \\
net_q &= \sum_{j=1}^{n} v_{qj} x_j \\
z_q &= a(net_q) \\
&= a(\sum_{j=1}^{n} v_{qj} x_j) \\
net_i &= \sum_{q=1}^{l} w_{iq} z_q \\
y_i &= a(net_i) \\
y_i &= a(\sum_{q=1}^{l} w_{iq} z_q) \\
y_i &= a(\sum_{q=1}^{l} w_{iq} a(\sum_{j=1}^{n} v_{qj} x_j))
\end{aligned}
$$

In this case the two a functions do not have to be the same but its much easier if they are.

What about the target data (denoted d_i)?

$$E = \frac{1}{2} \sum_{i=1}^{n} (d_i - y_i)^2$$