MULTI-LAYERED PERCEPTRONS (MLPs)

Generally much more versatile than single neurons. Many problems (such as XOR) require non-linear separation. Training is more time consuming. A multi-layered perceptron has three distinctive characteristics;

- A non-linear activation function
- One or more hidden layers of hidden neurons
- High degree of connectivity

Components of MLPs

1. **Network Architecture** – RNN, FeedForward, Backpropagation, Competitive (inhibition), Static, Dynamic
2. **Activation function**
   a. Non-linear activation function (Sigmoid, Hyperbolic sigmoid)
3. **Learning Mechanism**
   a. Supervised (train set, training algorithm, test set)
   b. Unsupervised (training algorithm, test set)

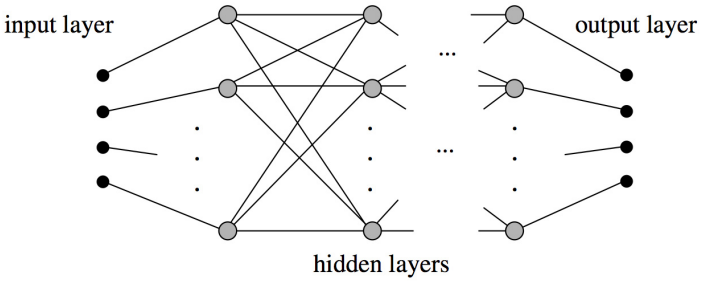

Fig 35: A generic multi-layered network

In layered architectures normally all units from one layer are connected to all other units in the following layer. If there are **m** units in the first layer and **n** units in the second one, the total number of weights is **m.n**. The total number of connections can become rather large which might require pruning the network.

The XOR problem revisited The properties of one- and two-layered networks can be discussed using the case of the XOR function as an example. We already saw that a single perceptron cannot compute this function, but a two-layered network can. The network in following figure is capable of doing this using the parameters shown in the figure.

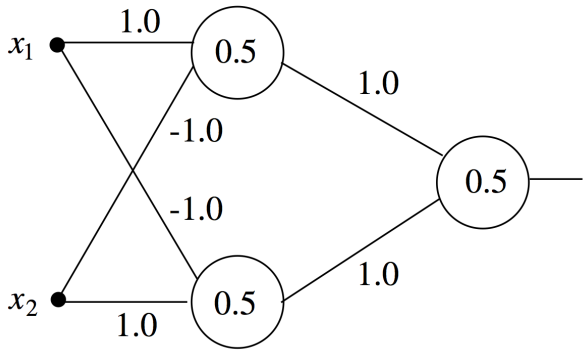

Fig 36: MLP for XOR

The network consists of three layers (adopting the usual definition of the layer of input sites as input layer) and three computing units. One of the units in the hidden layer computes the

function **x1 ^ ¬x2**, and the other the function **¬x1 ^ x2**. The third unit computes the OR function, so that the result of the complete network computation is

$$(x1 \wedge \neg x2) \vee (\neg x1 \wedge x2)$$

One of the interesting feature of MLPs is that the same structure can be used to represent multiple solutions.

Example.
Let's consider two input sets;

X1 = [1 0 1 0]
X2 = [1 1 0 0]

If, the application of function **f(X1, X2)** results into **0**, then we represent it by $\mathbf{f_0}$.

For example; for AND function, **f(X1, X2)** results into [1 0 0 0], which we represent by $\mathbf{f_8}$.

The sixteen possible functions of two variables are thus:

$f_0(x1, x2) = f_{0000}(x1, x2) = 0$
$f_1(x1, x2) = f_{0001}(x1, x2) = \neg(x1 \vee x2)$
$f_2(x1, x2) = f_{0010}(x1, x2) = x1 \wedge \neg x2$
$f_3(x1, x2) = f_{0011}(x1, x2) = \neg x2$
$f4(x1, x2) = f_{0100}(x1, x2) = \neg x1 \wedge x2$
$f5(x1, x2) = f_{0101}(x1, x2) = \neg x1$
$f6(x1, x2) = f_{0110}(x1, x2) = x1 \oplus x2$
$f7(x1, x2) = f_{0111}(x1, x2) = \neg(x1 \wedge x2)$
$f8(x1, x2) = f_{1000}(x1, x2) = x1 \wedge x2$
$f9(x1, x2) = f_{1001}(x1, x2) = x1 \equiv x2$
$f10(x1, x2) = f_{1010}(x1, x2) = x1$
$f11(x1, x2) = f_{1011}(x1, x2) = x1 \vee \neg x2$
$f12(x1, x2) = f_{1100}(x1, x2) = x2$
$f13(x1, x2) = f_{1101}(x1, x2) = \neg x1 \vee x2$
$f14(x1, x2) = f_{1110}(x1, x2) = x1 \vee x2$
$f15(x1, x2) = f_{1111}(x1, x2) = 1$

The XOR solution presented in figure 36 corresponds to the function composition;

$$X1 \oplus X2 = (X1 \wedge \neg X2) \vee (\neg X1 \wedge X2) = f_{14}(f_2(X1, X2), f_4(X1, X2))$$

Increasing the number of units in the hidden layer increases the number of possible combinations available. We say that the capacity of the network increases.

Refer to R. Rojas p. 128 – 137 for segmentation of input-space and error minimization for discrete input space and multiple-linear separation.

Note: While training MLP for XOR, sigmoid (logsig) activation function at hidden layer and threshold (hardlim) activation function at output layer should be used.

References

- Rojas, R., *Neural Networks – A Systematic Introduction, Chap 6*
- Haykin, S., *Neural Networks – A Comprehensive Foundation, Chap 4*
- Keller, B., *Neural Networks, CS-152 - BP*

PRACTICAL #5

Single layer Perceptrons for multi-category classification
(June 20, 2009)

Objectives
To implement a set of perceptrons in a single layer for multi-category classification.

Description
Implement a set of perceptrons that accept inputs and classify inputs into multi-category 'yes' / 'no' decisions.

Example of categories can be; sweet and sour taste, sweet and salty taste etc.

Platform
Any platform with any language of choice.

Model
Use Winner-Take-All (WTA) model as described in p. 17 of Unit 3.

Note: This task is an assignment. Implement the WTA model and submit by June 27, 2009.

PRACTICAL #6

MLP implementation using Matlab Neural Network Toolbox
(June 20, 2009)

Objectives

To implement MLP with feedforward-backpropagation training algorithm to solve non-linear classification problem (e.g. XOR).

Description

Use 'newff' function to create neural network with feed-forward training algorithm.

Let's create a network with three layers and train for XOR classification.

Note: The input layers are not considered to be existing layers in Matlab.

```
inputs = [1 0 1 0; 1 1 0 0];
target = [0 1 1 0];

net = newff(inputs, target, [2 1], {'logsig', 'logsig'});
net = train(net, inputs, target);
output = sim(net, inputs)
plot(inputs, target, inputs, output, 'o')
```

Let's see another example;

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [0 1 2 3 4 3 2 1 2 3 4];
net = newff(P,T,5);
net = train(net, P, T);
Y = sim(net,P);
plot(P,T,P,Y,'o')
```

The plotting shows that the input sets 'P' are not being classified quite well by the network. Let's change the number of epoch and see the output again.

```
net.trainParam.epochs = 50;
net = train(net,P,T);
Y = sim(net,P);
plot(P,T,P,Y,'o')
```

To do: Implement the MLPs described here using 'nntool'.