Neural Networks
- Homework 3 -

Petr Lukin, Ivan Vishniakou, Evgeniya Ovchinnikova

Lecture date: 17 October 2016

1 Exercises

1.1 Exercise 2.1

The delta rule described in the following equation:
$\Delta w_{kj}(n) = \eta e_k(n) x_j(n)$
and Hebb's rule described in:
$\Delta w_{kj}(n) = \eta y_k(n) x_j(n)$
represent the two different methods of learning. List the features that distinguish these two rules from each other.

Solution:

- Learning rule: delta rule is a part of error-correction learning, whereas Hebb's rule is a part of memory-based learning.
- Adjustment: in delta rule it is based on the difference between desired and actual output (the error) and in Hebb's rule - on correlation, whether two neurons connected by synapse that is being considered are activated simultaneously or not.

1.2 Exercise 2.10

Formulate the expression for the output y_j of neuron j in the network of Fig. 1, where:

$x_i = i^{th}$ input signal,
w_{ji} = synaptic weight from input i to neuron j,
c_{kj} = weight of lateral connection from neuron k to neuron j,
$y_j = \phi(v_j)$.

What is the condition that would have to be satisfied for neuron j to be the winning neuron?

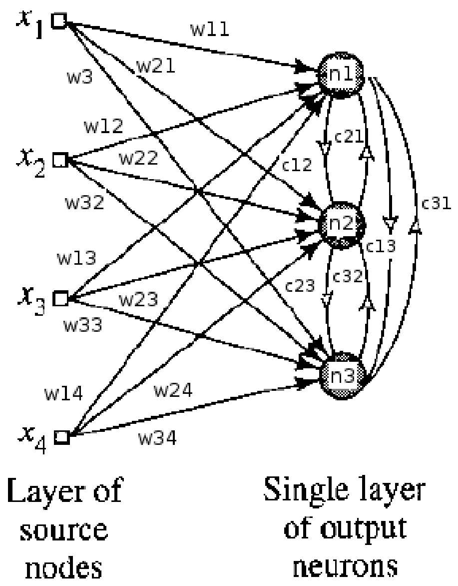
Solution:

$$y_j = \phi(\sum_{i=1}^{4} w_{ji} x_i + \sum_{k=1, k \neq j}^{3} c_{kj} y_k) = \phi(\sum_{i=1}^{4} w_{ji} x_i + \sum_{k=1, k \neq j}^{3} c_{kj} \phi(v_k))$$

, where k_1, k_2 are numbers from 1 to 3 those are not equal j.

Neuron j is a winning neuron if $v_j > v_k$ for all $k \neq j$.

Figure 1: Competitive neural network with feedforward connections.



1.3 Exercise 2.21

Fill in the details of the level of signal processing next to that described in figure:

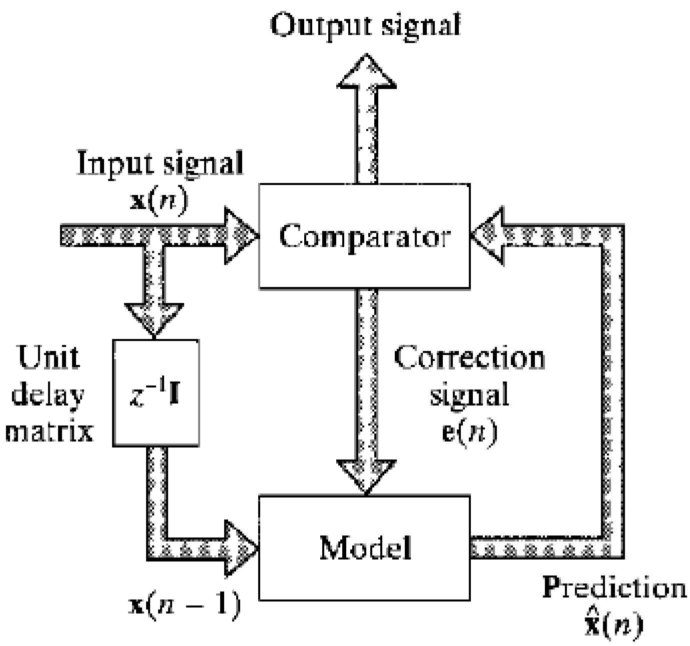


FIGURE P2.21

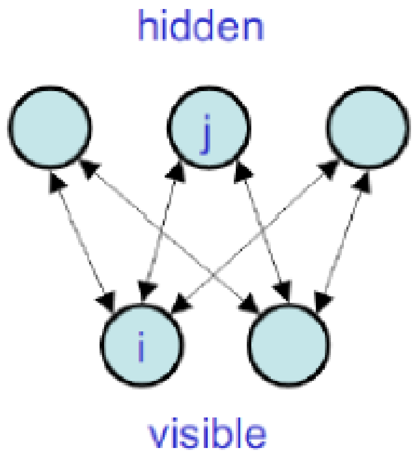
- In this diagram, predictive model block contains and adaptive filter that receives delayed signal and tries to predict current one.
- Adaptive filter in the model works as linear combiner and can be trained with LMS method.

- Comparator compares predicted value with plant output to produce error.
- This error can be used to update weight matrix in model.
- The model can be used to continuously predict new values and adapt.

1.4 Exercise 4

A simple network is given below (from lecture slides). You have to update the weights once using Boltzmann learning for this network. Please do calculations by hand or by using MATLAB or Python. Use random numbers to initialise the weights. For training use a training set as $[(0,1),(1,0)]$ or any training set of your choice.

Figure 2: NN from ex. 4.



Because this NN is recurrent and has 5 neurons, weight matrix has size 5x5 with zeros on a main diagonal. The training sample is $s = [1, -1, -1, -1, -1]$. Initial weight matrix was filled with $uniform[0, 1]$ numbers.
The weight update was done in matlab:

```
%% Exercise 5 from the homework pdf.
%Update weights once with Boltzmann learning rule
%Authors P.Lukin, I. Vishniakou, E. Ovchinnikova

%Initialization of a weight matrix. We have 5 neurons, so matrix is 5x5
w = zeros(5,5);
w(1:2,3:5) = rand(2,3);
w(3:5,1:2) = rand(3,2);
nu = 0.2;
%Neuron's states
s = -ones(1,5);
%Training sample s1 =1 s2=10
s(1) =1;
exps = s;

```

```
%% Clamped state
%Energy change
for i=3:5
    dE(i) = sum(w(1,:).*s);
end
%Flip

for i=3:5
    r = rand(1,1);
    p = 1/(1+exp(-dE(i)));
    if r>p
        s(i) = -s(i);
    end
end
sClamp =s;

%% Free run state state
s = exps;
%Energy change
for i=1:2
    dE(i) = sum(w(1,:).*s);
end
%Flip

for i=1:2
    r = rand(1,1);
    p = 1/(1+exp(-dE(i)));
    if r>p
        s(i) = -s(i);
    end
end
sFree = s;

%Weight update
for i=1:5
    for j=1:5
        dw(i,j) = corrcoef(sClamp(i),sFree(j));
        if isnan(dw(i,j))
            dw(i,j)=0;
        end
    end
end
dw(1:5+1:5*5) = 0;
'Weights update'
dw
w =w+nu*dw
```