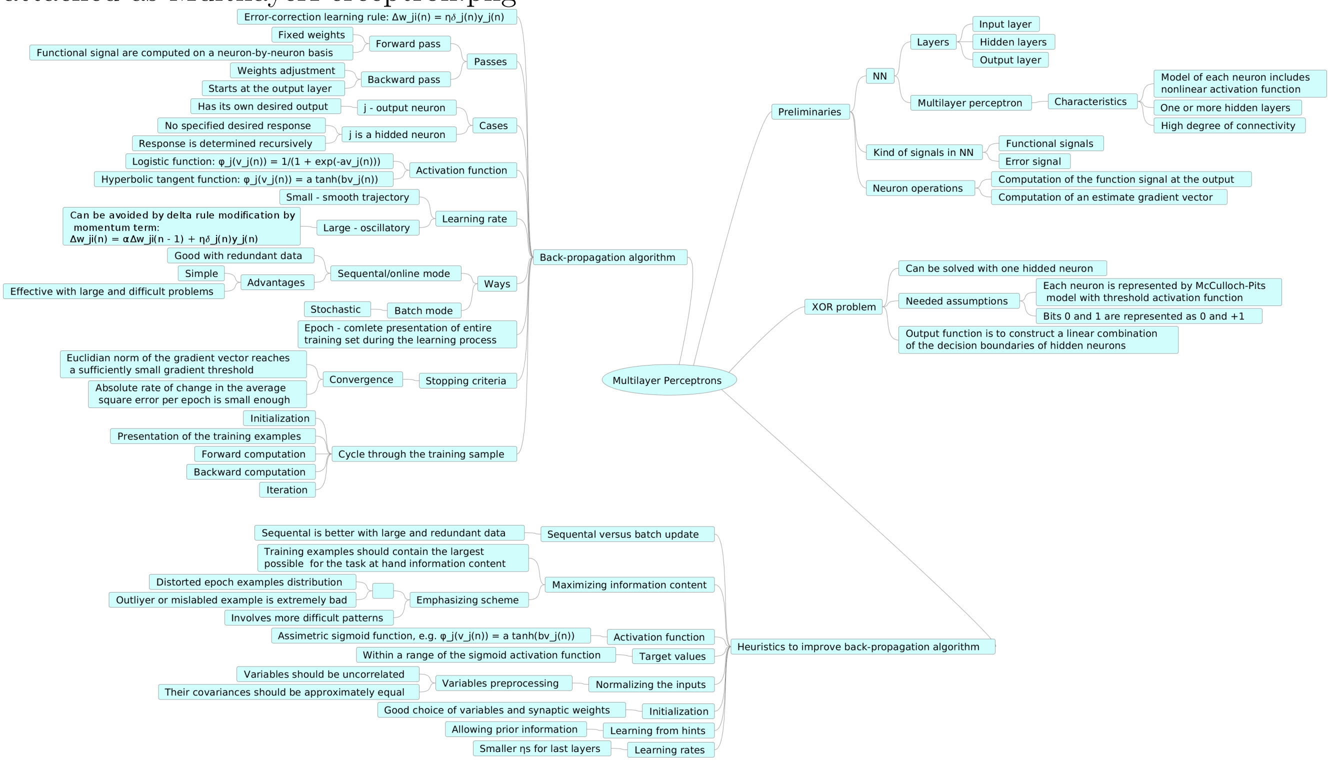
Neural Networks
- Homework 5 -

Petr Lukin, Evgeniya Ovchinnikova

Lecture date: 31 October 2016

1 Mind map

Figure 1: Mind map. Chapter 4 (first part) from Haykins book. A zoomed version is attached as MultilayerPerceptron.png



2 Exercises

2.1 Exercise 2

For this task you have to program the back-propogation (BP) for multi layered perceptron (MLP). Design your implementation for general NN with arbitrary many hidden layers. The test case is as follows: 2-2-1 multi layered perceptron (MLP) with sigmoid activation function on XOR data.

a. Experiments with initial weights

i. Train the network with zero initial weights i.e. $w_{ij} = 0$.

ii. Train with random initial weights

Compare and comment on the convergence.

b. Experiment with different learning rates e.g. 0.1, 0.3, 0.5, 0.9..

Compare the convergence and plot some resulting surfaces. You are not allowed to use any neural network toolbox for this solution.

NB: If you fail to implement the general case in order to get the full points it is sufficient to implement only the use case (2-2-1 MLP)

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA

%matplotlib inline

#constants:

sigma = 2
r_xd = 3
r_x = 4
eta = 0.1
eta_small = 0.01
eta_large = 0.45

w = np.arange(-10., 10, 0.2)
plt.ylabel('E')
plt.xlabel('weights')
plt.plot(w, 0.5*sigma**2 - r_xd*w + 0.5*r_x*w**2, 'go')
plt.show()

def steepest_descent(eta):
    err = 100
    #estimate from plot above:
    w_init = 2
    w = w_init
    weights = np.array([])
    Es = np.array([])
    iterations = 0

    while(abs(err) > 0.0001):
        iterations += 1
        E = 0.5*sigma**2 - r_xd*w + 0.5*r_x*w**2
        g = r_x*w - r_xd
```

```
        E_upd = E - eta * (LA.norm(g))**2
        w_upd = w - eta * g
        weights = np.append(weights, w)
        Es = np.append(Es,E)
        err = w_upd - w
        w = w_upd
    print "minimum weight"
    print w
    print "number of iterations"
    print iterations

    w = np.arange(-1., 2.5, 0.05)
    plt.ylabel('E')
    plt.xlabel('weights')
    plt.plot(w, 0.5*sigma**2 - r_xd*w + 0.5*r_x*w**2, 'go', weights, Es, 'bd', weights, Es, 'k')
    plt.show()

steepest_descent(eta)
steepest_descent(eta_small)
steepest_descent(eta_large)
```