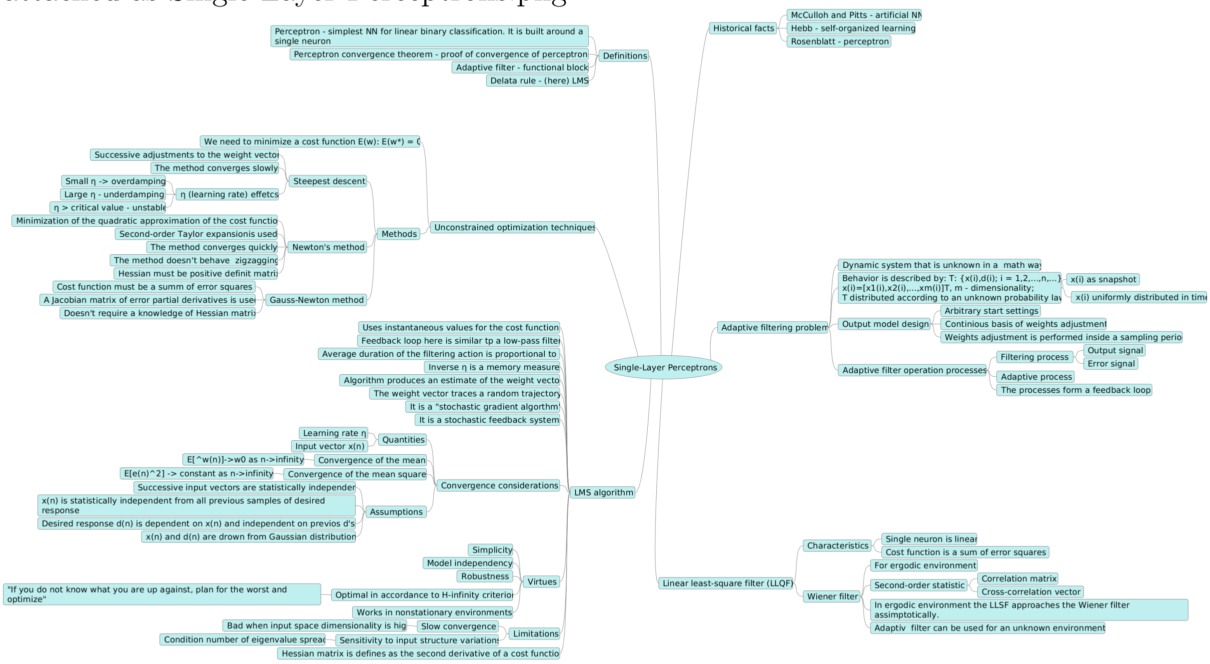
Neural Networks
- Homework 5 -

Petr Lukin, Evgeniya Ovchinnikova

Lecture date: 31 October 2016

1 Mind map

Figure 1: Mind map. Chapter 3 (first part) from Haykins book. A zoomed version is attached as Single-Layer Perceptrons.png



2 Exercises

2.1 Exercise 3.1

Explore the method of steepest descent involving a single weight w by considering the following cost function:

$$\varepsilon(w) = \frac{1}{2}\sigma^2 - r_{xd}w + \frac{1}{2}r_x w^2,$$

where σ^2, r_{xd} and r_x are constants.

Solution:

The steepest descent algorithm is describes as following:

$w(n+1) = w(n) - \eta g(n)$,
$\Delta w = -\eta g(n)$,

where η is a learning rate that is a positive constant and $g(n)$ is a gradient vector evaluated in w(n) point:

$g(n) = \nabla\varepsilon(w) = [\frac{\partial\varepsilon}{\partial w_1}, \frac{\partial\varepsilon}{\partial w_2}, ..., \frac{\partial\varepsilon}{\partial w_m}]^T$,

where m is dimensionality of the input space.

$\varepsilon(w(n+1)) \simeq \varepsilon(w(n)) - \eta||g(n)||^2$

Here:

$g(n) = -r_{xd} + r_x w$,

so:

$\Delta w = \eta(r_{xd} - r_x w)$,

$w(n+1) = w(n) + \eta(r_{xd} - r_x w)$,

$\varepsilon(w(n+1)) \simeq \varepsilon(w(n)) - \eta||g(n)||^2 = \frac{1}{2}\sigma^2 - r_{xd}w + \frac{1}{2}r_x w^2 - \eta||(r_x w - r_{xd})||^2$

To plot it we need to choose some values for the constants: $\sigma^2 = 2, r_{xd} = 3$ and $r_x = 4$.

To plot $\varepsilon(w)$ and paths with different η's (0.1, 0.01, 0.45) we used the following python code:

```
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)
```

The $\varepsilon(w)$ is shown in Fig. 2 and the results are depicted in Fig.3 - 5.
As we can see, the smoothest trajectory is with the smallest η - the transient response is overdamped, it converges quite slow. Large η shows zigzag behavior and if we continue to increase the η, the oscillations and a number of iterations will increase (see Fig.6). Finally, if we try to plot the path with $\eta \geqslant 0.5$ the algorithms diverges.
So, small ηs give a better result, but they might require too many iterations and one should be careful with large ηs because they can start oscillate.

Figure 2: $\varepsilon(w)$

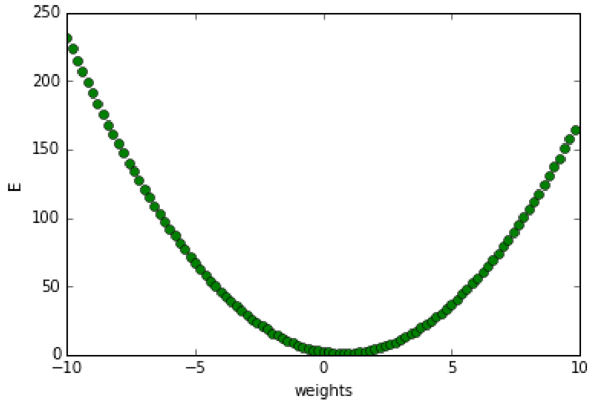


Figure 3: $\varepsilon(w)$ (green circles) and a weight path (blue diamonds connected with a line), $\eta = 0.1$. Minimum weight = 0.750126949946, number of iteration = 18

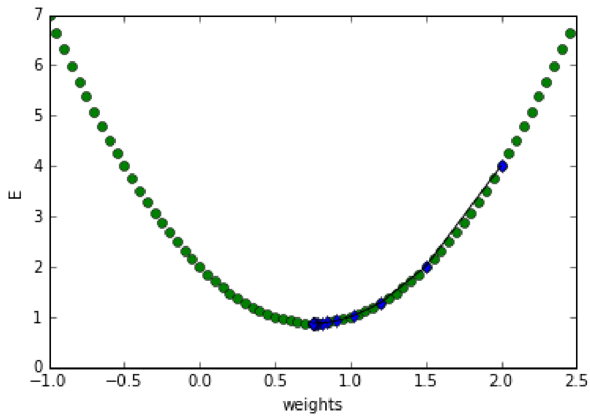


Figure 4: $\varepsilon(w)$ (green circles) and a weight path (blue diamonds connected with a line), $\eta = 0.01$. Minimum weight = 0.752326375959, number of iteration = 154

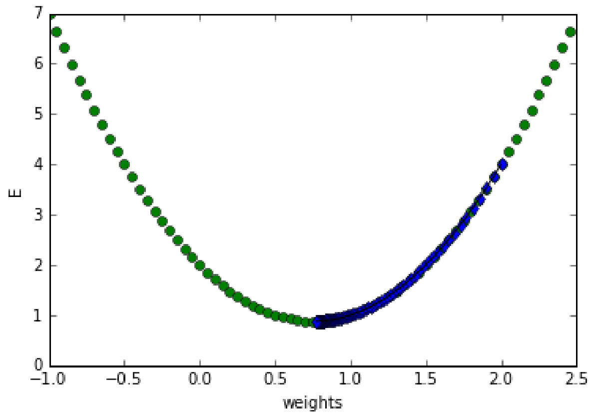


Figure 5: $\varepsilon(w)$ (green circles) and a weight path (blue diamonds connected with a line), $\eta = 0.45$. Minimum weight = 0.750043556143, number of iteration = 46

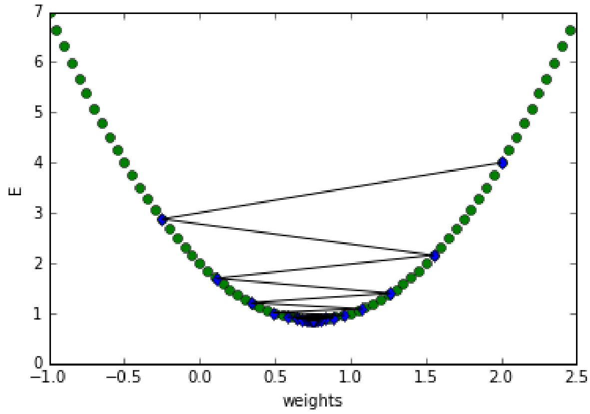
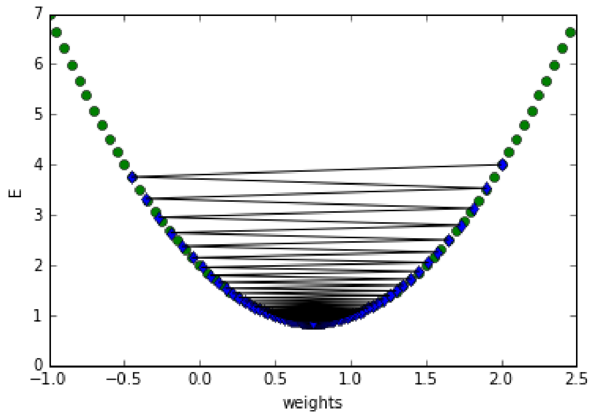


Figure 6: $\varepsilon(w)$ (green circles) and a weight path (blue diamonds connected with a line), $\eta = 0.49$. Minimum weight = 0.749951866545, number of iteration = 249



2.2 Exercise 3.2

Consider the cost function
$\varepsilon(\boldsymbol{w}) = \frac{1}{2}\sigma^2 - \boldsymbol{r}_{xd}^T\boldsymbol{w} + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{R}_x\boldsymbol{w}$,

where σ^2 is a constant and

$$\boldsymbol{r_{xd}} = \begin{pmatrix} 0.8182 \\ 0.354 \end{pmatrix}$$

$$\boldsymbol{R}_x = \begin{pmatrix} 1 & 0.8182 \\ 0.8182 & 1 \end{pmatrix}$$

(a) Find the optimum value $\boldsymbol{w}*$ for which $\varepsilon(\boldsymbol{w})$ reaches its minimum value.
(b) Use the method of steepest descent to compute $\boldsymbol{w}*$ for the following two values of learning-rate parameter: $\eta = 0.3$, $\eta = 1.0$. For each case, plot the trajectory by evolution of the weight vector $\boldsymbol{w}(n)$ in the W-plane.

Solution:

2.3 Exercise 3.4

The correlation matrix R_x of the input vector $x(n)$ in the LMS algorithm is defined by

$$R_x = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$$

Define the range of values for the learning-rate parameter η of the LMS algorithm for it to be convergent in the mean square.

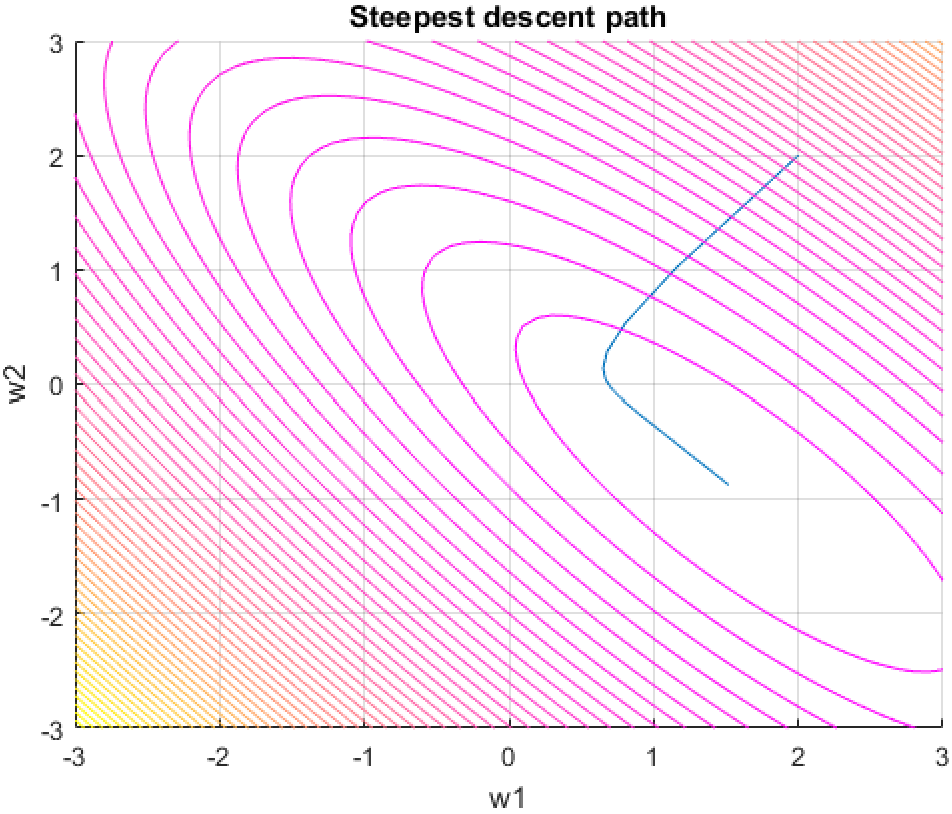
Solution:

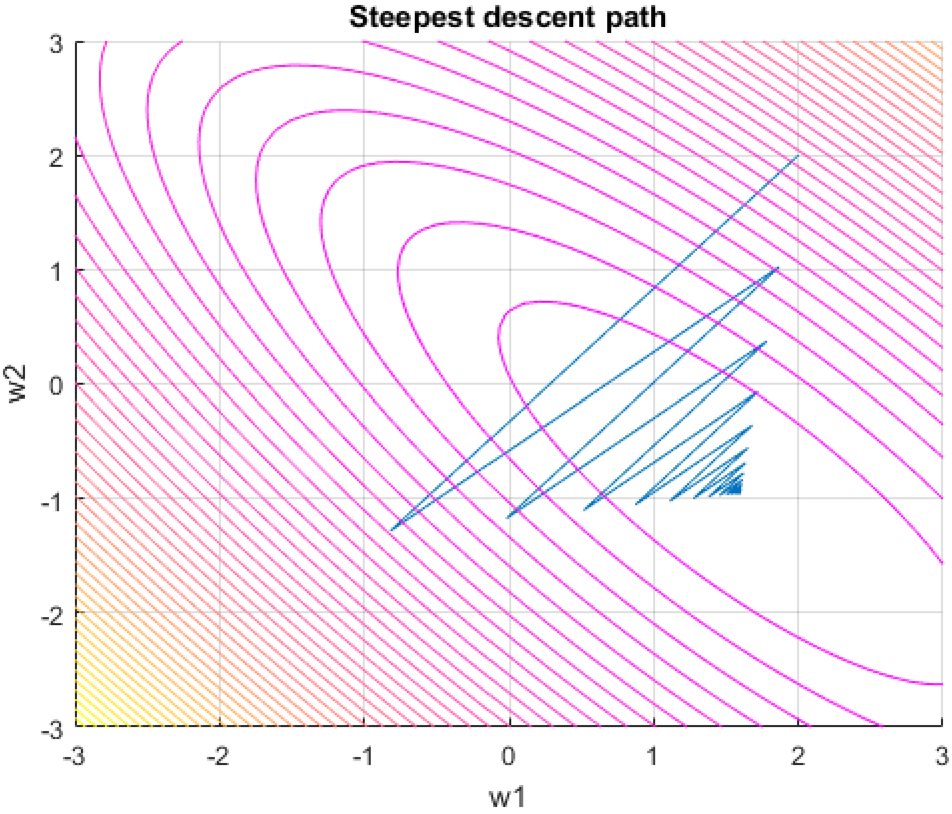
The algorithm converges if:

$0 < \eta < \frac{2}{\lambda_{max}}$,
where λ_{max} is the larges eigenvalue of the correlation matrix R_x.

$$R_x - \lambda I = \begin{vmatrix} 1-\lambda & 0.5 \\ 0.5 & 1-\lambda \end{vmatrix} = \lambda^2 - 2\lambda + 0.75,$$

so $\lambda_1 = 1.5$ and $\lambda_2 = 0.5$. Therefore, $\lambda_{max} = 1.5$ and:
$0 < \eta < \frac{2}{1.5} \Rightarrow 0 < \eta < 1.3333$
Countour plots for error function and descent trajectory for different η.





Optimal weight vector: $w* = [1.4779, -0.8332]$.

Matlab code for steepest descent:

```
%Initialize parameters
sigma = 1;
rxd = [0.8182 0.354];
R = [ 1 0.8182; 0.8182 1 ];

%Cost function evaluation
E = @(sigma,rxd,R,w) 0.5*sigma^2-rxd*w+0.5*transpose(w)*R*w;
eta = 0.3;
% eta=1.0;
% Two dimensional meshgrid for cost function
[ x1 , x2 ] = meshgrid (-3:0.1:3,-3:0.1:3);
z=zeros( length( x1 ), length( x2 ));
for i = 1:length ( x1 );
    for j = 1:length ( x2 );
        %cost function calculation
        z (i,j) = E(sigma,rxd,R,[x1(i,j); x2(i,j)]);
    end
end

%Steepest descent
%Starting point
w=[2;2];

delta = 100;
for k = 1:50;
if delta > 0.01;
    f = E(sigma,rxd,R,w); %Function evaluation
    g =-transpose(rxd) + R*w; %Gradient calculation for steepest
        descent
    f_new = f-eta*(norm(g))^2 ; %New function walue
    w_new = w-eta*g; %Weights update
    delta = w_new - w;
    w = w_new;
    delta = norm(delta);
else
    break
end

end

'Optimal weight'
w

figure(1)
hold on
plot(r,m)
```

```
[C, h ] = contour(x1,x2,z ,length(p)) ;
grid on
title('Steepest descent path')
xlabel('w1')
ylabel('w2')
colormap spring
hold off
```

2.4 Exercise 3.8

The ensemble-averaged counterpart to the sum of error squares viewed as a cost function is the mean-square value of the error signal:

$$J(w) = \frac{1}{2}E[e^2(n)] = \frac{1}{2}E[(d(n) - \mathbf{x}^T(n)\mathbf{w})^2]$$

Assuming that the input vector $x(n)$ and desired response $d(n)$ are drawn from a stationary environment, show that:

Assignment a)

Assuming that the input vector $x(n)$ and desired response $d(n)$ are drawn from a stationary environment, show that

$$J(w) = \frac{1}{2}\sigma_d^2 \mathbf{r}_{\mathbf{x}d}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{R}_{\mathbf{x}}\mathbf{w}$$

where

$$\sigma_d^2 = E[d^2(n)]$$

$$\mathbf{r}_{\mathbf{x}d} = E[\mathbf{x}(n)d(n)]$$

$$\mathbf{R}_{\mathbf{x}} = E[\mathbf{x}(n)\mathbf{x}^T(n)]$$

Solution a)

$$J(w) = \tfrac{1}{2}E[(d(n) - \mathbf{x}^T(n)\mathbf{w})^2] = \tfrac{1}{2}E[d(n)^2 - 2d(n)\mathbf{x}^T(n)\mathbf{w} + \mathbf{x}^T(n)\mathbf{w}\mathbf{x}^T(n)\mathbf{w}] =$$

$$= \tfrac{1}{2}(E[d(n)^2] - 2E[\mathbf{x}(n)d(n)]\mathbf{w} + E[\mathbf{w}^T\mathbf{x}(n)\mathbf{x}(n)^T\mathbf{w}]) =$$

$$= \tfrac{1}{2}\sigma_d^2 - \mathbf{r}_{\mathbf{x}d}^T\mathbf{w} + \mathbf{w}^T E[\mathbf{x}(n)\mathbf{x}(n)^T]\mathbf{w} = \tfrac{1}{2}\sigma_d^2 - \mathbf{r}_{\mathbf{x}d}^T\mathbf{w} + \tfrac{1}{2}\mathbf{w}^T\mathbf{R}_{\mathbf{x}}\mathbf{w} = J(w).$$

Assignment b)

For this cost function, show that the gradient vector and Hessian matrix of $J(w)$ are as follows, respectively:

$$g = -\mathbf{r}_{\mathbf{x}d} + \mathbf{R}_{\mathbf{x}}\mathbf{w}$$
$$\mathbf{H} = \mathbf{R}_{\mathbf{x}}$$

Solution b)

Gradient vector:

$$\mathbf{g} = \frac{\partial J}{\partial w} = \frac{\partial(\frac{1}{2}\sigma_d^2 - \mathbf{r_x}_d^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{R_x}\mathbf{w})}{\partial w} = 0 - \mathbf{r}_{\mathbf{x}d} + \mathbf{R_x}\mathbf{w} = -\mathbf{r}_{\mathbf{x}d} + \mathbf{R_x}\mathbf{w}.$$

Hessian matrix:

$$\mathbf{H} = \frac{\partial \mathbf{g}}{\partial w} = \frac{\partial(-\mathbf{r_x}_d^T + \mathbf{R_x}\mathbf{w})}{\partial w} = \mathbf{R_x}.$$

Assignment c)

In the LMS/Newton algorithm, the gradient vector g is replaced by its instantaneous value (Widrow and Strearns,1985).Show that this algorithm, incorporating a learning-rate parameter η, is described by

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{R_x}^{-1}\mathbf{x}(n)(d(n) - \mathbf{x}^T(n)\mathbf{w}(n)).$$

The inverse of the correlation matrix $\mathbf{R_x}$, assumed to be positive definite, is calculated ahead of time.

Solution c)

In the Newton's method we have the following rule:

$$w_{k+1} = w_k - \eta\mathbf{R}^{-1}\mathbf{g},$$

replace g with instantaneous values by omitting Expectations:

$$\hat{g} = -\mathbf{r}_{\mathbf{x}d} + \mathbf{R_x}\mathbf{w} = -\mathbf{x}(n)d(n) + \mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n).$$

So, LMS/Newton algorithm will be:

$$\begin{aligned}
&\hat{w}_{k+1} = \hat{w}_k - \eta\mathbf{R}^{-1}\hat{g} = \\
&= \hat{w}_k - \eta\mathbf{R}^{-1}(-\mathbf{x}(n)d(n) + \mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n)) = \\
&= \hat{\mathbf{w}}(n) + \eta\mathbf{R_x}^{-1}\mathbf{x}(n)(d(n) - \mathbf{x}^T(n)\mathbf{w}(n)).
\end{aligned}$$