

Neural Networks

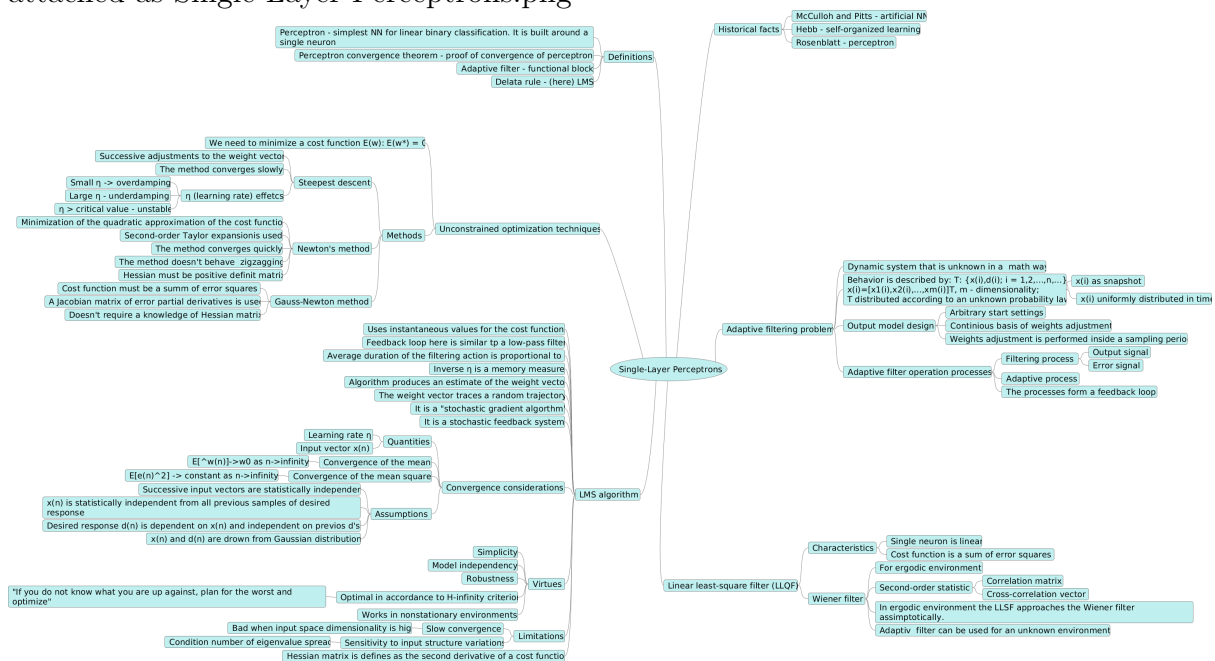
- Homework 5 -

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:

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

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:

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

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:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from numpy import linalg as LA
4
5 %matplotlib inline
6
7 #constants:
8
9 sigma = 2
10 r_xd = 3
11 r_x = 4
12 eta = 0.1
13 eta_small = 0.01
14 eta_large = 0.45
15
16 w = np.arange(-10., 10, 0.2)
17 plt.ylabel('E')
18 plt.xlabel('weights')
```

```

19 plt.plot(w, 0.5*sigma**2 - r_xd*w + 0.5*r_x*w**2, 'go')
20 plt.show()
21
22 def steepest_descent(eta):
23     err = 100
24     #estimate from plot above:
25     w_init = 2
26     w = w_init
27     weights = np.array([])
28     Es = np.array([])
29     iterations = 0
30
31     while(abs(err) > 0.0001):
32         iterations += 1
33         E = 0.5*sigma**2 - r_xd*w + 0.5*r_x*w**2
34         g = r_x*w - r_xd
35         E_upd = E - eta * (LA.norm(g))**2
36         w_upd = w - eta * g
37         weights = np.append(weights, w)
38         Es = np.append(Es, E)
39         err = w_upd - w
40         w = w_upd
41         print "minimum weight"
42         print w
43         print "number of iterations"
44         print iterations
45
46     w = np.arange(-1., 2.5, 0.05)
47     plt.ylabel('E')
48     plt.xlabel('weights')
49     plt.plot(w, 0.5*sigma**2 - r_xd*w + 0.5*r_x*w**2, 'go',
50             weights, Es, 'bd', weights, Es, 'k')
51     plt.show()
52
53 steepest_descent(eta)
54 steepest_descent(eta_small)
55 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 $\eta \geq 0.5$ the algorithms diverges.

So, small η s give a better result, but they might require too many iterations and one should be careful with large η s because they can start oscillate.

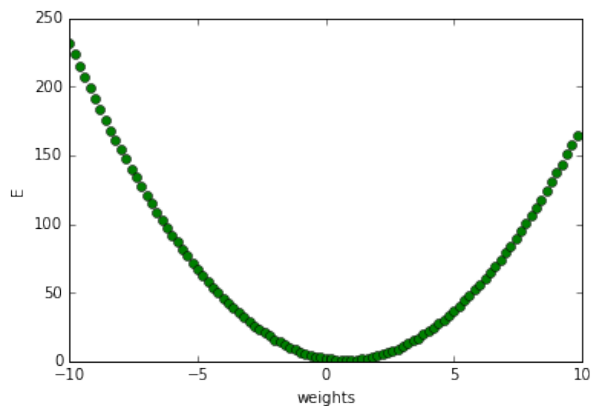
Figure 2: $\varepsilon(w)$ 

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

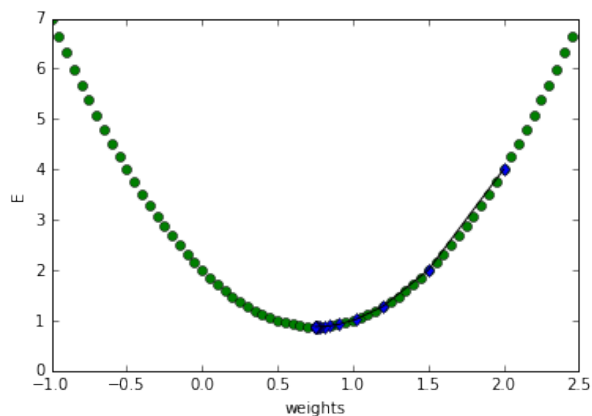


Figure 4: $\varepsilon(w)$ (green circles) and a weight path (blue diamonds connected with a line), $\eta = 0.01$. Minimum weight = 0.752326375959, number of iteration = 154

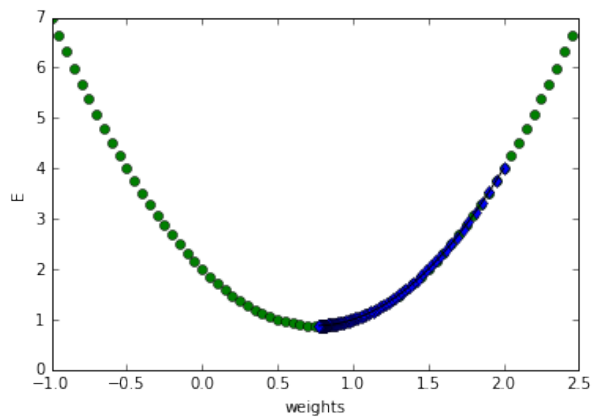


Figure 5: $\varepsilon(w)$ (green circles) and a weight path (blue diamonds connected with a line), $\eta = 0.45$. Minimum weight = 0.750043556143, number of iteration = 46

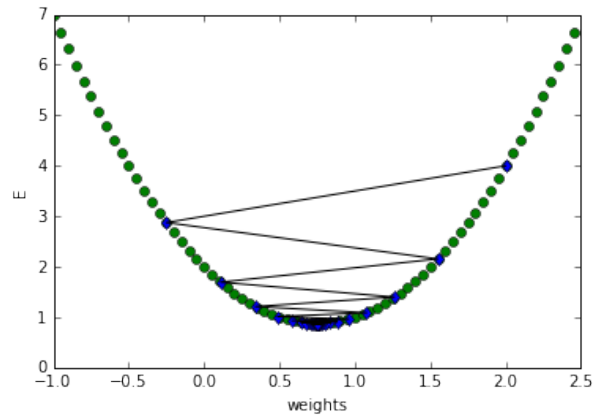
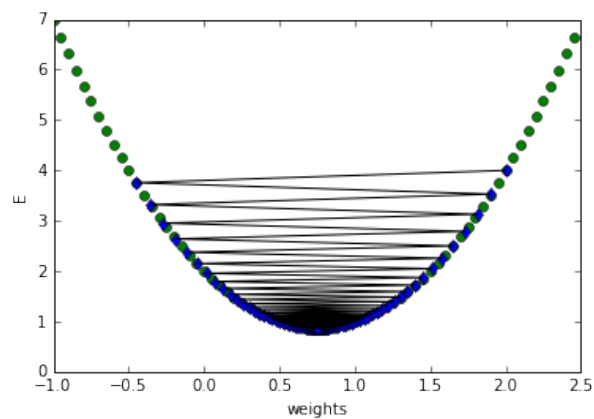


Figure 6: $\varepsilon(w)$ (green circles) and a weight path (blue diamonds connected with a line), $\eta = 0.49$. Minimum weight = 0.749951866545, number of iteration = 249



2.2 Exercise 3.2

Consider the cost function

$$\varepsilon(\mathbf{w}) = \frac{1}{2}\sigma^2 - \mathbf{r}_{xd}^T \mathbf{w} + \frac{1}{2}\mathbf{w}^T \mathbf{R}_x \mathbf{w},$$

where σ^2 is a constant and

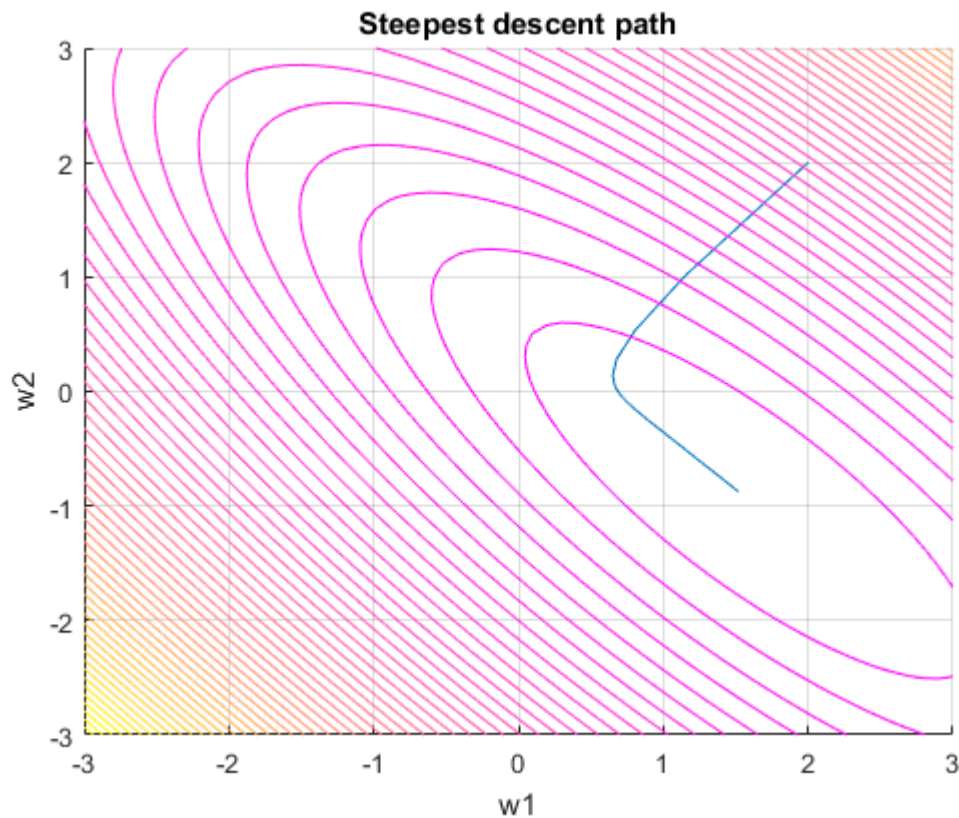
$$\mathbf{r}_{xd} = \begin{pmatrix} 0.8182 \\ 0.354 \end{pmatrix}$$

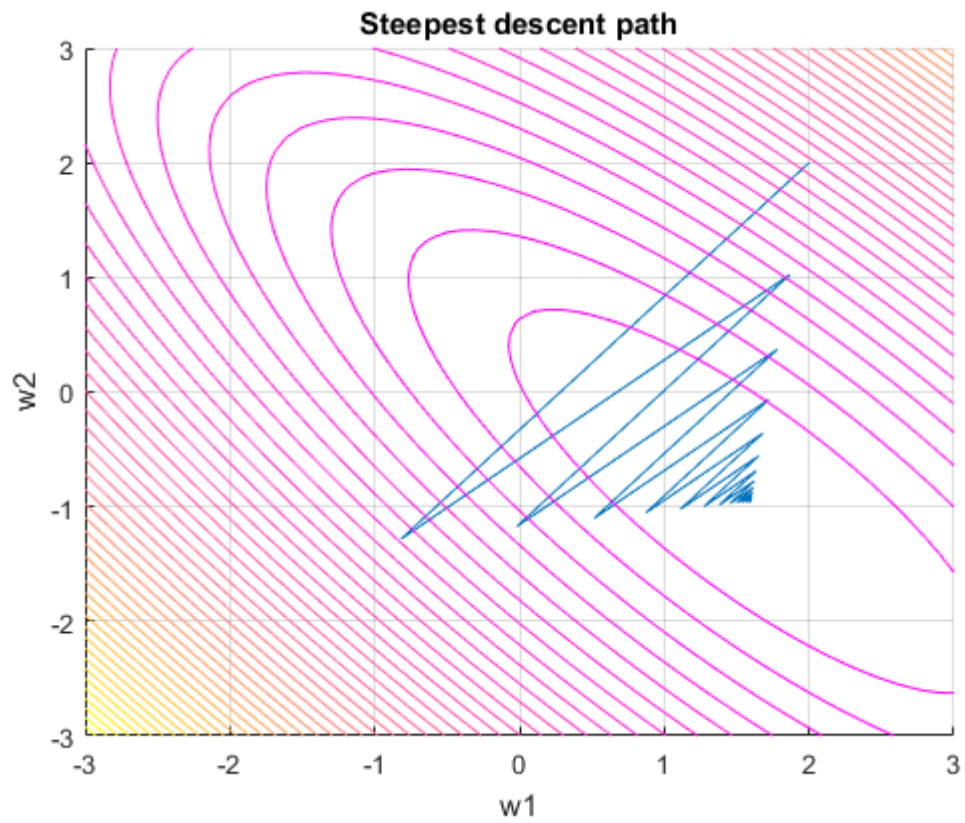
$$\mathbf{R}_x = \begin{pmatrix} 1 & 0.8182 \\ 0.8182 & 1 \end{pmatrix}$$

- Find the optimum value \mathbf{w}^* for which $\varepsilon(\mathbf{w})$ reaches its minimum value.
- Use the method of steepest descent to compute \mathbf{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 $\mathbf{w}(n)$ in the W-plane.

Solution:

Countour plots for error function and descent trajectory for different η .





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

Matlab code for steepest descent:

```

1 %Initialize parameters
2 sigma = 1;
3 rxd = [0.8182 0.354];
4 R = [ 1 0.8182; 0.8182 1 ];
5
6 %Cost function evaluation
7 E = @(sigma,rxid,R,w) 0.5*sigma^2-rxd*w+0.5*transpose(w)*R*w;
8 eta = 0.3;
9 % eta=1.0;
10 % Two dimensional meshgrid for cost function
11 [ x1 , x2 ] = meshgrid (-3:0.1:3,-3:0.1:3);
12 z=zeros( length( x1 ), length( x2 ));
13 for i = 1:length ( x1 );
14     for j = 1:length ( x2 );
15         %cost function calculation
16         z (i,j) = E(sigma,rxid,R,[x1(i,j); x2(i,j)]);
17     end
18 end
19
20 %Steepest descent
21 %Starting point
22 w=[2;2];
23
24 delta = 100;
25 for k = 1:50;
26     if delta > 0.01;
27         f = E(sigma,rxid,R,w); %Function evaluation
28         g =-transpose(rxd) + R*w; %Gradient calculation for steepest
           descent
29         f_new = f-eta*(norm(g))^2 ; %New function value
30         w_new = w-eta*g; %Weights update
31         delta = w_new - w;
32         w = w_new;
33         delta = norm(delta);
34     else
35         break
36     end
37
38 end
39
40 'Optimal weight '
41 w
42
43 figure(1)
44 hold on
45 plot(r,m)

```



```

46 [C, h] = contour(x1,x2,z,length(p)) ;
47 grid on
48 title('Steepest descent path')
49 xlabel('w1')
50 ylabel('w2')
51 colormap spring
52 hold off

```

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 largest 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:

$$0 < \eta < \frac{2}{1.5} \Rightarrow 0 < \eta < 1.3333$$

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

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

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

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

Solution a)

$$\begin{aligned} J(w) &= \frac{1}{2}E[(d(n) - \mathbf{x}^T(n)\mathbf{w})^2] = \frac{1}{2}E[d(n)^2 - 2d(n)\mathbf{x}^T(n)\mathbf{w} + \mathbf{x}^T(n)\mathbf{w}\mathbf{x}^T(n)\mathbf{w}] = \\ &= \frac{1}{2}(E[d(n)^2] - 2E[\mathbf{x}(n)d(n)]\mathbf{w} + E[\mathbf{w}^T\mathbf{x}(n)\mathbf{x}(n)^T\mathbf{w}]) = \\ &= \frac{1}{2}\sigma_d^2 - \mathbf{r}_{xd}^T\mathbf{w} + \mathbf{w}^TE[\mathbf{x}(n)\mathbf{x}(n)^T]\mathbf{w} = \frac{1}{2}\sigma_d^2 - \mathbf{r}_{xd}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{R}_x\mathbf{w} = J(w). \end{aligned}$$

Assignment b)

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

$$\begin{aligned} g &= -\mathbf{r}_{xd} + \mathbf{R}_x\mathbf{w} \\ \mathbf{H} &= \mathbf{R}_x \end{aligned}$$

Solution b)

Gradient vector:

$$\mathbf{g} = \frac{\partial J}{\partial w} = \frac{\partial(\frac{1}{2}\sigma_d^2 - \mathbf{r}_{xd}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{R}_x\mathbf{w})}{\partial w} = 0 - \mathbf{r}_{xd} + \mathbf{R}_x\mathbf{w} = -\mathbf{r}_{xd} + \mathbf{R}_x\mathbf{w}.$$

Hessian matrix:

$$\mathbf{H} = \frac{\partial \mathbf{g}}{\partial w} = \frac{\partial(-\mathbf{r}_{xd}^T + \mathbf{R}_x\mathbf{w})}{\partial w} = \mathbf{R}_x.$$

Assignment c)

In the LMS/Newton algorithm, the gradient vector g is replaced by its instantaneous value (Widrow and Stearns, 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:

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

So, LMS/Newton algorithm will be:

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