NN_DebarajBarua_NareshKumarGurulingan_061117

November 11, 2017

1 Hochschule Bonn-Rhein-Sieg

2 Neural Networks, WS17/18

3 Assignment 05 (06-november-2017)

3.1 Debaraj Barua, Naresh Kumar Gurulingan

```
In [183]: import numpy as np
          import matplotlib.pyplot as plt
          import sympy as sp
          import IPython
          sp.init_printing()
```

3.2 Question 1:

Read Haykin NN edition 2 : chapter 3.1-3.5. Make a summary

3.3 Question 2 (Haykin Edition:2, Ex: 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.

Answer 2:

```
In [184]: s,r_xd,r_x,w=sp.symbols('s,r_xd,r_x,w')

          e=1./2.*s**2-r_xd*w+1./2.*r_x*w**2

          nabla=sp.diff(e,w)
          nabla
```

Out[184]:

$$1.0r_x w - r_{xd}$$

So, Gradient,

$$\nabla\varepsilon(w) = \frac{\partial\varepsilon(w)}{\partial w}$$
$$= -r_{xd} + r_x w$$

According to steepest descent method, weight vector is updated in the direction of the steepest descent, i.e., in the direction opposite to the gradient vector.

So, $\Delta w = -\eta\nabla\varepsilon(w)$, where η is the learning rate

Now, updated weight is:

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

3.4 Question 3 (Haykin Edition:2, Ex: 3.2):

Consider the cost function

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

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

$$r_{xd} = \begin{bmatrix} 0.8182 \\ 0.354 \end{bmatrix}$$

$$R_x \begin{bmatrix} 1 & 0.8182 \\ 0.8182 & 1 \end{bmatrix}$$

3.4.1 3(a)

Find the optimum value of w^* for which $\varepsilon(w)$ reaches it's maximum value.

Answer 3(a): $\varepsilon(w)$ will reach maximum value when it's Jacobian will be zero.

```
In [185]: s=sp.symbols('s')
          w=sp.MatrixSymbol('w',2,1)
          r_xd=sp.Matrix([[0.8182],[0.354]])
          R_x=sp.Matrix([[1,0.8182],[0.8182,1]])

          e=1./2.*s**2-r_xd.T.dot(sp.Matrix(w))+1./2.*sp.Matrix(w).T.dot(R_x.dot(sp.Matrix(w)))
          e=sp.Matrix([e])
          symbolic_jacobian=e.jacobian(w)
          print "Jacobian Matrix is:"
          symbolic_jacobian
```

```
Jacobian Matrix is:
```

Out[185]:

$$\left[\begin{array}{cc}-0.8182 + 0.5 \left(2 w_{0,0} + 1.6364 w_{1,0}\right) & -0.354 + 0.5 \left(1.6364 w_{0,0} + 2 w_{1,0}\right)\end{array}\right]$$

```
In [186]: print "optimum values of weight, w*:"
          sp.Matrix([sp.solve(symbolic_jacobian,sp.Matrix(w))])
```

```
optimum values of weight, w*:
```

Out[186]:

$$\left[\begin{array}{c}\left\{ w_{0,0} : 1.59902944424901, \quad w_{1,0} : -0.954325891284541\right\}\end{array}\right]$$

3.4.2 3(b)

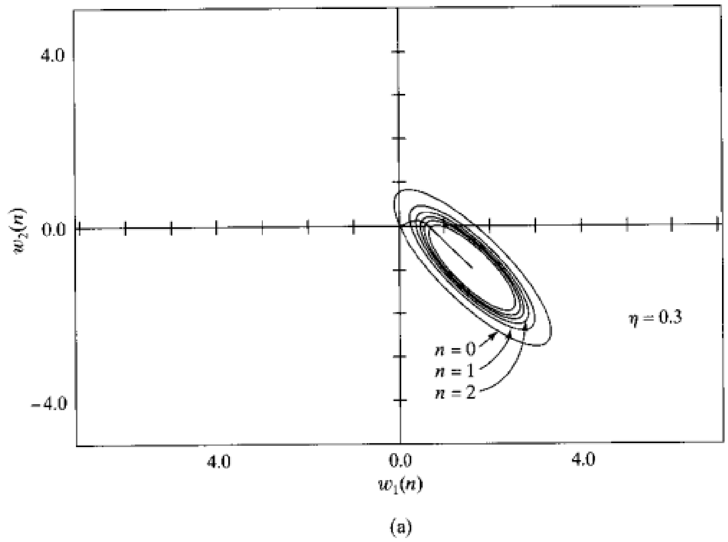
Use the method of steepest descent to compute w^* for the following two values of learning rate parameter:

<p>(i) $\eta=0.3$</p>
<p>(ii) $\eta=1.0$</p>

For each case, plot the trajectory traced the evolution of the weight vector $w(n)$ in the W-Plane.
Note: The trajectories obtained for cases (i) and (ii) of part (b) should correspond to the picture in Fig. 3.2

```
In [187]: IPython.display.Image("images/HaykinFig3.2.png")
```

Out[187]:



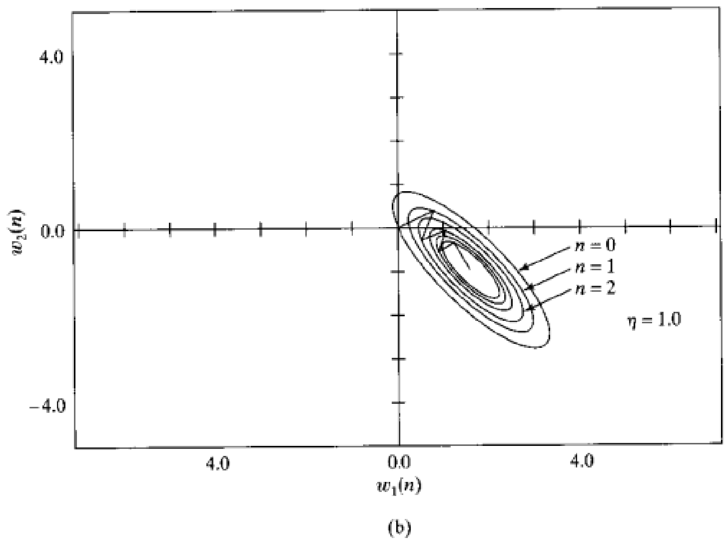


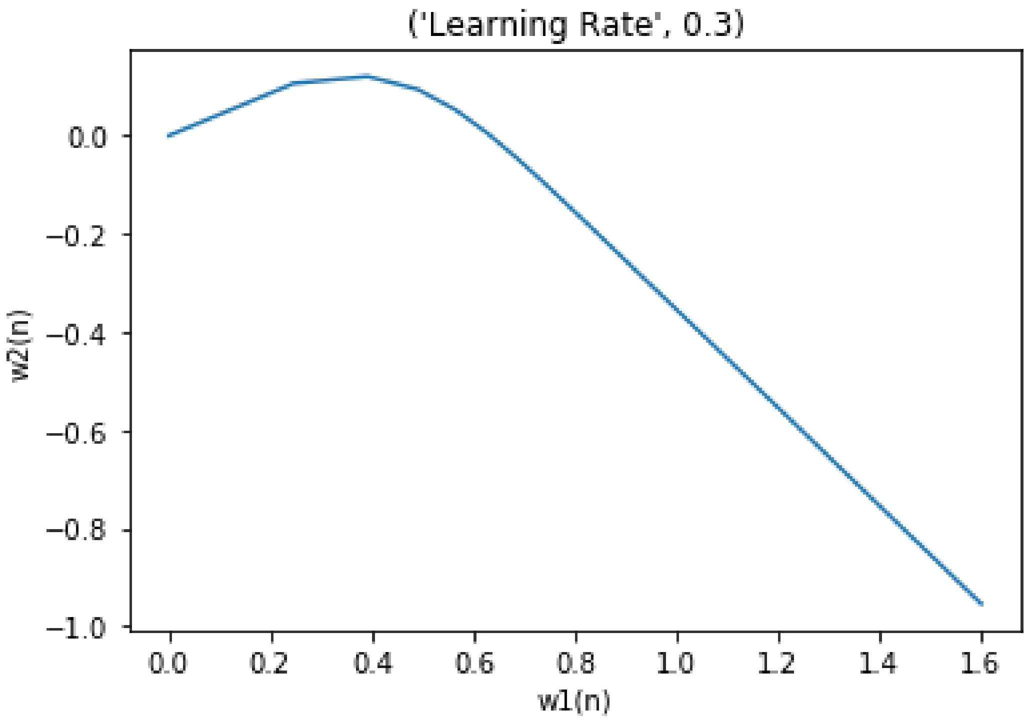
FIGURE 3.2 Trajectory of the method of steepest descent in a two-dimensional space for two different values of learning-rate parameter: (a) $\eta = 0.3$, (b) $\eta = 1.0$. The coordinates w_1 and w_2 are elements of the weight vector **w**.

Answer 3(b):

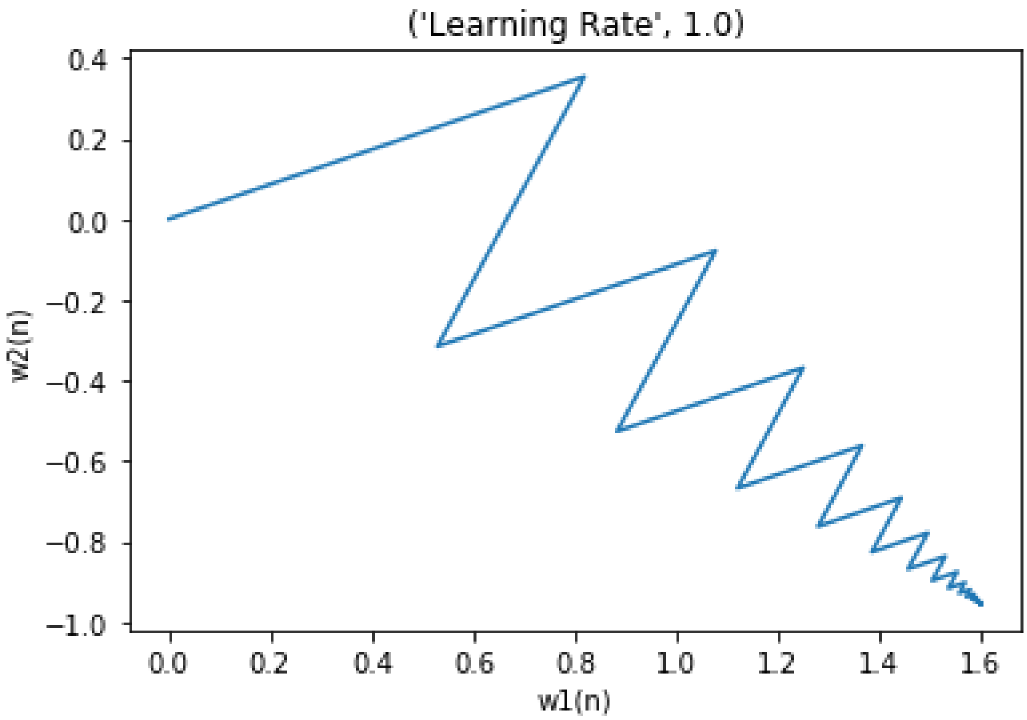
```
In [188]: #Lambda function for substitution of weight into symbolic jacobian matrix.
          delta_wt_func=sp.lambdify((w),symbolic_jacobian)

          def steepest_desc(weight, eta, max_iteration):
              found_optimum=False
              iteration=0
              wt_matrix=np.array(weight).T
```

```
        print "Learning Rate: ", eta
        while (found_optimum==False and iteration<max_iteration):
            iteration+=1
            delta_wt=delta_wt_func(weight)

            if(abs(delta_wt.all())<1e-5):
                """
                If absolute change in weight is less than 10^(-5),
                we consider we have reached the optimal solution
                """
                print "Reached optimum value"
                print "Absolute change in weight is less than 10^(-5), exiting loop."
                found_optimum=True
                break
            else:
                weight+=-eta*delta_wt.T
                wt_matrix=np.vstack((wt_matrix,weight.T))

        if iteration==max_iteration:
            print "Max iteration reached!"
        print "optimum values of weight, w*:",np.array(weight).T
        print "Total iterations required:",iteration
        print wt_matrix.shape
        plt.plot(wt_matrix[:,0],wt_matrix[:,1])
        plt.xlabel("w1(n)")
        plt.ylabel("w2(n)")
        plt.title(("Learning Rate",eta))
        plt.show()

    initial_weight = sp.Matrix([[0],[0]])
    print initial_weight.shape
    eta1 = 0.3
    eta2 = 1.0
    max_iteration =1000
    steepest_desc(initial_weight,eta1,max_iteration)
    steepest_desc(initial_weight,eta2,max_iteration)
```

```
(2, 1)
Learning Rate:  0.3
Reached optimum value
Absolute change in weight is less than 10^(-5), exiting loop.
optimum values of weight, w*: [[1.59897673946915 -0.954273186504682]]
Total iterations required: 181
(181, 2)
```




```
Learning Rate:  1.0
Reached optimum value
Absolute change in weight is less than 10^(-5), exiting loop.
optimum values of weight, w*: [[1.59899794732667 -0.954307093426509]]
Total iterations required: 55
(55, 2)
```



3.5 Question 4 (Haykin Edition:2, Ex: 3.4):

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

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

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

Answer 4: LMS algorithm is convergent in mean square in the below mentioned region:

$$0 < \eta < \frac{2}{\lambda_{max}}$$

Where, λ_{max} is the largest eigen value of the correlation matrix of the input vector.

```
In [191]: R_x=np.array([[1,0.5],[0.5,1]])

          eigval,eigvec=np.linalg.eig(R_x)
          max_eigvalue=eigval[np.argmax(eigval)]

          print "Ramge of values for learning rate: [0,",2./max_eigvalue,"]"
```

```
Ramge of values for learning rate: [0, 1.33333333333 ]
```

3.6 Question 5 (Haykin Edition:2, Ex: 3.8):

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

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

3.6.1 5(a)

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

$$J(w) = \frac{1}{2}\sigma_d^2 - r_{xd}^T w + \frac{1}{2}w^T R_x w$$

where,

$$\begin{aligned} \sigma_d^2 &= E[d^2(n)] \\ r_{xd} &== E[x(n)d(n)] \\ R_x &== E[x(n)x^T(n)] \end{aligned}$$

Answer 5(a):

3.6.2 5(b)

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

$$\begin{aligned} g &= -r_{xd} + R_x w \\ H &= R_x \end{aligned}$$

Answer 5(b): Similar to Section **??**,

Gradient is given by differentiating with respect to $w(n)$,

$$g = \nabla\varepsilon(w) = -r_{xd} + R_x w$$

For Hessian, we differntiate it again to get,

$$H = \nabla\nabla\varepsilon(w) = R_x$$

3.6.3 5(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

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

The inverse of the correlation matrix R_x, assumed to be positive definite, is calculated ahead of time.

Answer 5(c):
