

NN_RubanrajRavichandran_06112017

November 11, 2017

```
In [1]: import numpy as np
import sympy as sp
```

1 Exercise 3.2 part (a)

```
In [2]: wo,w1 = sp.symbols("w0,w1")

w = sp.Matrix([wo,w1])
r = sp.Matrix([0.8182,0.354])
R = sp.Matrix([[1,0.8182],
               [0.8182,1]])
print "The weight vector w ="
sp.pprint(w)
print "\n r_xd = "
sp.pprint(r)
print "\nAnd R_x = "
sp.pprint(R)
```

The weight vector w =
w

w

r_xd =
0.8182

0.354

And R_x =
1 0.8182

0.8182 1

$\xi(\omega) = \frac{1}{2}\sigma^2 - r_{xd}^T\omega + \frac{1}{2}\omega^T R_x \omega$. Substituting the values of r_{xd} and R_x , the values of $r_{xd}^T\omega$ and $\frac{1}{2}\omega^T R_x \omega$ is:

```
In [13]: rw = r.T * w
         sp.pprint(rw)
```

```
[0.8182w + 0.354w]
```

```
In [14]: w_tRw = 0.5*w.T*R*w
         sp.pprint(w_tRw)
```

```
[w(0.5w + 0.4091w) + w(0.4091w + 0.5w)]
```

$$\text{Thus } \zeta(\omega) = \frac{1}{2}\sigma^2 - [0.8182\omega_0 + 0.354\omega_1] + [\omega_0(0.5\omega_0 + 0.4091\omega_1) + \omega_1(0.4091\omega_0 + 0.5\omega_1)]$$

$$\zeta(\omega) = \frac{1}{2}\sigma^2 + 0.5.\omega_0^2 + 0.5.\omega_1^2 + 0.8182.\omega_0.\omega_1 - 0.8182.\omega_0 - 0.354.\omega_1$$

```
In [15]: sigma = sp.Symbol('sigma')
         xi = 0.5*sigma**2 + (0.5*wo**2) + (0.5*w1**2) + (0.8182*wo*w1) - (0.8182*wo) - (0.354*w1)
         eqn1 = sp.diff(xi,wo)
         eqn2 = sp.diff(xi,w1)
         gradient = sp.Matrix([eqn1,eqn2])
         print "Gradient :"
         sp.pprint(gradient)
```

```
Gradient :
1.0w + 0.8182w - 0.8182
```

```
0.8182w + 1.0w - 0.354
```

```
In [16]: w_optimal = sp.solve([eqn1,eqn2],[wo,w1])
         print "The optimal weight vector = ",w_optimal
```

```
The optimal weight vector = {w0: 1.59902944424901, w1: -0.954325891284541}
```

2 Exercise 3.2 part (b)

```
In [17]: import matplotlib.pyplot as plt
         %matplotlib inline
```

```
def steepest_descent(eta, xi , gradient):
    w_init = np.array([0,0])

    w_n = w_init.T
    xi_n = xi.subs({wo:w_n[0],w1:w_n[1]})
    gradient_n = gradient.subs({wo:w_n[0],w1:w_n[1]})
```

```

w_0 = []
w_1 = []

for i in range(200):
    eta_gn = np.asarray(eta*gradient_n)
    w_new = w_n - np.array([eta_gn[0,0],eta_gn[1,0]])
    #Update the weight vector, the gradient and corresponding value of xi

    gradient_n = gradient.subs({w0:w_new[0],w1:w_new[1]})
    xi_new = xi.subs({w0:w_new[0],w1:w_new[1]})

    xi_n = xi_new
    w_n = w_new

    w_0.append(w_n[0])
    w_1.append(w_n[1])

w_0 = np.asarray(w_0)
w_1 = np.asarray(w_1)

plt.plot(w_0,w_1)
plt.xlabel('w0 -->')
plt.ylabel('w1 -->')
return w_n

```

```

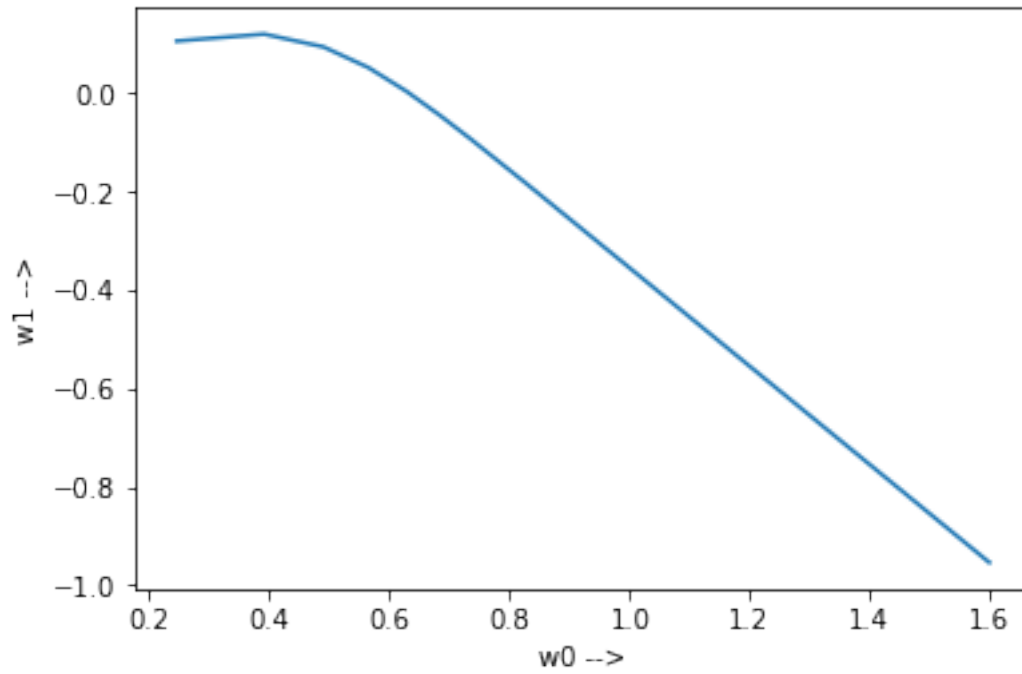
In [18]: #Learning rate = 0.3
         sp.pprint(steepest_descent(0.3,xi,gradient))

```

```

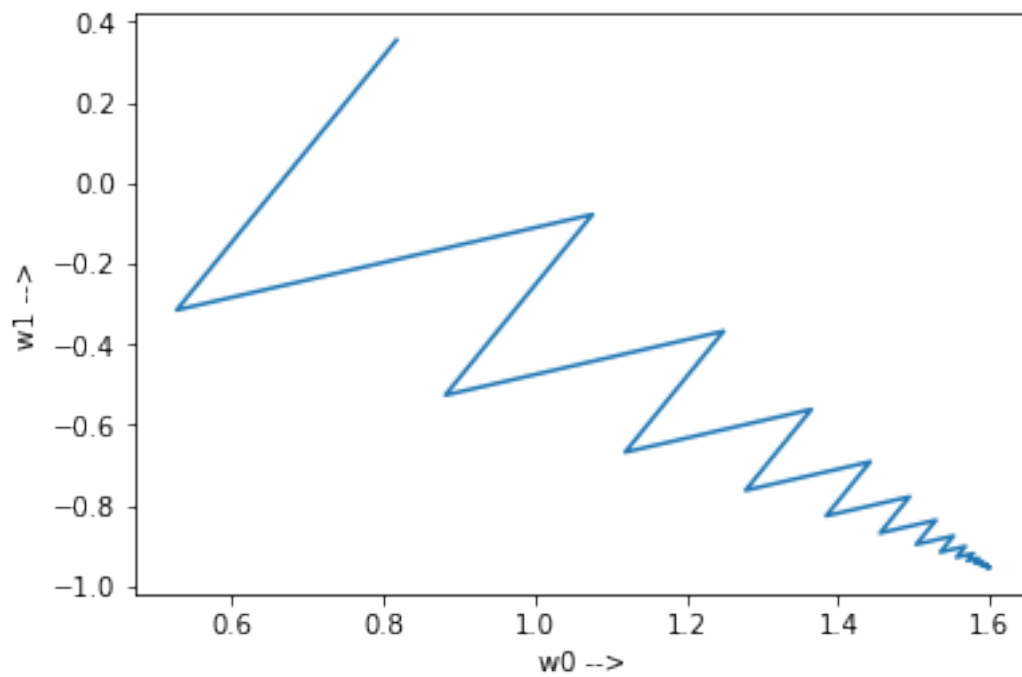
[1.59901227650625 -0.954308723541781]

```



```
In [19]: #Learning rate = 1.0
         sp.pprint(steepest_descent(1.0,xi,gradient))

[1.59902944424901 -0.954325891284542]
```



3 Exercise 3.4

Given correlation matrix R_x of the input vector $x(n)$ in the LMS algorithm

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

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

4 Solution

The learning rate η and the input vector $x(n)$ that determine the transmittance of the feedback loop. So, the convergence behaviour of the LMS algorithm is influenced by the statistical characteristics behaviour of the input vector and the value of learning parameter η .

As shown in Haykin (2006), that the LMS is convergent in the mean square provided that η satisfies the following condition, $0 < \eta < \frac{2}{\lambda_{max}}$, where λ_{max} is the largest eigen value of the correlation matrix R_x .

But in some typical applications of the LMS algorithm, the knowledge of λ_{max} is not available, so to avoid this problem we may take trace of R_x which is a conservative estimate of λ_{max} . So to be convergent in the mean square, the condition should be $0 < \eta < \frac{2}{tr[R_x]}$

```
In [20]: #using trace of correlation method
correlation_matrix = np.array([[1,0.5],[0.5,1]])
#trace calculation
correlation_matrix_trace = np.trace(correlation_matrix)
upper_range = 2/correlation_matrix_trace
print "Upper Range: " + str(upper_range)
```

Upper Range: 1.0

For the given correlation matrix, the learning rate parameter range from $0 < \eta < 1$