BONN-RHEIN-SIEG UNIVERSITY OF APPLIED SCIENCES
DEPARTMENT OF COMPUTER SCIENCE
NEURAL NETWORKS

Exercise 5

Poulastya Mukherjee, Benjamin Thompson

1 Read Chapter 3.1-3.5 from Haykin's book; summarize or sketch your insights in mind-map or an outline or a summary.

- Rosenblatt proved that given linearly separable data, a perceptron is proven to converge.
- Least mean square algorithm is the backbone of linear adaptive filters
- Adaptive filtering

  m dimensional input produces scalar output

  data equally distributed

  data can be spread over space (snapshot) or over time (uniformly spaced in time)

  Filtering process produces the output and error signals

  Adaptive process involves adjustments based on errors

  Error correction is an optimization problem
- Unconstrained optimization techniques

  Optimal solution is gradient of cost function equal to 0
- Steepest descent

  converges slowly

  size of eta produces overdamped response when small, under when large
- Newton method

  needs to be twice continuously differentiable wrt w to form hessian

  converges quickly and generally not subject to underdamped behavior of steepest descent

  Needs to be positive definite matrix, however there is no gaurantee of that.

- Gauss Newton method

  Only requires jacobian of the error vector as opposed to hessian of cost function

  Jacobian product must be non singular

- Least mean squares

  Inverse of the learning rate eta is the

  weight vector traverses random trajectory in contrast with steepest descent

  The stability of the system is determined by choosing an appropriate eta for x

  Model independent, therefore robust

  Needs approx 10x the dimensionality iterations to converge

2 (3.1)

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

Calculating Descent

$$\bigtriangledown\varepsilon(w) = g = r_x w - r_{xd}$$

Applying Steepest Descent

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

where g(n) is the gradient and η is the learning rate

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

By applying steepest descent we get the expression for the change in weight

$$\triangle w(n) = \eta.[r_x w(n) - r_{xd}]$$

3 (3.2)

3.1 a

$$w = \begin{bmatrix} w1 \\ w2 \end{bmatrix}$$

$$w^T = \begin{bmatrix} w1 & w2 \end{bmatrix}$$

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

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

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

After calculating the gradient of the equation and equating it to 0 we get the optimal values of
w^* as the point where the slope is zero is either minima or maxima

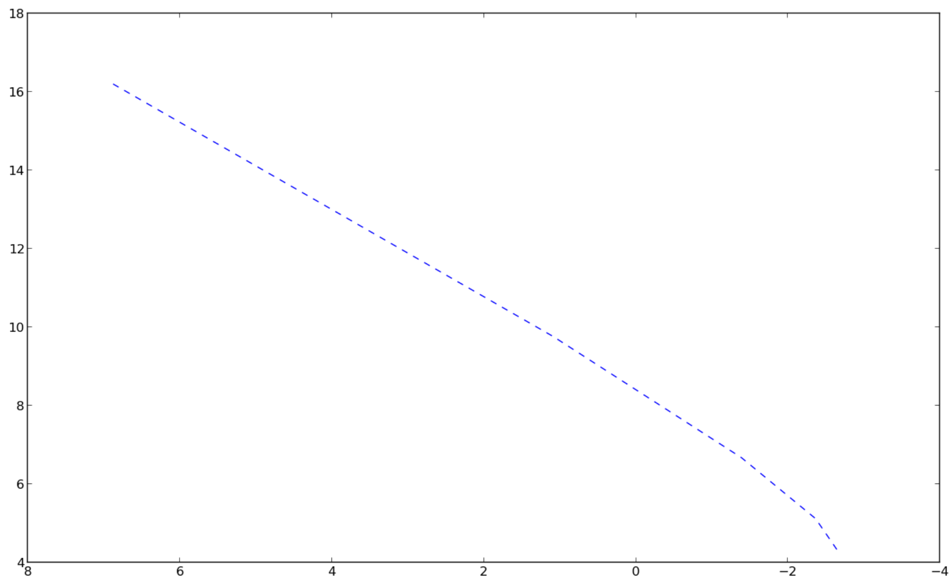
$$\nabla\varepsilon(w) = \begin{bmatrix} -0.8182 + w1 + 0.8182w2 \\ -0.354 + w1 + 0.8182w1 \end{bmatrix} = 0$$

If we differentiate the above equation again using the gradient we will see that the resulting matrix is same as R i.e. positive hence proving that the point we found is minima

$$w^* = \begin{bmatrix} 1.599 \\ -0.95 \end{bmatrix}$$

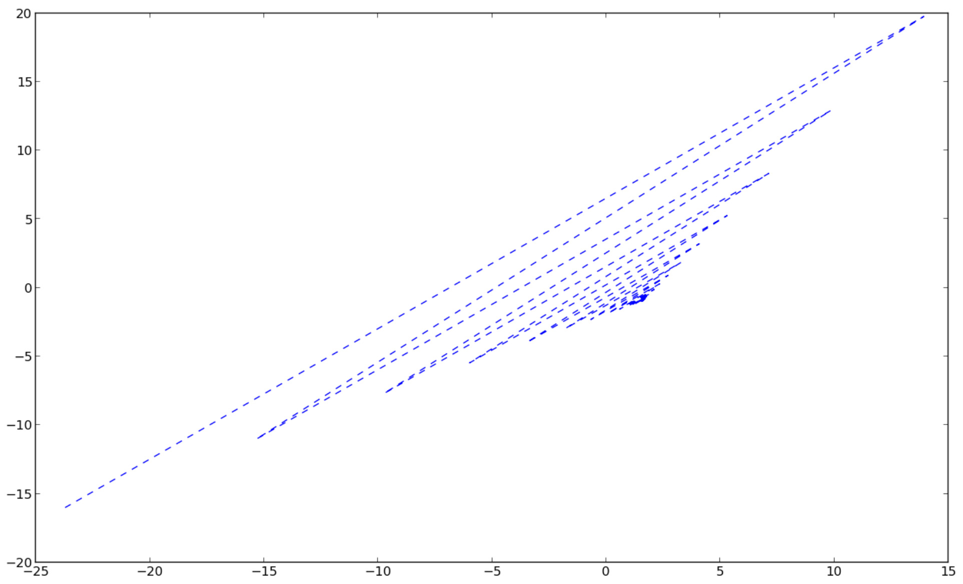
3.2 b

Code is provided at the end



$$w^* = \begin{bmatrix} 1.58164097 \\ -0.93693741 \end{bmatrix}$$

for learning rate 0.3



$$w^* = \begin{bmatrix} 1.5993919 \\ -0.95371617 \end{bmatrix}$$

for learning rate 1

So we can see varying the learning rate does not have effect on the optimal value as they are almost equal

4 (3.4)

The LMS algorithm converges for

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

where λ_{max} is the largest eigen value of the correlation matrix.
Given the correlation matrix

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

The SVD of R_x gives 1.5 as the the largest eigen value This particular LMS problem will converge for

$$0 < \eta < \frac{2}{1.5} = 1.33$$

5 (3.8)

5.1 (a)

Given

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

and E is a linear function,

$$J(\boldsymbol{w}) = \frac{1}{2}E\left[d^2(n) - 2\dot{(}n)\boldsymbol{x}(n)\boldsymbol{w} + \boldsymbol{x}(n)\boldsymbol{w}^T\boldsymbol{x}^T(n)\boldsymbol{w}\right]$$

$$= \frac{1}{2}E\left[d^2(n)\right] - \frac{1}{2}2E\left[d(n)\boldsymbol{x}(n)\right]\boldsymbol{w} + \frac{1}{2}\boldsymbol{w}^T E\left[\boldsymbol{x}(n)\boldsymbol{x}^T(n)\right]\boldsymbol{w}$$

$$= \frac{1}{2}\sigma_d^2 - \boldsymbol{r}_{\boldsymbol{x}d}^T\boldsymbol{w} + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{R}_{\boldsymbol{x}}\boldsymbol{w}$$

where

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

$$\boldsymbol{r}_{\boldsymbol{x}d} = E\left[\boldsymbol{x}(n)d(n)\right]$$

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

5.2 (b)

Taking the gradient wrt to $\boldsymbol{w}$ of

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

gives

$$\nabla J(\boldsymbol{w}) = -\boldsymbol{r}_{\boldsymbol{x}d} + \boldsymbol{R}_{\boldsymbol{x}}\boldsymbol{w} = g$$

derivating again wrt $\boldsymbol{w}$ we arrive at the Hessian

$$\boldsymbol{H} = \boldsymbol{R}_{\boldsymbol{x}}$$

because of the transpose relation of the two $\boldsymbol{w}$ terms and because

$$\boldsymbol{H}(f)(x) = \boldsymbol{J}(\nabla f)(x)$$

5.3 (c)

$$e(n) = d(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n)$$

Hence

$$\frac{\partial e(n)}{\partial \boldsymbol{w}(n)} = -\boldsymbol{x}(n)$$

the estimate of the gradient vector is then

$$\hat{\boldsymbol{g}}(n) = -\boldsymbol{x}(n)e(n)$$

and the steepest descent is described by

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \boldsymbol{g}(n)$$

we can formulate

$$\hat{\boldsymbol{w}}(n+1) = \hat{\boldsymbol{w}}(n) + \eta \boldsymbol{x}(n) e(n)$$

which can be seen to equal

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

However, the $\boldsymbol{R_x}$ term is still not accounted for.

6 Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  7 12:33:53 2015

@author: poulastya
"""

import sympy
import numpy
import numpy as np
from numpy import *
from sympy import *
import math
import matplotlib.pyplot as plt
from matplotlib import *

"""
 Setting Initial weights at 20 and 30 and chaning the learning
rates
"""
w = np.matrix(([20],[30]))
l_r = 1
x = []
y = []

gradient = np.matrix(([l_r*(-0.8182+w.item(0)+0.8182*w.item(1))],

[l_r*(-0.354+w.item(1)+0.8182*w.item(0))]))

while (gradient.item(0) > 0.001 or gradient.item(0) < -0.001):
    w = w - gradient
    gradient = np.matrix(([l_r*(-0.8182+w.item(0)+0.8182*w.item(1))],
    [l_r*(-0.354+w.item(1)+0.8182*w.item(0))]]))
    #print w
    x.append(w.item(0))
    y.append(w.item(1))

print '####'
print w

plt.plot(x,y,'b--')

plt.show()
```