PRML HOMEWORK 1

TADIPATRI UDAY KIRAN REDDY
EE19BTECH11038

January 30, 2022

HW-1

Apply Newtons method to steepest-descent algorithm to the optimal step size η, and check how many iterations are required for convergence.

$$\overline{w}^{new} = \overline{w}^{old} + \eta \mathbf{X}^T(\mathbf{t} - \mathbf{X}\overline{w})|_{\overline{w}=\overline{w}^{old}}$$

We find that optimal step size η should be the inverse of Hessian matrix, $\left(\frac{\partial^2 J(\overline{w})}{\partial \overline{w} \partial \overline{w}^T}\right)^{-1} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}$.
Using substituting the optimal η in update equation we get,
#Iteration = 1

$$\overline{w}^{new} = \overline{w}^{old} + \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T(\mathbf{t} - \mathbf{X}\overline{w})|_{\overline{w}=\overline{w}^{old}}$$
$$\implies \overline{w}^{new} = \left(\mathbf{I} - \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{X}\right)\overline{w}_{old} + \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{t}$$
$$\implies \overline{w}_{new} = w^*$$

So theoritically only one iteration is sufficient with optimal eta to make the solution to converge to optimal weights.
The implementation of this algorithm is given here Appendix 1. The data generated here is just the equation

$$t = 3.879x - 65$$

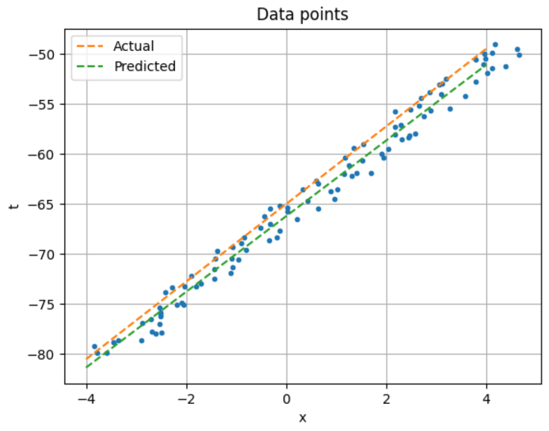


Figure 1: Data points

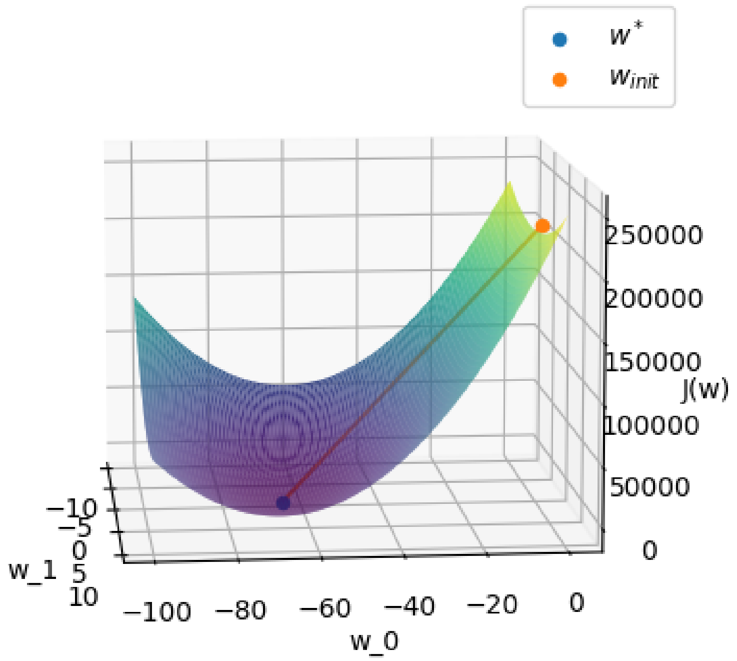
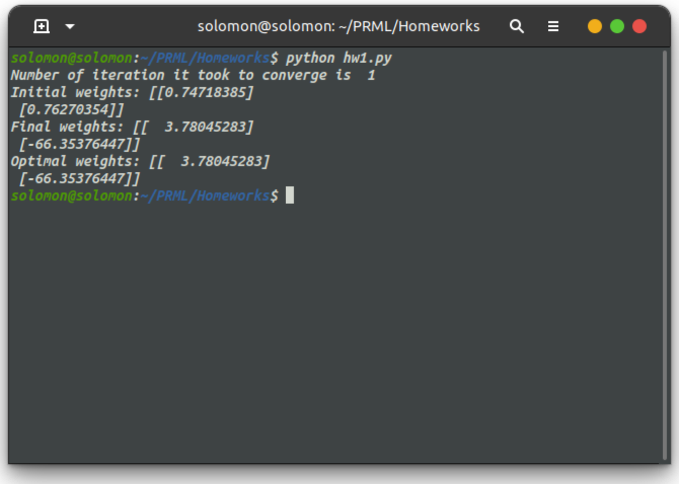


Figure 2: Error surface with path of steepest descent algorithm.

Observe that in Figure 2, it took only 1 iteration to converge to the desired solution, this has been proved mathematically as well.



```
solomon@solomon:~/PRML/Homeworks$ python hw1.py
Number of iteration it took to converge is  1
Initial weights: [[0.74718385]
 [0.76270354]]
Final weights: [[  3.78045283]
 [-66.35376447]]
Optimal weights: [[  3.78045283]
 [-66.35376447]]
solomon@solomon:~/PRML/Homeworks$
```

Figure 3: Output

HW-2

Suppose you are experimenting with L1 and L2 regularization. Further imagine that you are running gradient descent and at some iteration your weight vector is $\overline{w} = [1, \epsilon] \in R^2$ where $\epsilon > 0$ is very small. With the help of this example explain why L2 norm does not encourage sparsity i.e., it will not try to drive ϵ to 0 to produce a sparse weight vector. Give mathematical explanation.

Since $\overline{w} = \begin{bmatrix}1 & \epsilon\end{bmatrix}^T$ and $\epsilon > 0$.

$$L_1(\overline{w}) = ||\overline{w}||_1 \implies \frac{\delta L_1 \overline{w}}{\delta \overline{w}} = \begin{bmatrix}sgn(w_1) & sgn(w_2)\end{bmatrix} = \begin{bmatrix}1 & 1\end{bmatrix}$$
$$L_2(\overline{w}) = \frac{1}{2}||\overline{w}||_2^2 \implies \frac{\delta L_2 \overline{w}}{\delta \overline{w}} = \overline{w}^T$$

By applying GDA on this we get, for brevity we are ignoring the MSE cost since it is a common term for both

$$\overline{w}_{L1} = \begin{bmatrix}1 - \eta\lambda \\ \epsilon - \eta\lambda\end{bmatrix}$$
$$\overline{w}_{L2} = (1 - \eta\lambda)\begin{bmatrix}1 \\ \epsilon\end{bmatrix}$$

Since ϵ is very small and it is most likely to tend to zero than 1 in L1 regulatisation but where as in the case of L2 we see that both dimensions are equally scaled and also it's gradient depends upon current weights. In L1 even if second diemension is negative the gradients are -1 and it tries to bring it to zero. But in the case L2 it just scale down the weights.
Thus we can say that L2 norm does'nt encourage sparsity compared to L1.

HW-3

Till now we have been considering a scalar target t from a vector of input observations $\overline{x}$. How do you extend this approach for regressing a vector of targets $\overline{t} = (t1, t2, ..., tp)$. Derive the close form solutions and write sequential update equations using SGD.

Variable	Dimension
T	N X P
X	N X (D+1)
W	(D+1) X P
E	N X P

Where N is number of examples, P is output vector dimension and D is input vector dimension.

Taking our cost function as Forbenious norm(Appendix 1) of Error Matrix(**E**).

$$\mathbf{E} = \mathbf{T} - \mathbf{XW}$$

$$\boxed{J(\mathbf{W}) = \frac{1}{2}||\mathbf{E}||_F^2}$$

We obtain optimal weights by differentiating and equation to zero.

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = -\mathbf{X}^T\left(\mathbf{T} - \mathbf{X}\mathbf{W}\right) = 0$$

$$\implies \boxed{\mathbf{W}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{T}}$$

Now we obtain stochastic gradient descent equations,

$$\mathbf{W}_{(n+1)} = \mathbf{W}_{(n)} - \eta\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

$$\boxed{\mathbf{W}_{(n+1)} = \mathbf{W}_{(n)} + \eta\mathbf{X}^T\left(\mathbf{T} - \mathbf{X}\mathbf{W}\right)}$$

Using taylor series we find optimal η,

$$J(\mathbf{W} + \Delta\mathbf{W}) = J(\mathbf{W}) + \Delta\mathbf{W}^T\nabla J(\mathbf{W}) + \frac{1}{2}\Delta\mathbf{W}^T\nabla^2 J(\mathbf{W})\Delta\mathbf{W}$$

$$\frac{\partial J(\mathbf{W} + \Delta\mathbf{W})}{\partial \Delta\mathbf{W}} = \mathbf{0} = \nabla J(\mathbf{W}) + \nabla^2 J(\mathbf{W})\Delta\mathbf{W}$$

$$\implies \boxed{\Delta\mathbf{W}^* = -(\nabla^2 J(\mathbf{W}))^2\nabla J(\mathbf{W})}$$

$$\implies \boxed{\eta^* = (\mathbf{X}^T\mathbf{X})^{-1}}$$

Appendix

HW1 implementation

```
import numpy as np
import matplotlib.pyplot as plt

# Random linear data
def gen_data(N):
  X = np.linspace(-4, 4, N)
  X = X.reshape(N, 1)
  t = 3.879*X - 65
  t = t + np.random.rand(N, 1)
  X = X + np.random.rand(N, 1)
  return t, X

# J(w)
def cost(t, X, w):
  e = t - X@w
  return 0.5*e.T@e

N = 100

t, X_data = gen_data(N)

ones = np.ones((N, 1))
X = np.hstack((X_data, ones))

w = np.random.rand(2, 1)
w_init = w


# GDA
```

```python
def steepest_decent(t, X, w, eta):
  # print(w.shape, (X.T@(t-X@w)).shape)
  if type(eta) is np.ndarray:
    return w + eta@(X.T@(t-X@w))
  else:
    return w + eta*(X.T@(t-X@w))

tolerance = 1e-6
def converged(w_old, w_new, tolerance):
  x = np.linalg.norm(w_old-w_new)/np.linalg.norm(w_old)
  return x <= tolerance

itr = 0

eta = np.linalg.inv(X.T@X)

err_data = []
err_data.append(np.array(([w[0, 0], w[1, 0], cost(t, X, w)[0, 0]])))

w_star = np.linalg.inv(X.T@X)@(X.T)@t



while not converged(w, w_star, tolerance):
  w = steepest_decent(t, X, w, eta)
  itr += 1
  err_data.append(np.array(([w[0, 0], w[1, 0], cost(t, X, w)[0, 0]])))

err_data = np.array(err_data).T

print("Number of iteration it took to converge is ", itr)
print("Initial weights: {}".format(w_init))
print("Final weights: {}".format(w))
print("Optimal weights: {}".format(w_star))

w_1 = np.linspace(-10, 10, 100)
w_0 = np.linspace(-100, 0, 100)

XX, YY = np.meshgrid(w_1, w_0)

ZZ = np.empty_like(XX)

for i in range(XX.shape[0]):
  for j in range(XX.shape[1]):
    w_ = np.array([XX[i, j], YY[i, j]]).reshape(2, 1)
    e_ = t - X@w_
    ZZ[i, j] = 0.5*e_.T@e_

plt.figure()
plt.title("Data points")
plt.plot(X_data[:, 0], t[:, 0], ".")
plt.plot(np.linspace(-4, 4, 100), 3.879*np.linspace(-4, 4, 100) - 65, "--", label="Actual")
plt.plot(np.linspace(-4, 4, 100), w[0, 0]*np.linspace(-4, 4, 100) + w[1, 0], "--", label="
    Predicted")
plt.grid()
plt.legend()
plt.xlabel("x")
plt.ylabel("t")

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(XX, YY, ZZ, rstride=1, cstride=1, cmap='viridis', edgecolor='none')
ax.plot(err_data[0, :], err_data[1, :], err_data[2, :])
ax.scatter(w_star[0, 0], w_star[1, 0], cost(t, X, w_star)[0, 0], label=r'$w^*$')
ax.scatter(w_init[0, 0], w_init[1, 0], cost(t, X, w_init)[0, 0], label=r'$w_{init}$')
ax.set_xlabel('w_1')
ax.set_ylabel('w_0')
ax.set_zlabel('J(w)');
ax.legend()
plt.show()
```

Listing 1: Python implementation of steepest decent algorithm with optimal η

Frobenius norm

$$||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{M} A_{ij}^2} \tag{1}$$