Intro to ML Homework2

Haotian Yi N18800809

February 17, 2020

1 Question 1

(a)
For each sample, it has relationship that:

$$\hat{y}_i = w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} \cdots + w_d x_i^{(d)}$$

For all n samples:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_1^{(d)} \\ 1 & x_2^{(1)} & \cdots & x_2^{(d)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^{(1)} & \cdots & x_n^{(d)} \end{bmatrix} \quad w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = Xw$$

Thus:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\left(\sqrt{\sum_{i=1}^{n}(y_i - \hat{y}_i)}\right)^2 = \frac{1}{n}\|y - \hat{y}\|^2 = \frac{1}{n}\|y - Xw\|^2$$

Dimension of w and y are respectively $(d+1)\times 1$ and $n \times 1$.
Coordinates of w represent coefficients of linear model of y and x

(b)
To get w^*, we do $\frac{\partial MSE}{\partial w} = \frac{\partial(\frac{1}{n}\|y - Xw\|^2)}{\partial w} = 0$, because $\frac{1}{n}$ is constant, let's ignore it and thus we do $\frac{\partial MSE}{\partial w} \Rightarrow \frac{\partial SE}{\partial w} = 0$, and we express in the format of matrix:

(AB)T = BT*AT
(A-B)T = AT - BT

=(yT-wTXT)(y-Xw)

$$\begin{aligned} \frac{\partial SE}{\partial w} &= \frac{\partial(y - Xw)^2}{\partial w} = \frac{\partial[(y - Xw)^T(y - Xw)]}{\partial w} \\ &= \frac{\partial(y^T y - y^T Xw - w^T X^T y + w^T X^T Xw)}{\partial w} \\ &= \frac{\partial(y^T y)}{\partial w} - \frac{\partial(y^T Xw)}{\partial w} - \frac{\partial(w^T X^T y)}{\partial w} + \frac{\partial(w^T X^T Xw)}{\partial w} \end{aligned}$$

(1) $\dfrac{\partial(y^T y)}{\partial w} = 0$, because $y^T y$ is a scalar and y has no correlation with w

(2) $\dfrac{\partial(y^T X w)}{\partial w} = (y^T X)^T = X^T y$

(3) For $\dfrac{\partial(w^T X^T y)}{\partial w}$, because $w^T X^T y$ is a scalar so:

$$\frac{\partial(w^T X^T y)}{\partial w} = \frac{\partial(w^T X^T y)^T}{\partial w} = \frac{\partial(y^T X w)}{\partial w} = (y^T X)^T = X^T y$$

(4) For $\dfrac{\partial(w^T X^T X w)}{\partial w}$, $w^T X^T X w$ is also a scalar so:

separately get derivatives of two w

$$\begin{aligned}\frac{\partial(w^T X^T X w)}{\partial w} &= (w^T X^T X)^T + \frac{\partial(w^T X^T X w)^T}{\partial w} = X^T X w + \frac{\partial(w^T X^T X w)}{\partial w} \\ &= X^T X w + (w^T X^T X)^T = X^T X w + X^T X w \\ &= 2X^T X w\end{aligned}$$

Thus:

$$\frac{\partial SE}{\partial w} = (1) - (2) - (3) + (4) = 0 \Rightarrow 0 - X^T y - X^T y + 2X^T X w = 0$$

$$\Rightarrow X^T X w = X^T y \Rightarrow w = (X^T X)^{-1} X^T y$$

For this question I assume:

(1) There exists a linear model for X and y
(2) X and y have no correlation
(3) x_i and x_j have no correlation
(4) The inverse matrix of $X^T X$ exists

2 Question 2

Convexity is not a necessary condition.
For instance, the function below and its graph show a curve with no pure convexity, but follow the direction of negative gradient, we can still reach to the optimized point.

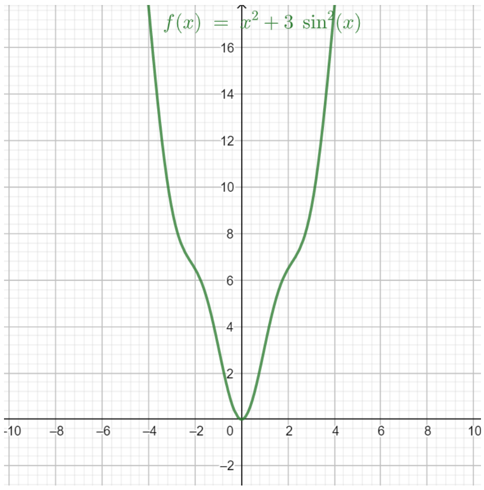


Figure 1: Graph of $f(x) = x^2 + 3sin^2x$

3 Question 3

Question 3

(a)

write a function for calculating MSE

```
[ ] def  MSE(w,X,y):
      n,d = X.shape
      Xw = X.dot(w)#predictions
      mse = (1/n) * (np.linalg.norm(y-Xw))**2
      return mse
```

Implement a function for learning the parameters of a linear model for a given tranining data with user-specified learning rate *learning_rate* and number of epochs *epochnumber*.

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

    def GD (x,y,learning_rate,epochnumber):
      n,d = x.shape
      #print("n:",n," ","d:",d)
      x0 = np.ones((n,1))
      X = np.hstack((x0,x))
      #print(X)
      #print("n:",n," ","d:",d)

      w = np.random.randn(d+1,1)
```

```
  optimalw = np.dot ( np.linalg.inv( np.dot(X.T,X) ) , (np.dot(X.T,y)) ) #use formula
  print("optimalw by formula:")
  print(optimalw)

  mse = []
  R2 = []
  for it in range(epochnumber):
    Xw = np.dot(X,w)
    w = w -(2/n)*learning_rate*( X.T.dot((Xw - y)))
    mse.append(MSE(w,X,y))
    #calculating R2 (better if close to 1)
    FVU = MSE(w,X,y) / (np.std(y)**2)
    R2.append(1-FVU)
  print("Training Result:")
  print("MSE: ",mse[-1])
  print("R^2: ",R2[-1])
  %matplotlib inline
  plt.figure(1)
  plt.plot(range(epochnumber),mse)
  plt.xlabel('epochnumber')
  plt.ylabel('MSE')
  plt.grid(True)
  plt.figure(2)
  plt.plot(range(epochnumber),R2,'r')
  plt.xlabel('epochnumber')
  plt.ylabel('R^2')
  plt.grid(True)

  return w
```

(b)

import gulcose data used in Lecture 2

```
[49] from sklearn import datasets, linear_model
     # Load the diabetes dataset
     diabetes = datasets.load_diabetes()
     x = diabetes.data
     y = diabetes.target

     print(x.shape)
     #print(y.shape)
     print(y.shape)
```

```
(442, 10)
(442,)
```

split 70-30 test-train data

```
xtest = np.split(x,[310,442],axis = 0)[0]
xtrain = np.split(x,[310,442],axis = 0)[1]
print("x shape: ",x.shape," xtest shape :",xtest.shape," xtrain shape: ",xtrain.shape)

ytest = np.split(y,[310,442])[0]
ytrain = np.split(y,[310,442])[1]
print("y shape: ",y.shape," ytest shape :",ytest.shape," ytrain shape: ",ytrain.shape)
```

```
x shape:  (442, 10)  xtest shape : (310, 10)  xtrain shape:  (132, 10)
y shape:  (442,)  ytest shape : (310,)  ytrain shape:  (132,)
```

Train linear model by sklearn

```
[51] regr = linear_model.LinearRegression()
     regr.fit(xtrain,ytrain)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

print out intercept and coefficients by sklearn

```
[52] print("intercep: ",regr.intercept_)
     print("coefficients: ",regr.coef_)
```

```
intercep:  152.19232660772457
coefficients:  [  -22.36235507  -199.24901422   466.5438688    408.97375681
 -1177.29512755   832.96506034   192.22760442   179.8446175
   871.38750025    -9.2876511 ]
```

Print out R^2 number of test data's prediction by linear model created by sklearn

```
ytest_pred = regr.predict(xtest)
RSS = np.mean((ytest_pred-ytest)**2)/(np.std(ytest)**2)
Rsq = 1-RSS
#print("RSS per sample = {0:f}".format(RSS))
print("R^2 = {0:f}".format(Rsq))
```

```
R^2 = 0.496374
```

Start to validate my own model:

Validate own function:

(1)Print out the result of using own training function: a combination of intercept and coefficient

(2)It will firstly (by function itself) print out optimal coefficients produced by using (X^TX)^(-1)X^T*y and MSE, R^2 value after using my training function

(3)At the buttom there will be two graph showing variation of MSE, R^2 with epoch

I tried set pair of learning rate and epoch to (0.2, 1000), (0.2, 4000), (0.2, 8000), (0.2, 20000),(0.2,60000), (0.9, 1000) and there R^2 value results are respectively 0.455663, 0.491797, 0.493444, 0.494453, 0.496588, 0.492356. As I observed, fix learning rate to 0.2 and lift up epoch, if epoch is larger than 4000, improvement of R^2 is inefficient.

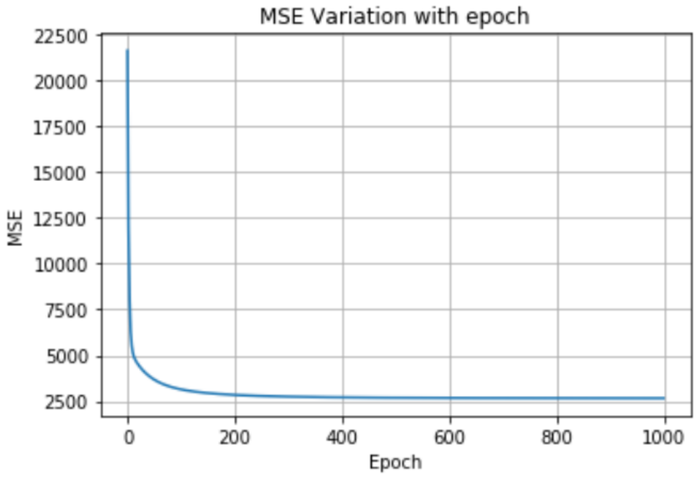
When learning rate is 0.2 and epoch is 60000, R^2 is 0.496588 which is very close to that produced by sklearn! But I think it is inefficient.

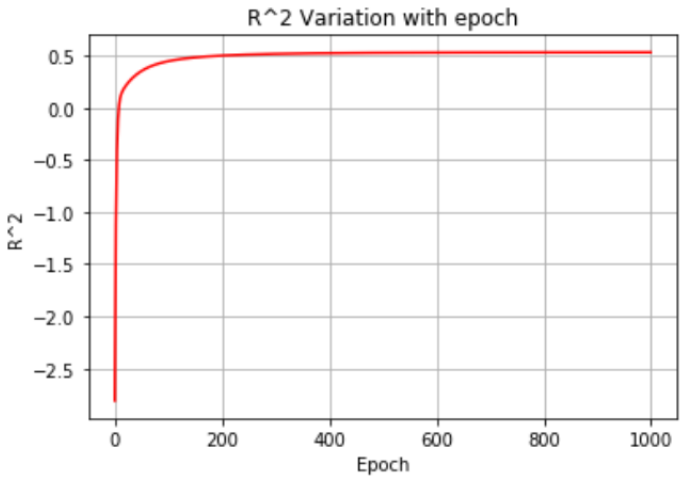
Here I display the output when I set epoch value to 1000, learning rate to 0.9, R^2 (0.492356) value we get is also similar to R^2 (0.496374) by sklearn to some extent.

```
[60] ytrain = ytrain.reshape((132,1))#reshape y to colum vector
     ytest = ytest.reshape((310,1))

     np.set_printoptions(suppress=True)
     wtrain = GD(xtrain,ytrain,0.9,1000)
     print(wtrain)
```

```
optimalw by formula:
[[  152.19232661]
 [  -22.36235507]
 [ -199.24901422]
 [  466.5438688 ]
 [  408.97375681]
 [-1177.29512755]
 [  832.96506034]
 [  192.22760442]
 [  179.8446175 ]
 [  871.38750025]
 [   -9.2876511 ]]
Training Result:
[[ 151.54699669]
 [  -1.74555074]
 [-151.39724919]
 [ 499.35835893]
 [ 385.85388172]
 [ -68.33666882]
 [ -49.92596709]
 [-235.94498653]
 [ 110.11703543]
 [ 360.87293338]
 [  25.79387104]]
```





test my model and report R^2

```
[61] Xtest = np.hstack((np.ones((310,1)),xtest))
     #print(Xtest.shape)
     MSEtest = MSE(wtrain,Xtest,ytest)
     print("MSEtest = ",MSEtest)
     FVUtest = MSEtest / (np.std(ytest)**2)
     R2test = (1-FVUtest)
     print("R^2 = {0:f}".format(R2test))
```

```
MSEtest =  3052.5605701995537
R^2 = 0.492356
```

4 Question 4

Question 4:

(a)read in data

```
[ ] import numpy as np
    import pandas as pd
    names =[
    't', # Time (secs)
    'q1', 'q2', 'q3', # Joint angle
    'dq1', 'dq2', 'dq3', # Joint velocity
    'I1', 'I2', 'I3', # Motor current (A)
    'eps21', 'eps22', 'eps31', 'eps32', # Strain measurements
    'ddq1', 'ddq2', 'ddq3' # Joint accelerations
    ]
    df = pd.read_csv('exp_train.csv', header=None,sep=',',names=names, index_col=0)
    df.head(6)
```

t	q1	q2	q3	dq1	dq2	dq3	I1	I2	I3	eps21	eps22	eps31	eps32	ddq1	ddq2	ddq3
0.00	-0.000007	2.4958	-1.1345	-7.882100e-21	-4.940656e-321	3.913100e-29	-0.081623	-0.40812	-0.30609	-269.25	-113.20	3.5918	1.57860	-9.904900e-19	-6.210306e-319	4.917400e-27
0.01	-0.000007	2.4958	-1.1345	-2.258200e-21	-4.940656e-321	2.626200e-31	-0.037411	-0.37241	-0.26698	-270.91	-116.05	1.4585	-1.73980	4.248100e-19	-1.766878e-319	-1.381100e-27
0.02	-0.000007	2.4958	-1.1345	-6.469800e-22	-4.940656e-321	1.762500e-33	-0.066319	-0.40302	-0.31459	-269.25	-112.97	3.5918	0.86753	3.233800e-19	-4.990557e-320	-4.117300e-28
0.03	-0.000007	2.4958	-1.1345	-1.853600e-22	-4.940656e-321	1.182800e-35	-0.068020	-0.43703	-0.28398	-269.97	-114.39	1.6956	-0.08059	1.500500e-19	-1.394253e-320	-1.173100e-28
0.04	-0.000007	2.4958	-1.1345	-5.310600e-23	-4.940656e-321	-5.270900e-03	-0.052715	-0.40472	-0.30779	-269.97	-114.15	3.1177	0.86753	5.932400e-20	-3.581976e-321	-3.770800e-01
0.05	-0.000007	2.4958	-1.1345	-1.521500e-23	-4.940656e-321	3.252600e-04	-0.088425	-0.42342	-0.29589	-269.25	-114.15	2.4066	-0.08059	2.164600e-20	-1.141292e-321	2.930300e-01

(b)create training data

```
[ ] df1 = df [['q2','dq2','eps21', 'eps22', 'eps31',
    'eps32','ddq2']]
    y = np.array(df['I2']).reshape(8000,1)
    X = np.array(df1)
    #print(y.shape)
    print("n: ",X.shape[0]," d: ",X.shape[1])
    #print(X)
```

```
n:  8000  d:  7
```

(c)fit a linear model by sklearn

```
[ ] from sklearn import linear_model
    regr = linear_model.LinearRegression()
    regr.fit(X,y)

    intercept = regr.intercept_
    coef = regr.coef_
    print("intercpt: ",intercept )
    print("w: ",coef)
```

```
intercpt:  [-0.08408084]
w:  [[ 0.06255018  0.20584896  0.00118784  0.00044457 -0.0031362   0.00603298
   0.05487097]]
```

Report MSE

```
[ ] y_pred = regr.predict(X)
    mse = np.mean((y_pred-y)**2)
    RSS = mse/(np.std(y)**2)
    Rsq = 1-RSS
    print("MSE of this model: ",mse)
    print("RSS per sample = {0:f}".format(RSS))
    print("R^2 = {0:f}".format(Rsq))
```

```
MSE of this model:  0.010936466882766276
RSS per sample = 0.095833
R^2 = 0.904167
```

(d)

read in test data:

```
[ ] import numpy as np
    import pandas as pd
    names =[
    't', # Time (secs)
    'q1', 'q2', 'q3', # Joint angle
    'dq1', 'dq2', 'dq3', # Joint velocity
    'I1', 'I2', 'I3', # Motor current (A)
    'eps21', 'eps22', 'eps31', 'eps32', # Strain measurements
    'ddq1', 'ddq2', 'ddq3' # Joint accelerations
    ]
    df = pd.read_csv('exp_test.csv', header=None,sep=',',names=names, index_col=0)
    df.head(6)
```

t	q1	q2	q3	dq1	dq2	dq3	I1	I2	I3	eps21	eps22	eps31	eps32	ddq1	ddq2	ddq3
0.00	-0.000007	1.9024	0.26063	-0.000364	4.940656e-321	0.012596	-0.096928	-0.15134	-0.017005	-130.83	-41.856	-6.3635	5.13410	-0.045712	6.210306e-319	1.582900
0.01	0.000013	1.9024	0.26073	0.000739	4.940656e-321	0.012095	-0.028908	-0.11903	-0.020406	-138.18	-51.100	-14.6590	-5.05820	0.125580	1.766878e-319	0.414660
0.02	-0.000007	1.9024	0.26086	-0.000580	4.940656e-321	0.011596	-0.059517	-0.13944	-0.047614	-139.36	-51.812	-14.6590	-5.29520	-0.130080	4.990557e-320	0.082286
0.03	0.000013	1.9024	0.26099	0.001409	4.940656e-321	0.013933	-0.079923	-0.15304	-0.023807	-135.57	-48.019	-11.3410	-0.79168	0.213010	1.394253e-320	0.190650
0.04	-0.000007	1.9024	0.26110	-0.001273	4.940656e-321	0.010793	-0.025507	-0.12924	-0.006802	-135.81	-49.204	-12.0520	-2.21390	-0.276490	3.581976e-321	-0.170400
0.05	-0.000007	1.9024	0.26124	0.001928	4.940656e-321	0.011915	-0.083324	-0.14964	-0.034010	-139.60	-53.471	-16.0820	-6.95450	0.323560	1.141292e-321	0.031745

Create test data

```
[ ] df1 = df [['q2','dq2','eps21', 'eps22', 'eps31',
    'eps32','ddq2']]#.dropna()

    y = np.array(df['I2']).reshape(8000,1)
    x = np.array(df1)
    #print(y.shape)
    print("n: ",x.shape[0]," d: ",x.shape[1])
    #print(x)
```

```
n:  8000  d:  7
```

use linear model built above and calculating MSE

```
[ ] y_pred = regr.predict(x)
    mse = np.mean((y_pred-y)**2)
    print("MSE: ",mse)
```

```
MSE:  0.009723098281465446
```