1. Load data

- data_train.csv, data_test.csv 파일을 동일 디렉터리에 위치시키고 데이터를 읽었습니다.
  읽는 과정에서 theta0의 계수를 위해 1을 추가하면서 읽었습니다.
- 읽은 데이터들을 (x, y, z)와 (h)로 split하여 matrix를 만들었습니다.

```
import csv
import numpy as np
import matplotlib.pyplot as plt
'''
1. Data
 - Config
 - Load Data
 - Split loaded data
'''

# Config
learning_rate=0.00002
opt_threshold=1e-5
t0, t1, t2, t3=-1., 1., 3., 5.
theta=np.array([[t0, t1, t2, t3]])

# Load data
path = "drive/My Drive/Classroom/Machine Learning (2) 2020-1\
/class-MachineLearning/assignment04/"
train=[]
test=[]

with open(path+'data_train.csv', newline='') as myfile:
    reader  = csv.reader(myfile, delimiter=',')
    ct = 1
    for i in reader:
        temp=[1.0, float(i[0]), float(i[1]), float(i[2]), float(i[3])]
        train.append(temp)
        ct += 1
    train_m=ct

with open(path+'data_test.csv', newline='') as myfile:
    reader  = csv.reader(myfile, delimiter=',')
    ct = 1
    for i in reader:
        temp=[1.0, float(i[0]), float(i[1]), float(i[2]), float(i[3])]
        test.append(temp)
        ct += 1
    test_m=ct

# Split loaded data
train=np.array(train)
test=np.array(test)
```

```
test=np.array(test)
train_x, train_y=train[:, :4], train[:, 4]
train_y=train_y[:,np.newaxis]
test_x, test_y=test[:,:4], test[:,4]
test_y=test_y[:,np.newaxis]
```

2. Optimize linear function

- 각각 train, test data에 대한 hypothesis, objective function, gradient descent를 정의했습니다.
- 연산과정들을 matrix로 빠르게 계산하기 위해 np.dot함수를 사용했습니다.
- 각 theta(feature)들에 맞는 update함수를 정의함으로써 학습동안 각각의 theta들이 모두 update되도록
- 수렴하는 기준을 1e-5로 설정하고 iteration에 대한 loss변화량이 이 기준값보다 작으면 수렴했다 판단했

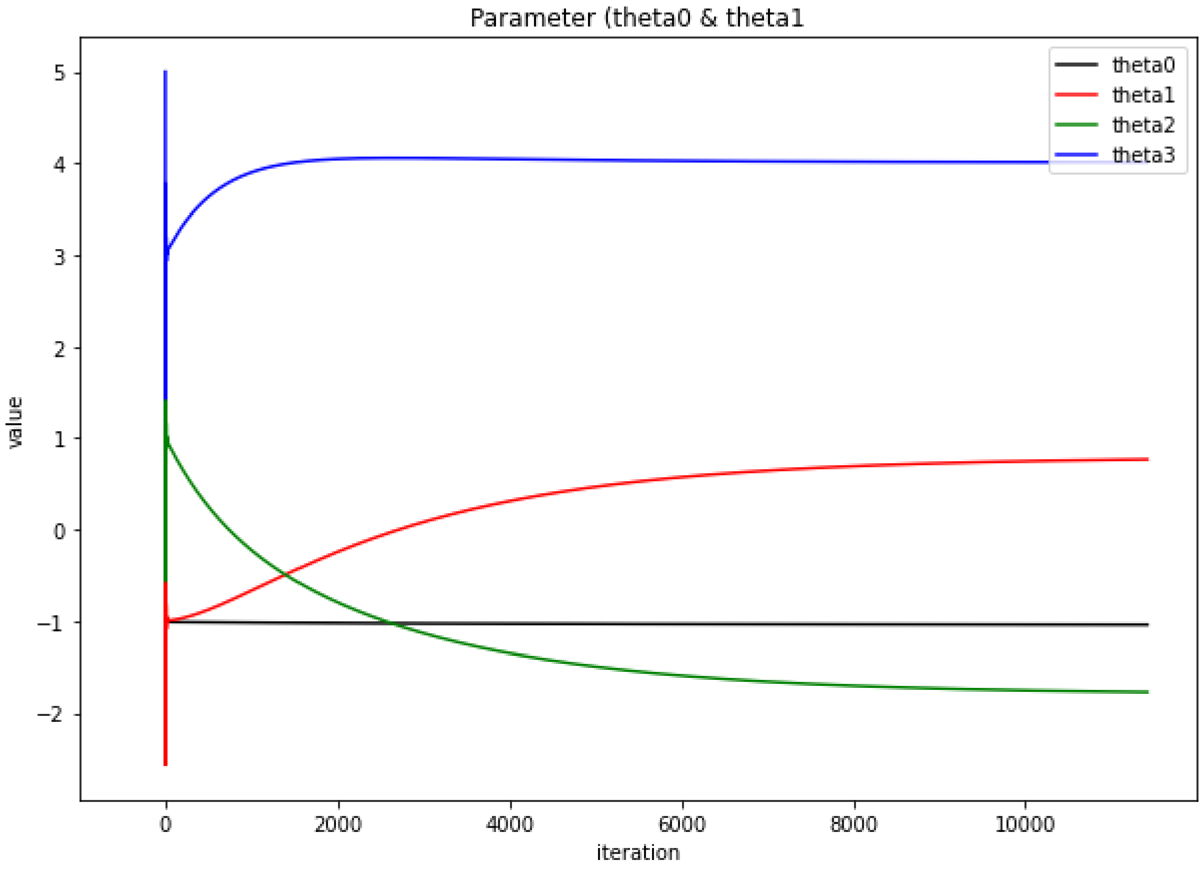
```
'''
2. Hypothesis
 - Define hypothesis for each train and test

3. Objective function
 - Define objective function for each train and test

4. Gradient Descent
 - Define derivations
 - Update theta0, theta1, theta2, theta3
'''

# Define hypothesis
def train_hypothesis() :
    return train_x.dot(theta.T)

def test_hypothesis() :
    return test_x.dot(theta.T)

train_h=train_hypothesis()
test_h=test_hypothesis()

# Define objective function
def train_objective_function() :
    train_J = np.sum((train_h-train_y)**2) / (2*train_m)
    return train_J

def test_objective_function() :
    test_J = np.sum((test_h-test_y)**2) / (2*test_m)
    return test_J

# Define derivations
def dev() :
    dev0=np.sum(train_h-train_y) / train_m
    dev1=np.sum(np.squeeze(train_h-train_y)*(train_x[:, 1])) / train_m
    dev2=np.sum(np.squeeze(train_h-train_y)*(train_x[:,2])) / train_m
    dev3=np.sum(np.squeeze(train_h-train_y)*(train_x[:,3])) / train_m

    dev=[dev0, dev1, dev2, dev3]
    return dev

# Update theta0, theta1, theta2, theta3
def gradient_descent() :
    return (theta[0][0]-(learning_rate*dev0), theta[0][1]-(learning_rate*dev1),\
            theta[0][2]-(learning_rate*dev2), theta[0][3]-(learning_rate*dev3))

train_J_list=[]
test_J_list=[]
theta0_list=[]
theta1_list=[]
theta2_list=[]
theta3_list=[]

# Train
count=0
while(1) :
    count+=1
    train_J=train_objective_function()
    test_J=test_objective_function()

    train_J_list.append(train_J)
    test_J_list.append(test_J)
    theta0_list.append(theta[0][0])
    theta1_list.append(theta[0][1])
    theta2_list.append(theta[0][2])
    theta3_list.append(theta[0][3])

    dev0, dev1, dev2, dev3 = dev()
    temp0, temp1, temp2, temp3 = theta[0]
    theta[0][0], theta[0][1], theta[0][2], theta[0][3] = gradient_descent()

    train_h=train_hypothesis()
    test_h=test_hypothesis()

    if len(train_J_list)>=2 and \
    abs(train_J_list[-2] - train_J_list[-1]) <opt_threshold :
        break

print("theta0 : ", theta[0][0])
print("theta1 : ", theta[0][1])
print("theta2 : ", theta[0][2])
print("theta3 : ", theta[0][3])
print("loss : ", train_J)
```

```
theta0 :  -1.0397169098603019
theta1 :  0.7669794916708456
theta2 :  -1.7708953304884383
theta3 :  4.009639516613057
loss :  103.12575206537106
```

3. Model parameters

- update를 할때마다 변하는 theta0, theta1, theta2, theta3의 값들을 따로 리스트에 저장해뒀었습니다.
- 각 iteration마다 위 theta들이 각각 어떻게 변하는지 시각화했습니다.

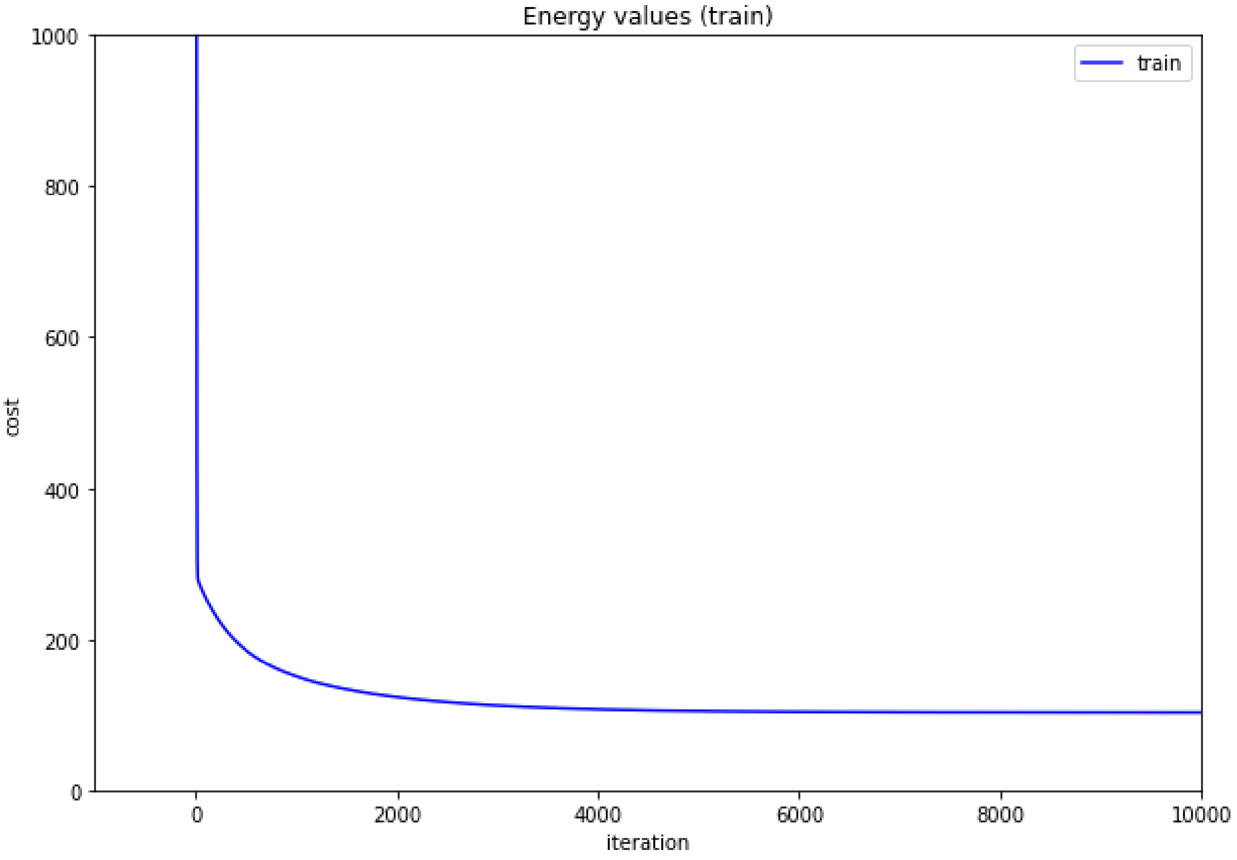
```
'''
Visualize
'''
plt.figure(figsize=(10, 7))
plt.plot(theta0_list, color='black', label='theta0')
plt.plot(theta1_list, color='red', label='theta1')
plt.plot(theta2_list, color='green', label='theta2')
plt.plot(theta3_list, color='blue', label='theta3')
plt.legend(loc='upper right')
plt.xlabel("iteration")
plt.xlim(left=-1000)
plt.ylabel("value")
plt.title("Parameter (theta0 & theta1")
plt.show()
```



4. Energy Values (training data)

- training data에 대해서 update를 할때마다 train_J_list에 그동안의 loss 값들을 넣었습니다.
- 각 iteration마다 loss값이 어떻게 변하는지 시각화했습니다.

```
'''
Visualize training error at every iteration
'''
plt.figure(figsize=(10, 7))
plt.plot(train_J_list, color='blue', label='train')
plt.xlabel("iteration")
plt.ylim(top=1000, bottom=0)
plt.xlim(right=10000, left=-1000)
plt.ylabel("cost")
plt.legend(loc='upper right')
plt.title("Energy values (train)")
plt.show()
```

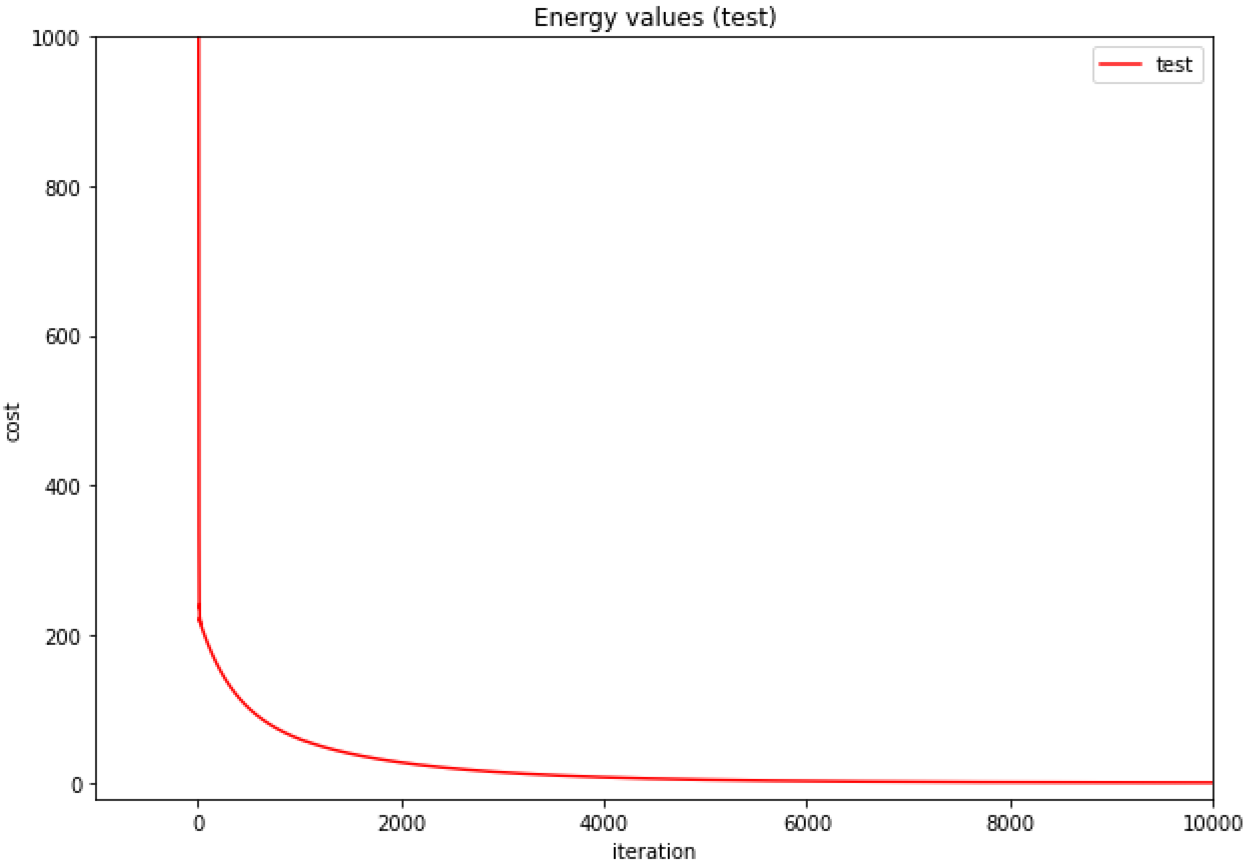


5. Energy Values (test data)

- training data를 이용해 update를 할때마다 test_J_list에 test data에 대한 loss 값들을 넣었습니다.
- 각 iteration마다 loss값이 어떻게 변하는지 시각화했습니다.

```
'''
Visualize training error at every iteration
```

```
'''
plt.figure(figsize=(10, 7))
plt.plot(test_J_list, color='red', label='test')
plt.xlabel("iteration")
plt.ylim(top=1000, bottom=-20)
plt.xlim(right=10000, left=-1000)
plt.ylabel("cost")
plt.legend(loc='upper right')
plt.title("Energy values (test)")
plt.show()
```


Energy values (test)
test
cost
iteration