코드 세미나 발표

< 6조 >

팀 모델 1개 코드

개별 데이터 실험 결과

강재훈, 김민선, 송영훈

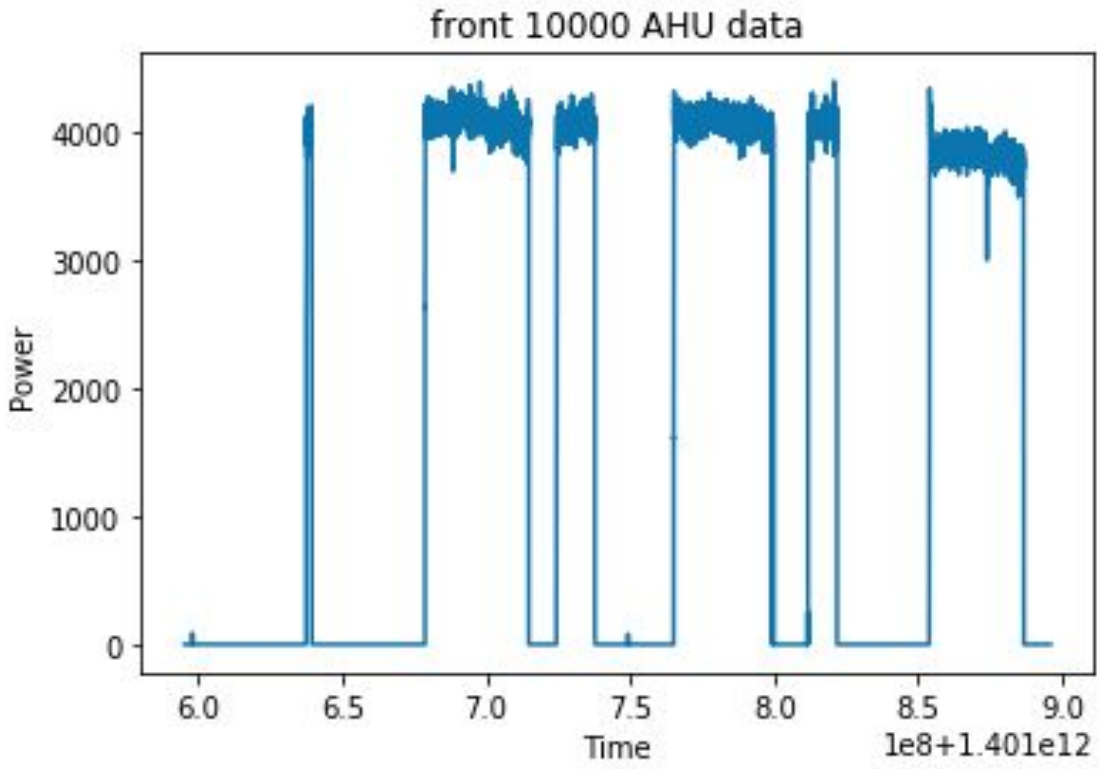
Floor 2

대표 모델 (기본문제) - 사용함수

```
def plotting(x,y,title):
    plt.plot(x,y)
    plt.title(title)
    plt.show()

def plotting_time_power(data, title):
    x_data = data[:,[0]]
    y_data = data[:,[1]]
    plt.plot(x_data, y_data)
    plt.title(title)
    plt.xlabel("Time")
    plt.ylabel("Power")
    plt.show()
    print("total data size : ", len(data))

def plotting_xy(data, title):
    x_data = data[:,[1]]
    y_data = data[:,[2]]
    plt.plot(x_data, y_data,'bo')
    plt.title(title)
    plt.xlabel("t-power")
    plt.ylabel("t+1-power")
    plt.show()
    print("total data size : ",len(data))
```

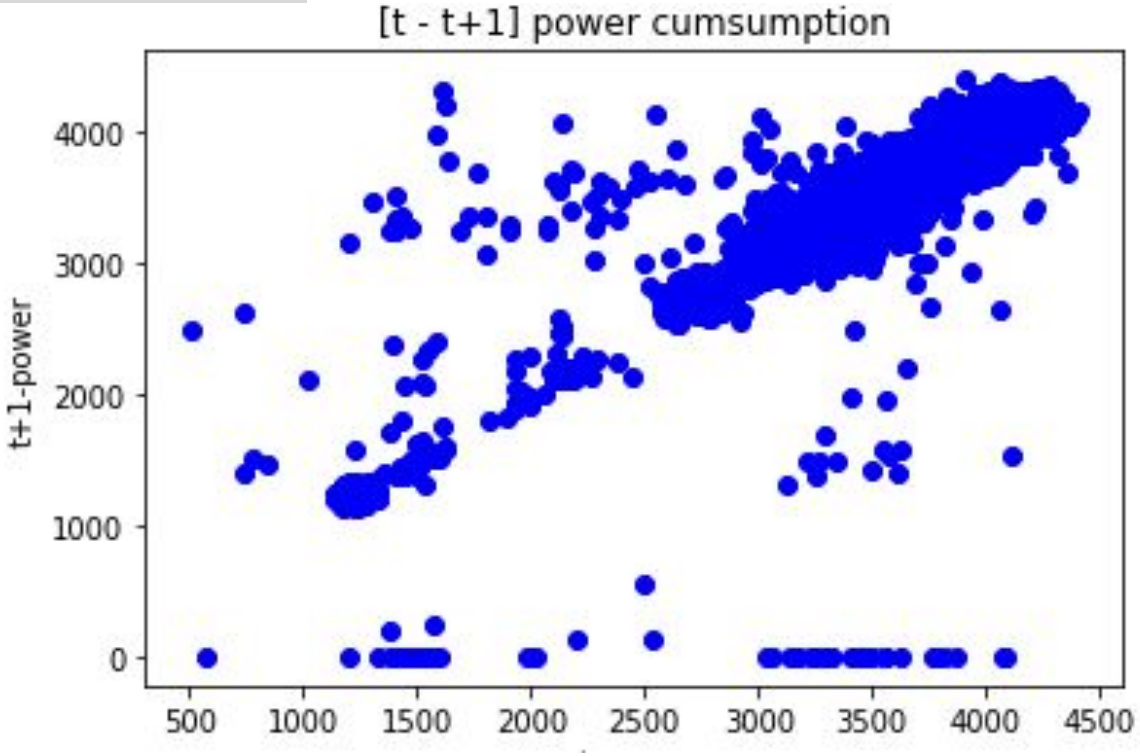
대표 모델 (기본문제) - 사용함수

```
def plotting_time_power(data, title):
    x_data = data[:,[0]]
    y_data = data[:,[1]]
    plt.plot(x_data, y_data)
    plt.title(title)
    plt.xlabel("Time")
    plt.ylabel("Power")
    plt.show()
    print("total data size : ",
```



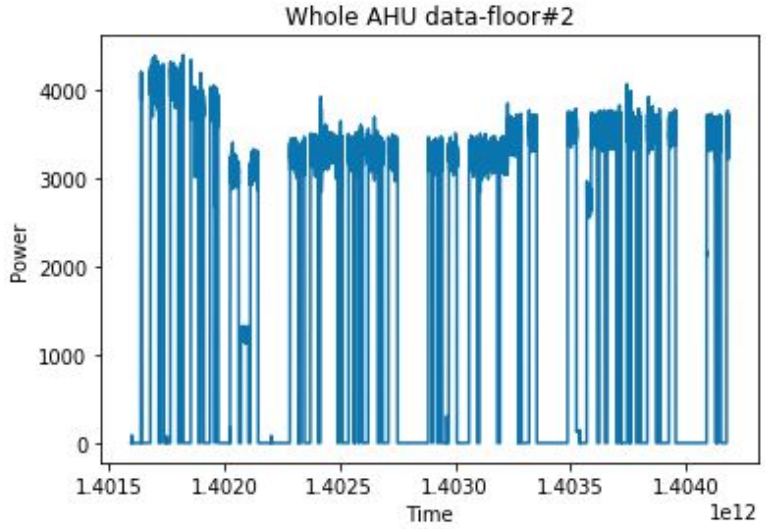
대표 모델 (기본문제) - 사용함수

```
def plotting_xy(data, title):
    x_data = data[:,[1]]
    y_data = data[:,[2]]
    plt.plot(x_data, y_data,'bo')
    plt.title(title)
    plt.xlabel("t-power")
    plt.ylabel("t+1-power")
    plt.show()
    print("total data size :
```

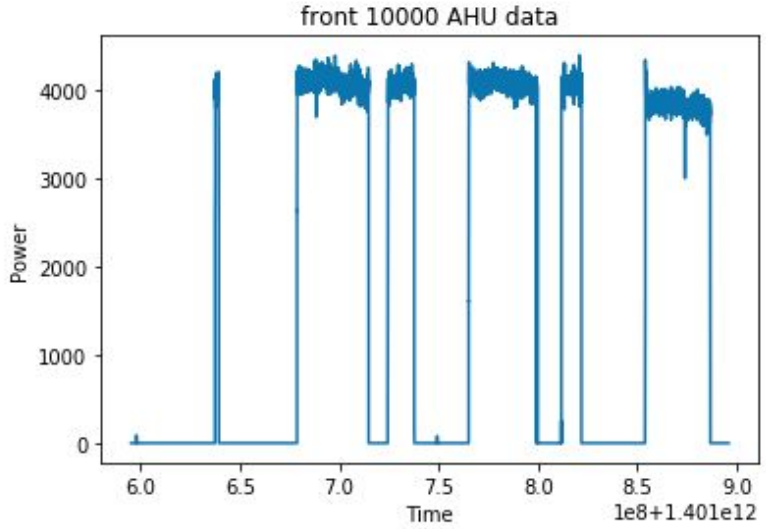


```
data = np.loadtxt('Power.csv', delimiter=',', dtype=np.float32)
timeArr = np.array([],dtype=np.string_) #later
#print(type(timestamp2time(data[0,0])))
fist_sample_time = int(data[0,0])/1000
print("first sampling time : ",timestamp2time(fist_sample_time))
plotting_time_power(data,"Whole AHU data-floor#2")
plotting_time_power(data[0:10000,:],"front 10000 AHU data")
```

```
first sampling time :  2014-06-01 04:00:32
```




```
total data size :  86198
```



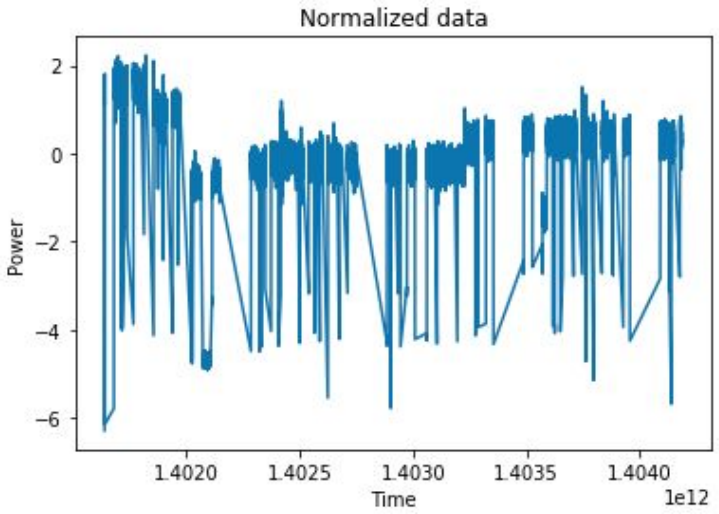

```
total data size :  10000
```

```
# manipulate data (make 'y' data)
new_data = data[0:-1,]
print(new_data.shape)
y_data = data[1:,[1]]
print(y_data.shape)
new_data = np.append(new_data,y_data,axis=1)
print(new_data.shape)
```

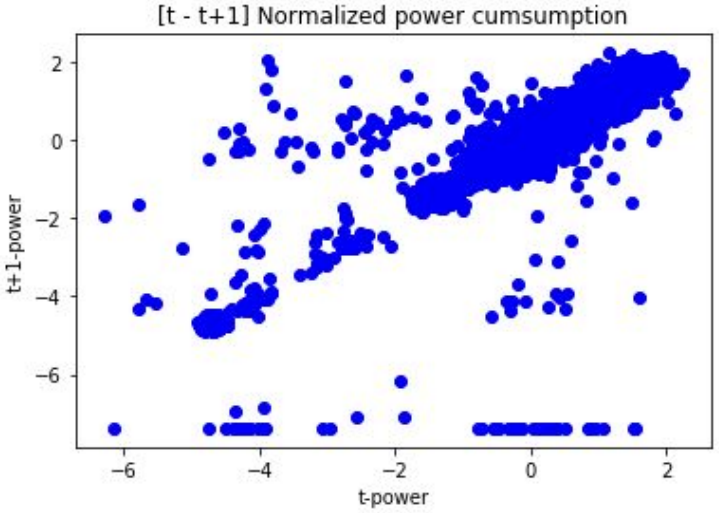
```
(86197, 2)
(86197, 1)
(86197, 3)
```

```
# Normalization
m = np.mean(new_data[:,1])
r = np.std(new_data[:,1])
print("mean: ",m, " stdev: ",r)
for i in range(0,len(new_data)):
    new_data[i,1] = (new_data[i,1]-m)/r
    new_data[i,2] = (new_data[i,2]-m)/r
plotting_time_power(new_data, "Normalized data")
plotting_xy(new_data,"[t - t+1] Normalized power cumsumption")
```

```
mean:  3384.4165  stdev:  458.72726
```




```
total data size :  41880
```




```
total data size :  41880
```

```
# Decompsition data into training
length = len(new_data)
train_length = int(length*0.8)
print("total data size : ", length)
print("train data size : ",train_length)
print("test  data size : ",length-train_length)

ts_train = new_data[0:train_length,[0]]
x_train = new_data[0:train_length,[1]]
y_train = new_data[0:train_length,[2]]


ts_test = new_data[train_length:-1,[0]]
x_test = new_data[train_length:-1,[1]]
y_test = new_data[train_length:-1,[2]]
```

```
total data size :  41880
train data size :  33504
test  data size :  8376
```

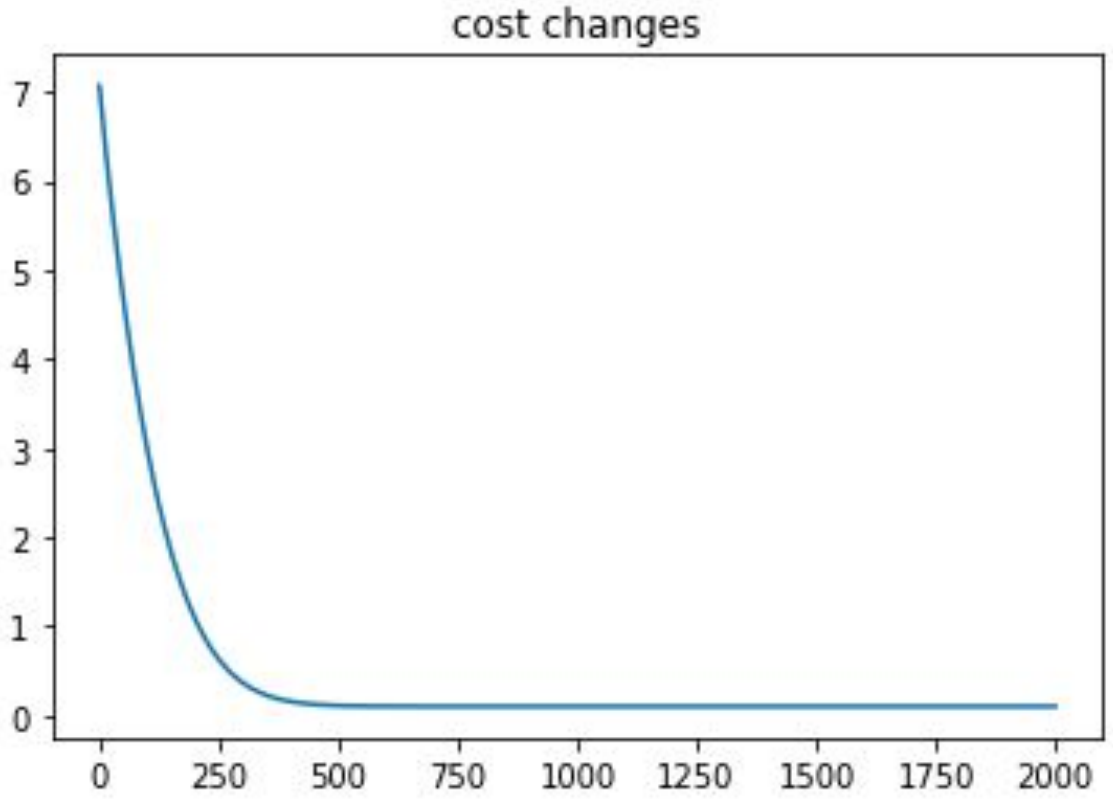
```
X = tf.placeholder(tf.float32, shape=[None, 1])
Y = tf.placeholder(tf.float32, shape=[None, 1])
```

```
W = tf.Variable(tf.random_normal([1, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

model = tf.matmul(X, W) + b
cost = tf.reduce_mean(tf.square(model - Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learn_rate) # 1e-5 = 1 x 10^-5
train = optimizer.minimize(cost)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
costList = np.array([])
epochList = np.arange(training_epoches)
for step in range(training_epoches):
    cost_val, hy_val, _ = sess.run([cost, model, train], feed_dict={X: x_train, Y: y_train})
    costList = np.append(costList,cost_val)
    if step % 200 == 0:
        print(step, ",  Cost: ", cost_val) # show front 3 estimated output
```

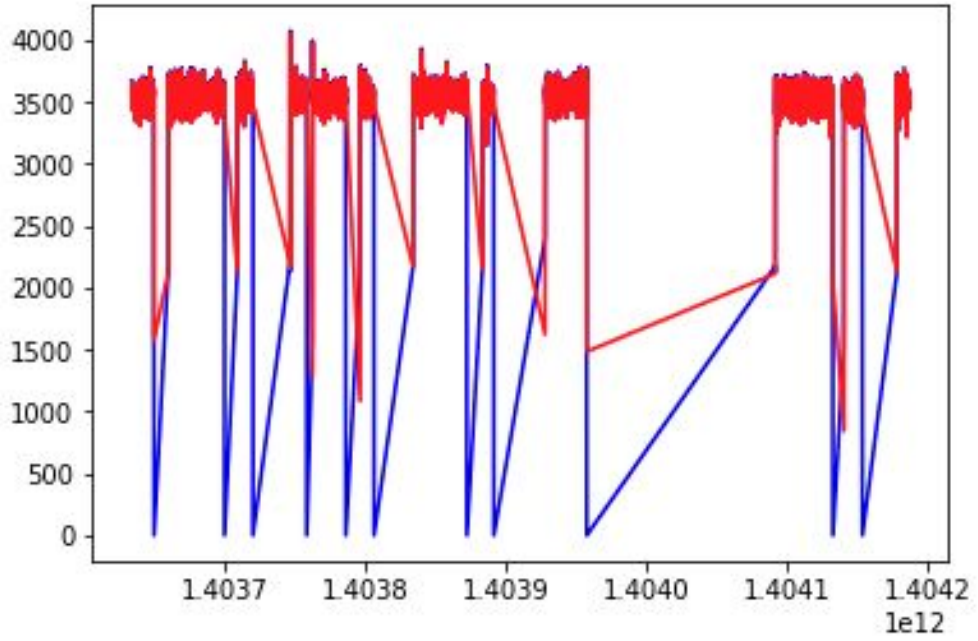
```
0 ,  Cost:  7.072536
200 ,  Cost:  1.0873991
400 ,  Cost:  0.15534899
600 ,  Cost:  0.10141915
800 ,  Cost:  0.10039633
1000 ,  Cost:  0.10039022
1200 ,  Cost:  0.10039021
1400 ,  Cost:  0.10039022
1600 ,  Cost:  0.10039022
1800 ,  Cost:  0.10039021
2000 ,  Cost:  0.10039021
```

```
# cost change
plotting(epochList,costList,"cost changes")
```



```
print("mean : ", m, "  stdev : ",r)
predict = m+r*sess.run(model, feed_dict={X:x_test })
real = m+r*y_test
mse = np.sqrt(np.mean((predict-real)**2))
print("MSE ", mse)



plt.plot(ts_test,real,'b')
plt.plot(ts_test,predict,'r')
plt.show()
```

```
mean :  3384.4165   stdev :  458.72726
MSE  163.26917
```



Floor 0

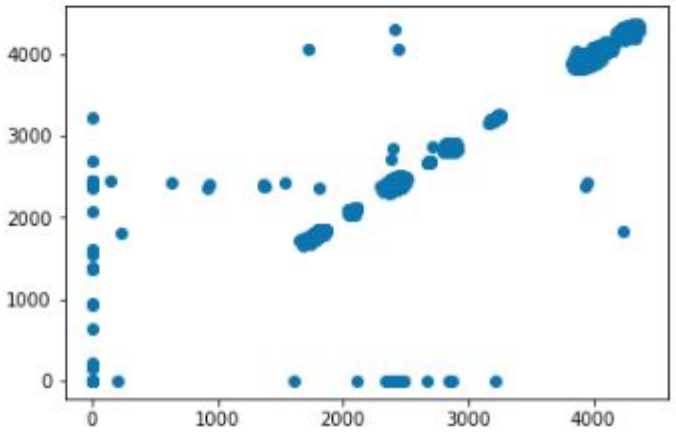
In [1]:
```
import tensorflow as tf
import pandas as pd
import numpy as np

import scipy as sp
import scipy.stats
import matplotlib.pyplot as plt
import math
from sklearn.linear_model import LinearRegression
```

In [2]:
```
data=np.loadtxt(fname='Power.csv', delimiter=',', dtype=np.float32)
data
```

Out[2]:
```
array([[  1.40159523e+12,   1.20000043e-17],
       [  1.40159523e+12,   1.20000043e-17],
       [  1.40159523e+12,   1.20000043e-17],
       ...,
       [  1.40418718e+12,   2.45426489e+03],
       [  1.40418718e+12,   2.42050635e+03],
       [  1.40418718e+12,   2.41890601e+03]], dtype=float32)
```

In [3]:
```
'''raw data'''

x_data=data[:-1,1]
y_data=data[1: ,1]

print(x_data.shape, y_data.shape)

plt.scatter(x_data, y_data)
plt.show()
```

```
(86191,) (86191,)
```



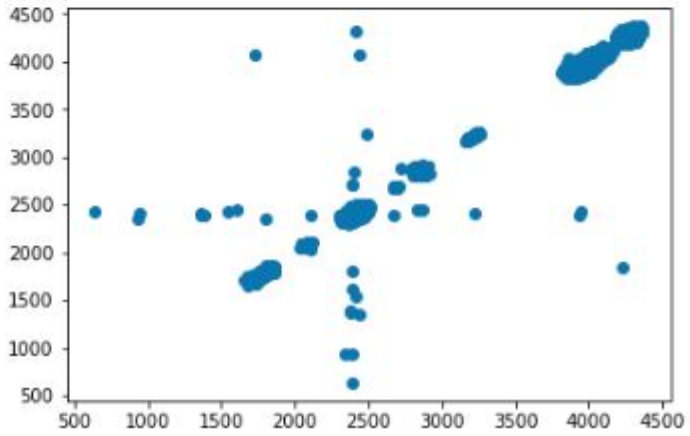
In [4]:
```
'''
1. 기본 문제
'''
```

Out[4]:
```
'\n1. 기본 문제\n'
```

In [5]:
```
data2=data[data[:,1]>500]
x_data2 = data2[:-1,1]
y_data2 = data2[1: ,1]

print(x_data2.shape, y_data2.shape)
```

```
(38078,) (38078,)
```

In [6]:
```
#sum(x_data<500) #off 상태의 data 수
#len(y_data[x_data>500]) # on 상태의 data

#x_data2 = x_data[x_data>500]
#y_data2 = y_data[x_data>500]

#y_data2 = y_data2[y_data2>500]
#x_data2 = x_data2[y_data2>500]

plt.scatter(x_data2, y_data2)
plt.show()
```

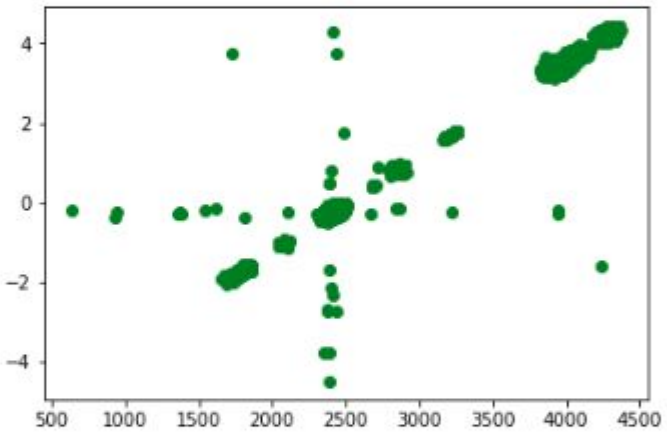


In [7]:
```
y_data2_mean = np.mean(y_data2)
y_data2_std = np.std(y_data2)

print(y_data2_mean, y_data2_std)
```

```
2508.33 417.997
```

In [8]:
```
y_data2_norm = (y_data2-y_data2_mean)/y_data2_std
print(y_data2_norm)

plt.scatter(x_data2, y_data2_norm, c='g')
plt.show()
```

```
[-1.06159723 -0.99129468 -1.00637484 ..., -0.12933181 -0.21009447
 -0.21392307]
```



In [9]:
```
data3 = list()

for i in range(len(x_data2)):
    data3.append([x_data2[i], y_data2_norm[i]])

data3= np.array(data3)
data3
```

Out[9]:
```
array([[  2.08338794e+03,  -1.06159723e+00],
       [  2.06458081e+03,  -9.91294682e-01],
       [  2.09396704e+03,  -1.00637484e+00],
       ...,
       [  2.42636914e+03,  -1.29331812e-01],
       [  2.45426489e+03,  -2.10094467e-01],
       [  2.42050635e+03,  -2.13923067e-01]], dtype=float32)
```

In [10]:
```
#number of training data
num_training=int(np.round(len(data3)*0.8))
#number of testing data
num_test =len(data3)-num_training

print(len(data3), num_training, num_test)
```

```
38078 30462 7616
```

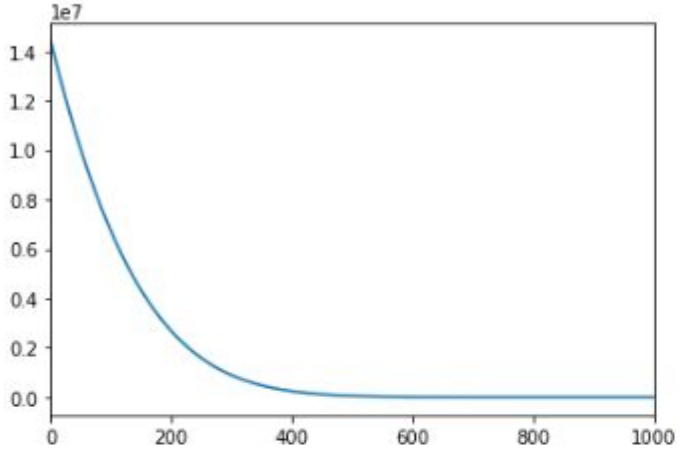
In [11]:
```
training_data = data3[:num_training, : ]
test_data = data3[num_training:, : ]

xtrain = training_data[:, [0]]
ytrain = training_data[:, [1]]
xtest = test_data[:, [0]]
ytest = test_data[:, [1]]
```

In [12]:
```
X= tf.placeholder(tf.float32, shape=[None, 1])
Y= tf.placeholder(tf.float32, shape=[None, 1])

W= tf.Variable(tf.random_normal([1, 1]), name='weight')
b= tf.Variable(tf.random_normal([1]), name='bias')

model= tf.add(tf.matmul(X, W), b) #linear regression

cost = tf.reduce_mean(tf.square(model - Y))

train=tf.train.AdamOptimizer(learning_rate=0.005).minimize(cost)
```

In [13]:
```
sess= tf.Session()
sess.run(tf.global_variables_initializer())

trial_num=list()
cost_list=list()

for step in range(20001):
    cost_val, _=sess.run([cost, train], feed_dict={X:xtrain, Y:ytrain})
    cost_list.append(cost_val)
    trial_num.append(step)

    if step % 1000 == 0:
        print(step, cost_val)
```

```
0 1.43977e+07
1000 1.73279
2000 0.910745
3000 0.910032
4000 0.908827
5000 0.906822
6000 0.903515
7000 0.89808
8000 0.889185
9000 0.874709
10000 0.851354
11000 0.814214
12000 0.756537
13000 0.670408
14000 0.549829
15000 0.397923
16000 0.236405
17000 0.110196
18000 0.0521593
19000 0.0280297
20000 0.0182221
```

In [22]:
```
# cost function

plt.plot(trial_num, cost_list)
plt.xlim(0, 1000)
#plt.ylim(0, 1)
plt.show()
```



In [15]:
```
# prediction for 7616 test data

result=sess.run(model, feed_dict={X: xtest})
result_=(result*y_data2_std)+y_data2_mean

print(result_)
```

```
[[ 2400.33203125]
 [ 2438.0390625 ]
 [ 2425.10449219]
 ...,
 [ 2440.37060547]
 [ 2466.08959961]
 [ 2434.96508789]]
```

In [16]:
```
# Accuracy measurement
ytest_hat=list()

for i in range(len(result)):
    ytest_hat.append(result[i,][0])
```

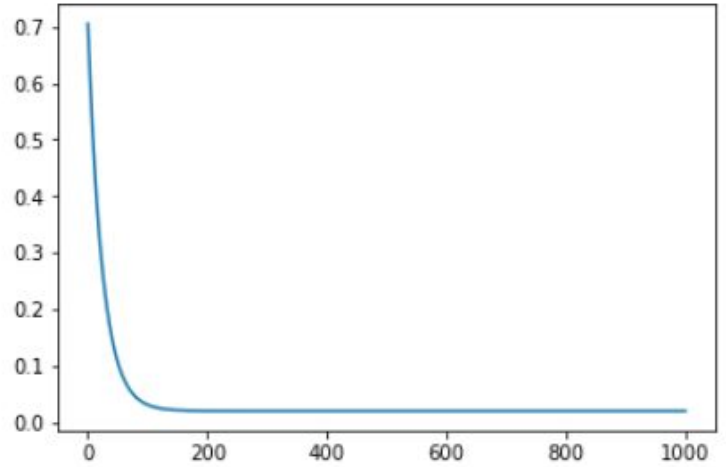
In [17]:
```
rmse_test1 = np.sqrt( ((ytest-ytest_hat)**2).mean() )
rmse_test1
```

Out[17]: 0.54796588

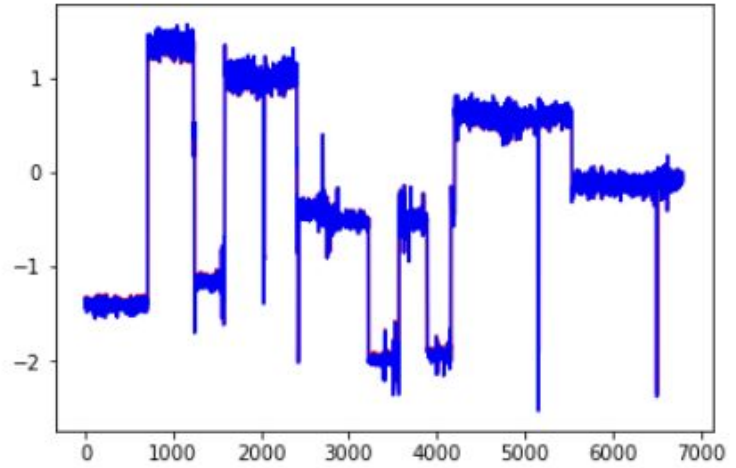
FNN 실습 결과 - 1st floor

- Learning Rate = 0.01
- Epoch = 1000

```
0       cost: 0.705226838589
100     cost: 0.0307684969157
200     cost: 0.0201648958027
300     cost: 0.0199965629727
400     cost: 0.0199938584119
500     cost: 0.0199938137084
600     cost: 0.0199938118458
700     cost: 0.0199938137084
800     cost: 0.0199938155711
900     cost: 0.0199938118458
```

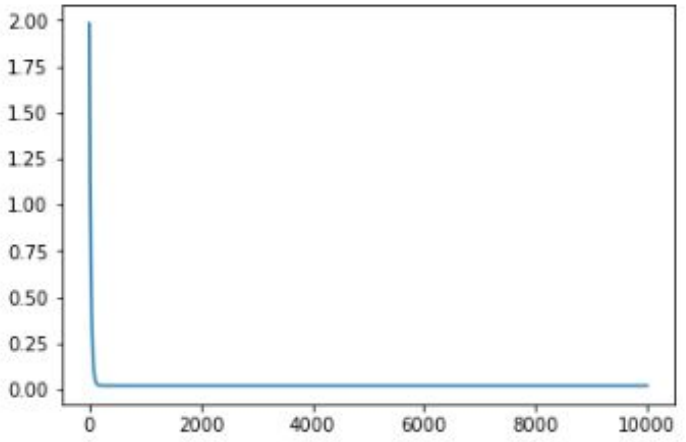



```
accuracy: 0.998269954231
```

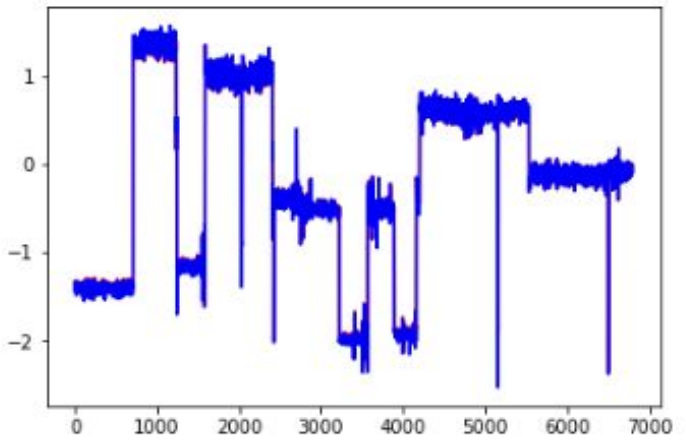


- Learning Rate = 0.01
- Epoch = 10000

```
0       cost: 1.98068869114
1000    cost: 0.0199938137084
2000    cost: 0.0199938137084
3000    cost: 0.0199938118458
4000    cost: 0.0199938118458
5000    cost: 0.0199938118458
6000    cost: 0.0199938118458
7000    cost: 0.0199938137084
8000    cost: 0.0199938137084
9000    cost: 0.0199938137084
```

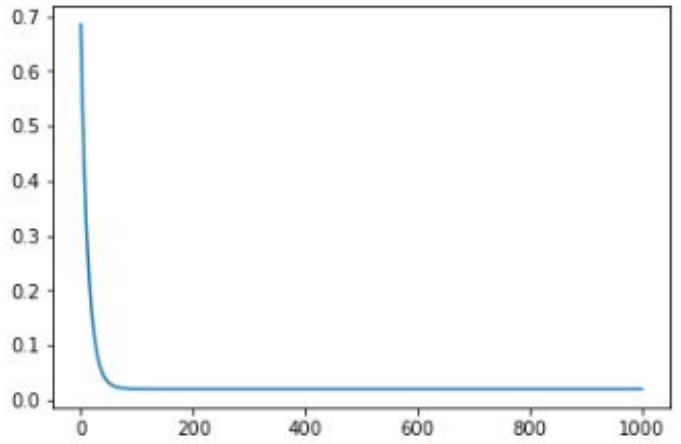



```
accuracy: 0.998269865755
```

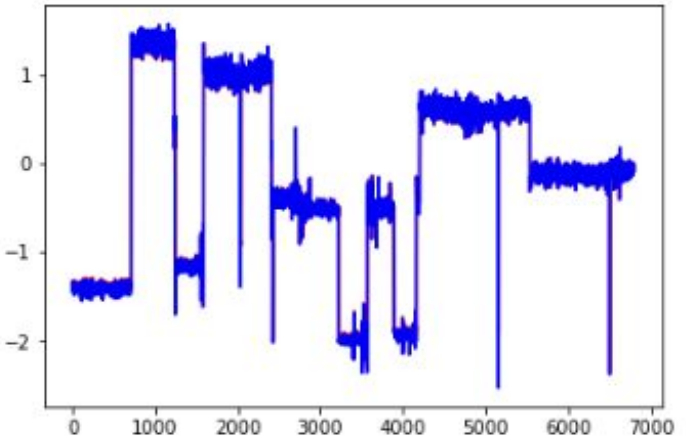


- Learning Rate = 0.02
- Epoch = 1000

```
0       cost: 0.6844009161
100     cost: 0.0201899353415
200     cost: 0.0199938770384
300     cost: 0.0199938118458
400     cost: 0.0199938118458
500     cost: 0.0199938118458
600     cost: 0.0199938118458
700     cost: 0.0199938137084
800     cost: 0.0199938155711
900     cost: 0.0199938155711
```

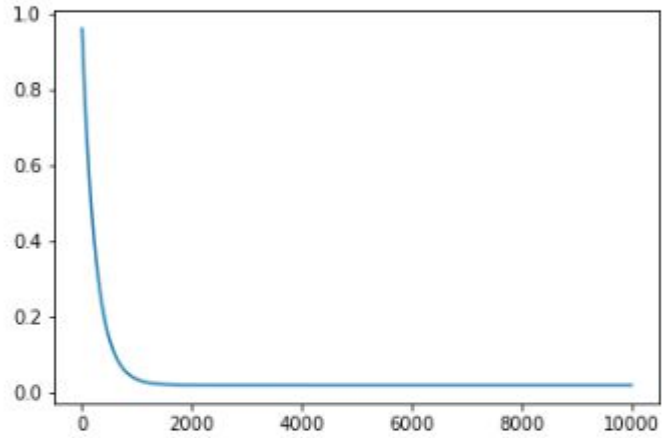



```
accuracy: 0.998270268436
```

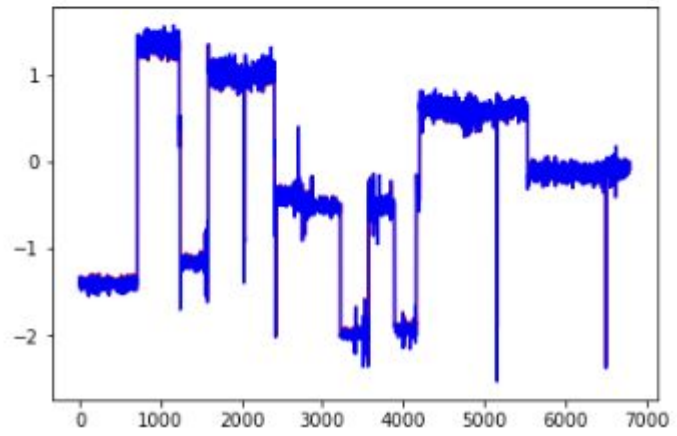


- Learning Rate = 0.02
- Epoch = 10000

```
0       cost: 0.960259318352
1000    cost: 0.0355213992298
2000    cost: 0.0202533118427
3000    cost: 0.0199982151389
4000    cost: 0.0199938900769
5000    cost: 0.0199938155711
6000    cost: 0.0199938137084
7000    cost: 0.0199938137084
8000    cost: 0.0199938137084
9000    cost: 0.0199938137084
```

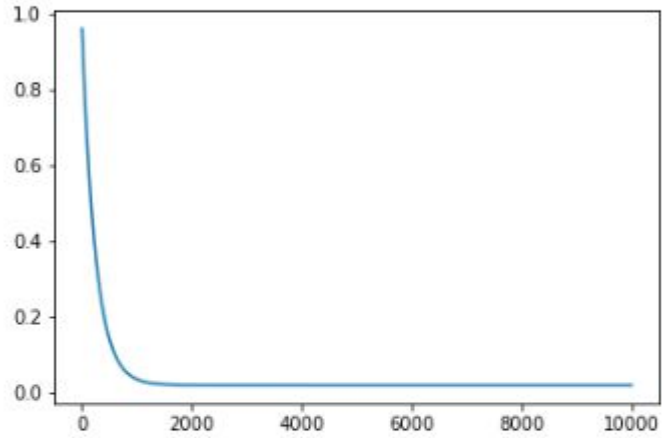



```
accuracy: 0.998267359333
```

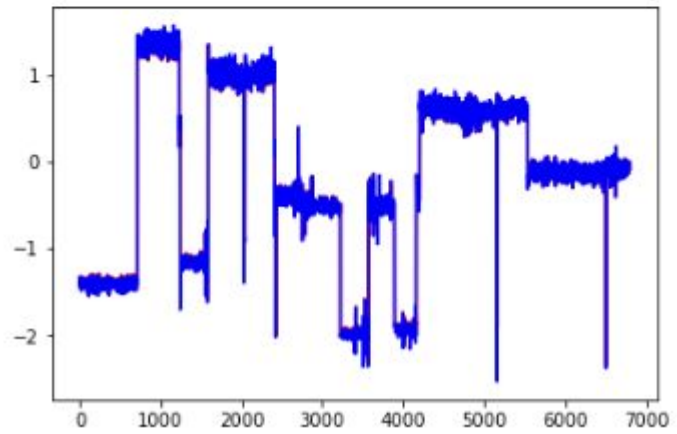


- Learning Rate = 0.001
- Epoch = 10000

```
0       cost: 0.960259318352
1000    cost: 0.0355213992298
2000    cost: 0.0202533118427
3000    cost: 0.0199982151389
4000    cost: 0.0199938900769
5000    cost: 0.0199938155711
6000    cost: 0.0199938137084
7000    cost: 0.0199938137084
8000    cost: 0.0199938137084
9000    cost: 0.0199938137084
```

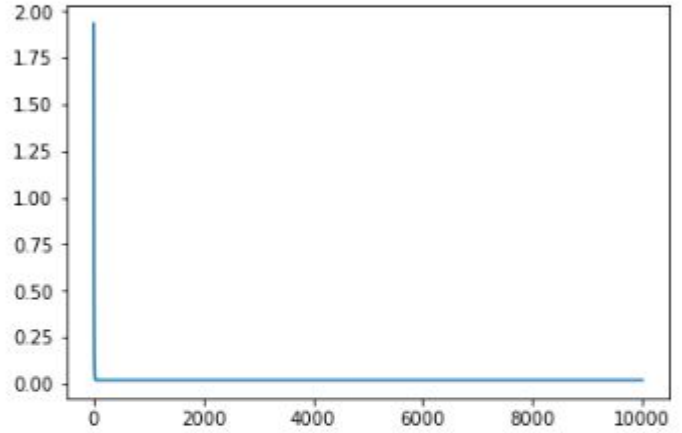



```
accuracy: 0.998267359333
```



- Learning Rate = 0.05
- Epoch = 10000

```
0       cost: 1.93209183216
1000    cost: 0.0199938155711
2000    cost: 0.0199938118458
3000    cost: 0.0199938118458
4000    cost: 0.0199938118458
5000    cost: 0.0199938118458
6000    cost: 0.0199938118458
7000    cost: 0.0199938118458
8000    cost: 0.0199938118458
9000    cost: 0.0199938118458
```




```
accuracy: 0.998270090087
```