TensorFlow 설치

In [2]:

```
import tensorflow as tf
```

In [3]:

```
tf.__version__
```

Out[3]:

```
'1.14.0'
```

Hypothesis

$$H(x) = Wx + b$$
$$H(x1, x2, x3) = w1x1 + w2x2 + w3x3 + b$$

Cost function

$$H(x1, x2, x3) = w1x1 + w2x2 + w3x3 + b$$
$$cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} (H(x1^{(i)}, x2^{(i)}, x3^{(i)}) - y^{(i)})^2$$

In [ ]:

```

```

EX1. x, y 값이 주어졌을 때, W, b 값을 예측하기

1. 그래프 빌드

2. 세션을 통해 그래프를 실행

3. 실행결과가 그래프를 업데이트

In [5]:

```
from tqdm import tqdm_notebook
import tensorflow as tf

tf.set_random_seed(777)  # for reproducibility

x1_data = [73., 93., 89., 96., 73.]
x2_data = [80., 88., 91., 98., 66.]
x3_data = [75., 93., 90., 100., 70.]

y_data = [152., 185., 180., 196., 142.]

# placeholders for a tensor that will be always fed.
x1 = tf.placeholder(tf.float32)
x2 = tf.placeholder(tf.float32)
x3 = tf.placeholder(tf.float32)

Y  = tf.placeholder(tf.float32)

w1 = tf.Variable(tf.random_normal([1]), name='weight1')
w2 = tf.Variable(tf.random_normal([1]), name='weight2')
w3 = tf.Variable(tf.random_normal([1]), name='weight3')
b  = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = x1 * w1 + x2 * w2 + x3 * w3 + b
print(hypothesis)

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize. Need a very small learning rate for this data set
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)
```

```
Tensor("add_5:0", dtype=float32)
```

In [6]:

```
from tqdm import tqdm_notebook

# Launch the graph in a session.
sess = tf.Session()

# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

for step in tqdm_notebook(range(2001)):
    cost_val, hy_val, _ = sess.run([cost, hypothesis, train],
                                   feed_dict={x1: x1_data, x2: x2_data, x3: x3_data, Y: y_data})

    if step % 100 == 0 or step < 10 :
        print("₩nStep : {} ₩nCost : {} ₩nPrediction :₩n{}".format(step, cost_val, hy_val))
```

```
HBox(children=(IntProgress(value=0, max=2001), HTML(value='')))


Step : 0
Cost : 36010.02734375
Prediction :
[-16.280907 -17.989803 -18.515007 -22.175758 -11.355569]

Step : 1
Cost : 11287.6767578125
Prediction :
[57.82734  71.08274  69.24972  73.39786  56.584312]

Step : 2
Cost : 3538.53369140625
Prediction :
[ 99.31784  120.951164 118.385956 126.90603   94.62132 ]

Step : 3
Cost : 1109.588134765625
Prediction :
[122.54687 148.87067 145.89555 156.86331 115.91684]

Step : 4
Cost : 348.2430419921875
Prediction :
[135.55196 164.50177 161.29715 173.6353  127.83938]

Step : 5
Cost : 109.6019515991211
Prediction :
[142.83302 173.25302 169.91994 183.02533 134.51434]

Step : 6
Cost : 34.80030059814453
Prediction :
[146.90945 178.15253 174.74754 188.2825  138.25142]

Step : 7
Cost : 11.354058265686035
Prediction :
[149.1917  180.89558 177.45032 191.2258  140.34361]

Step : 8
```

```
Cost : 4.00492525100708
Prediction :
[150.46944 182.43129 178.9635  192.87364 141.51494]

Step : 9
Cost : 1.7012488842010498
Prediction :
[151.18483 183.2911  179.81071 193.79626 142.17073]

Step : 100
Cost : 0.6468736529350281
Prediction :
[152.09683 184.38309 180.88858 194.97302 143.00008]

Step : 200
Cost : 0.6439381837844849
Prediction :
[152.09897 184.38124 180.88878 194.97656 142.99483]

Step : 300
Cost : 0.6410459280014038
Prediction :
[152.10104 184.37949 180.88896 194.98007 142.98969]

Step : 400
Cost : 0.6381894946098328
Prediction :
[152.10301 184.37779 180.88913 194.98357 142.98463]

Step : 500
Cost : 0.6353749632835388
Prediction :
[152.10487 184.37616 180.88927 194.987   142.97966]

Step : 600
Cost : 0.6325802803039551
Prediction :
[152.10669 184.3746  180.88939 194.99046 142.9748 ]

Step : 700
Cost : 0.629828929901123
Prediction :
[152.10838 184.3731  180.88948 194.99385 142.96999]

Step : 800
Cost : 0.6270951628684998
Prediction :
[152.11002 184.37166 180.88956 194.99727 142.96529]

Step : 900
Cost : 0.6244239211082458
Prediction :
[152.11156 184.37024 180.8896  195.0006  142.96066]

Step : 1000
Cost : 0.6217418909072876
Prediction :
[152.11302 184.36891 180.8896  195.00394 142.9561 ]

Step : 1100
Cost : 0.6191033124923706
```

```
Prediction :
[152.1144  184.36761 180.8896  195.00725 142.95161]

Step : 1200
Cost : 0.6164812445640564
Prediction :
[152.1157  184.3664  180.88956 195.01053 142.9472 ]

Step : 1300
Cost : 0.6139095425605774
Prediction :
[152.11696 184.36522 180.88954 195.0138  142.94289]

Step : 1400
Cost : 0.6113581657409668
Prediction :
[152.11813 184.36407 180.88947 195.01703 142.93864]

Step : 1500
Cost : 0.6088140606880188
Prediction :
[152.11925 184.363   180.88939 195.02026 142.93445]

Step : 1600
Cost : 0.6063039898872375
Prediction :
[152.12029 184.36195 180.88928 195.02347 142.93033]

Step : 1700
Cost : 0.6038143038749695
Prediction :
[152.12125 184.36095 180.88914 195.02663 142.92625]

Step : 1800
Cost : 0.6013458967208862
Prediction :
[152.12215 184.36    180.88899 195.02977 142.92226]

Step : 1900
Cost : 0.5989128351211548
Prediction :
[152.12302 184.35909 180.88885 195.03291 142.91833]

Step : 2000
Cost : 0.5964996814727783
Prediction :
[152.12381 184.35823 180.8887  195.03603 142.91447]
```

EX2. Plaeholder 사용

데이터의 형태만 지정하고 실제 데이터는 실행단계에서 입력받는 방법

1.그래프 빌드

2.세션을 통해 그래프를 실행

3.실행결과가 그래프를 업데이트

In [7]:

```
import tensorflow as tf

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

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

# Our hypothesis XW+b
hypothesis = X * W + b

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()

# Initializes global variables in the graph.
# sess.run(tf.initialize_all_variables())
sess.run(tf.global_variables_initializer())
```

In [8]:

```
steps     = []
cost_vals = []
W_vals    = []
b_vals    = []
for step in tqdm_notebook(range(2001)):
    cost_val, W_val, b_val, _ = sess.run([cost, W, b, train],
                                         feed_dict={X: [1, 2, 3],
                                                    Y: [1, 2, 3]})
    steps.append(step)
    cost_vals.append(cost_val)
    W_vals.append(W_val)
    b_vals.append(b_val)
    # W_vals.append(float(W_val))
    # b_vals.append(float(b_val))

    if step % 200 == 0 or step < 5:
        print("step={step}, ₩t cost={cost_val}, ₩t W={W_val}, ₩t b={b_val}".format(
            step=step, cost_val=cost_val, W_val=W_val, b_val=b_val
            ));
```

```
HBox(children=(IntProgress(value=0, max=2001), HTML(value='')))

step=0,          cost=39.70097732543945,        W=[-0.80500764],        b=[-1.79648
88]
step=1,          cost=31.40232276916504,        W=[-0.5646807],         b=[-1.68835
87]
step=2,          cost=24.842575073242188,       W=[-0.3511095],         b=[-1.59200
43]
step=3,          cost=19.657350540161133,       W=[-0.16132578],        b=[-1.50611
98]
step=4,          cost=15.558608055114746,       W=[0.00730941],         b=[-1.42954
44]
step=200,        cost=0.038831520825624466,     W=[1.2283194],          b=[-0.51902
36]
step=400,        cost=0.014827747829258442,     W=[1.1410873],          b=[-0.32072
487]
step=600,        cost=0.005661958362907171,     W=[1.0871834],          b=[-0.19818
835]
step=800,        cost=0.0021620059851557016,    W=[1.0538739],          b=[-0.12246
828]
step=1000,       cost=0.0008255562861450016,    W=[1.0332907],          b=[-0.07567
774]
step=1200,       cost=0.0003152418939862400З,   W=[1.0205718],          b=[-0.04676
443]
step=1400,       cost=0.00012037125270580873,   W=[1.0127119],          b=[-0.02889
728]
step=1600,       cost=4.596456346916966e-05,    W=[1.0078552],          b=[-0.01785
678]
step=1800,       cost=1.7552025383338332e-05,   W=[1.0048541],          b=[-0.01103
447]
step=2000,       cost=6.702131940983236e-06,    W=[1.0029995],          b=[-0.00681
877]
```

Ex3. Linear Regression

In [9]:

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


## x,y 값 설정
number_of_points = 200
x_point = []
y_point = []

w = 0.25
b = 0.75

for i in range(number_of_points):
    x = np.random.normal(0.0, 0.5)
    y = w*x + b +np.random.normal(0.0, 0.1)
    x_point.append([x])
    y_point.append([y])
```

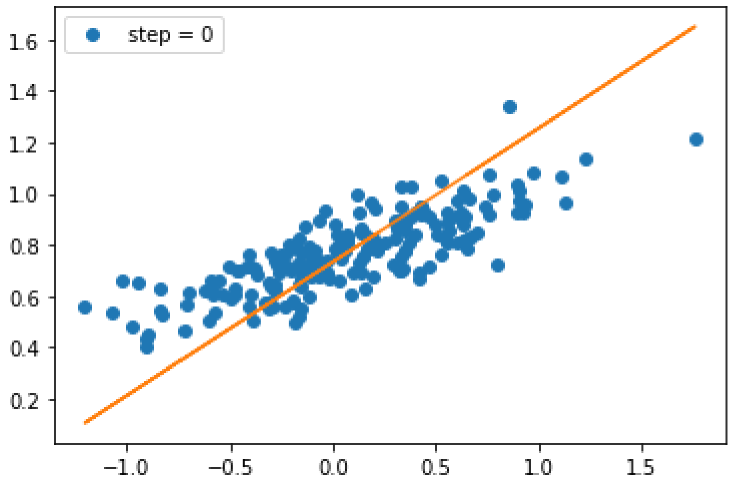
In [10]:

```
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
B = tf.Variable(tf.zeros([1]))
y = W * x_point + B # Hypothesis

# Computes the mean of elements across dimensions of a tensor
cost_function = tf.reduce_mean(tf.square(y - y_point))

# Optimizer that implements the gradient descent algorithm
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.5)

# Add operations to minimize cost_function
train = optimizer.minimize(cost_function)

# Returns an Op that initializes global variables
# model = tf.initialize_all_variables()
model = tf.global_variables_initializer()
```

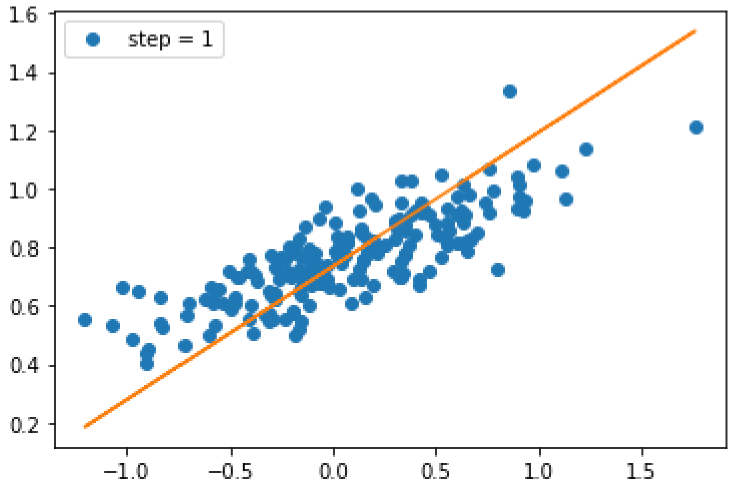
In [11]:

```
with tf.Session() as sess:
    sess.run(model)
    for step in range(0,2001):
        sess.run(train)
        if (step % 200) == 0 or step<5:
            print("\n y = {w} x + {b} ".format(w=sess.run(W), b=sess.run(B)))
            plt.plot(x_point,y_point,'o',label='step = {}'.format(step))
            plt.plot(x_point,sess.run(W)*x_point+sess.run(B))
            plt.legend(loc=2)
            plt.show()
```

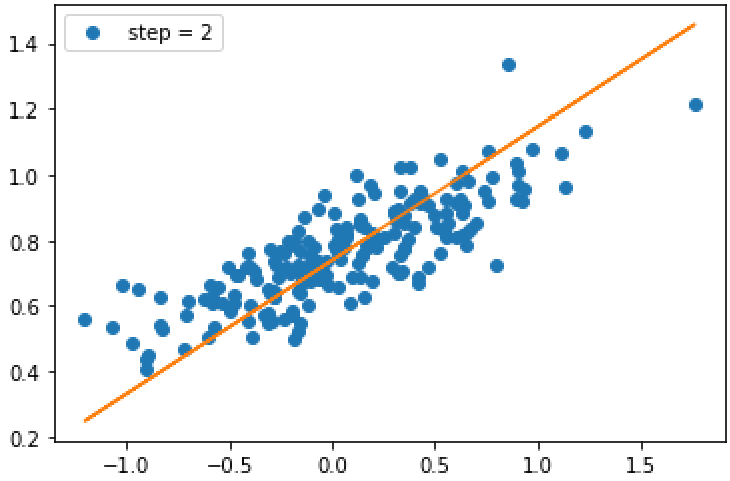
```
y = [0.52068263] x + [0.73392]
```



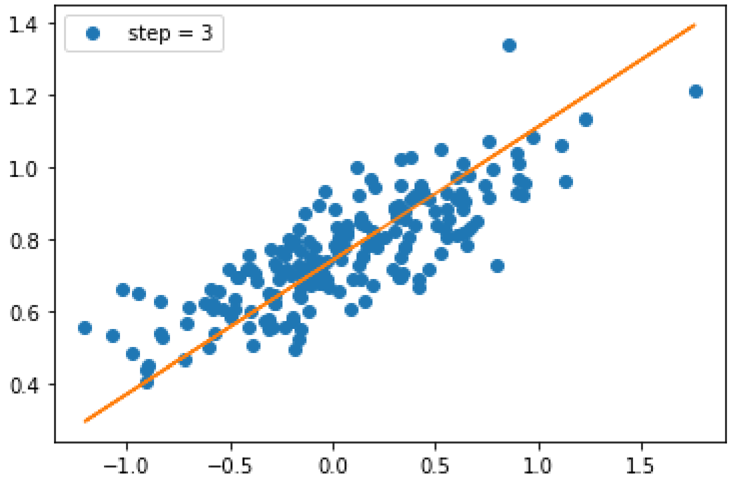

```
y = [0.4565214] x + [0.73557264]
```



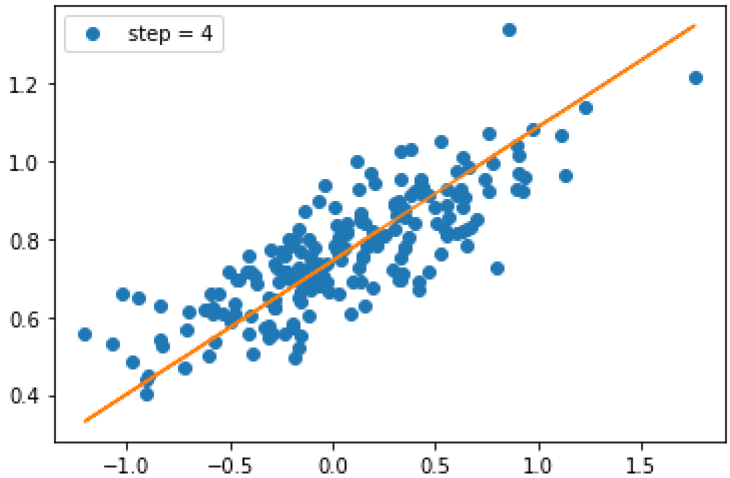

```
y = [0.40784615] x + [0.73929757]
```



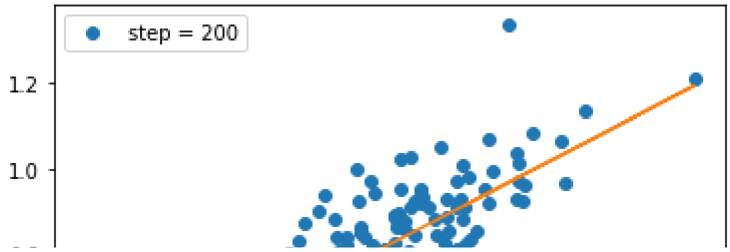
y = [0.37077573] x + [0.7421234]



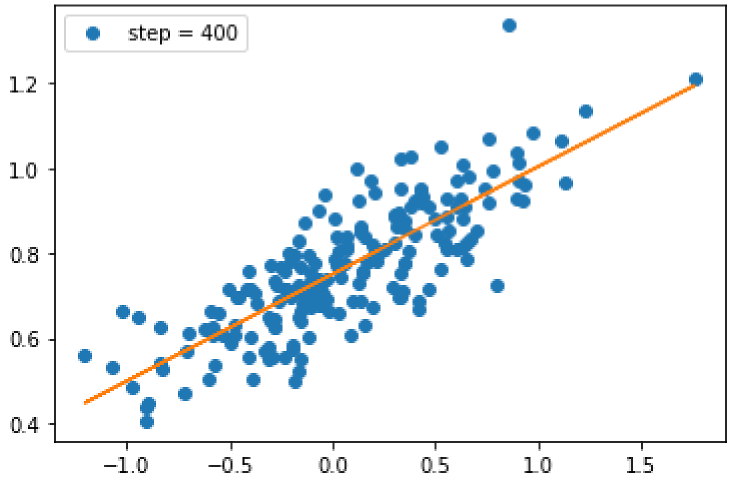
y = [0.34254402] x + [0.74427557]



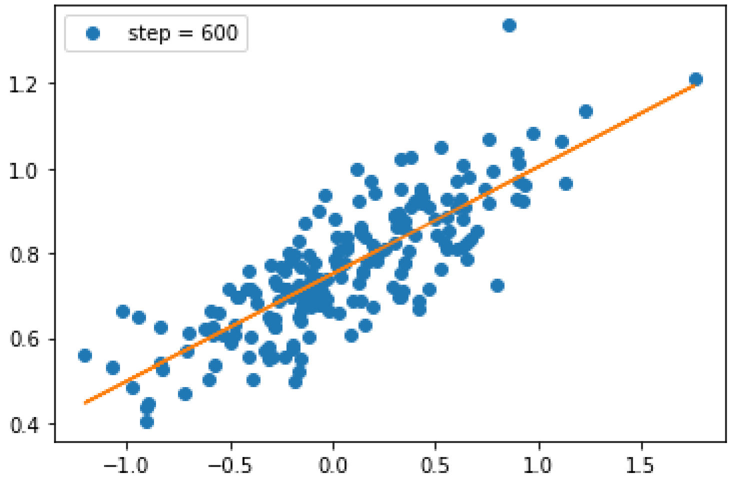
y = [0.25236934] x + [0.7511497]



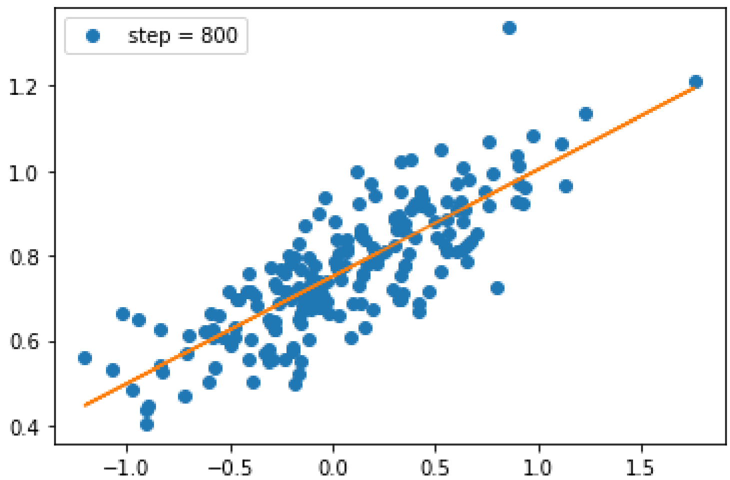
y = [0.25236934] x + [0.7511497]



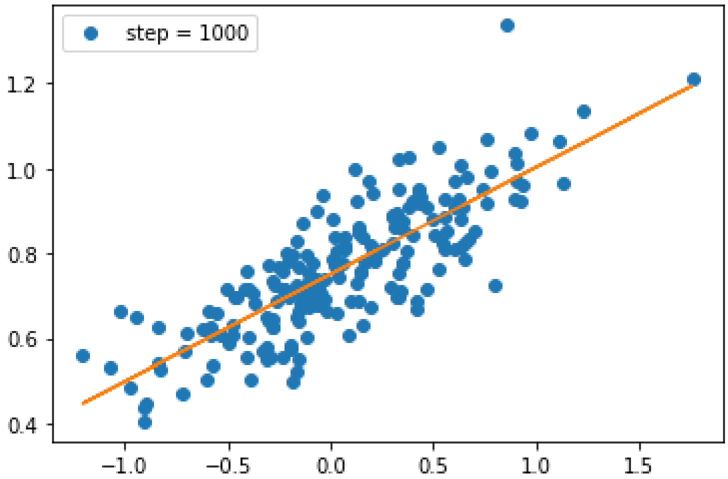
y = [0.25236934] x + [0.7511497]



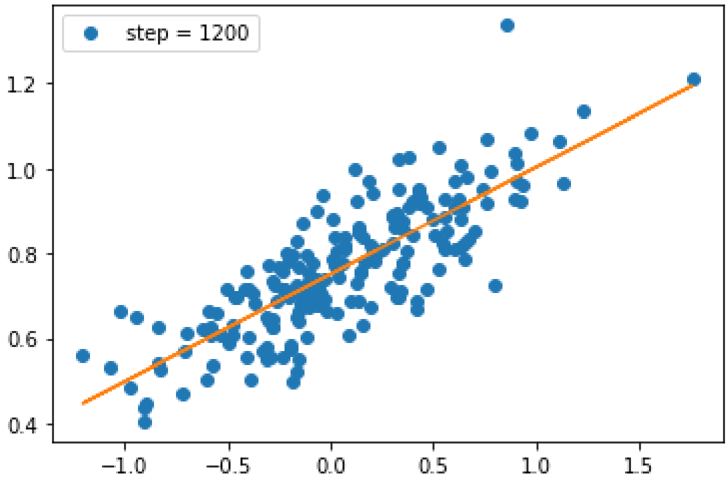
y = [0.25236934] x + [0.7511497]



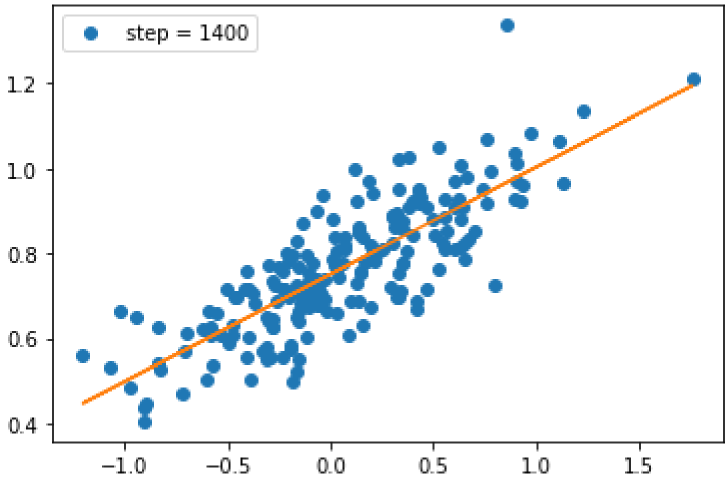
y = [0.25236934] x + [0.7511497]



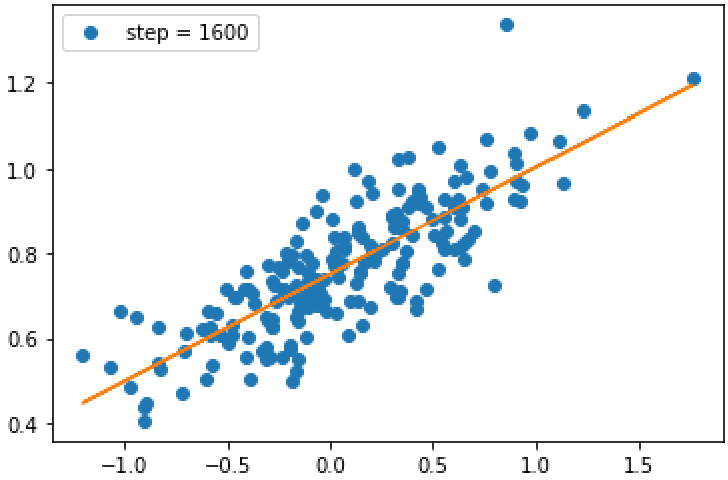
y = [0.25236934] x + [0.7511497]



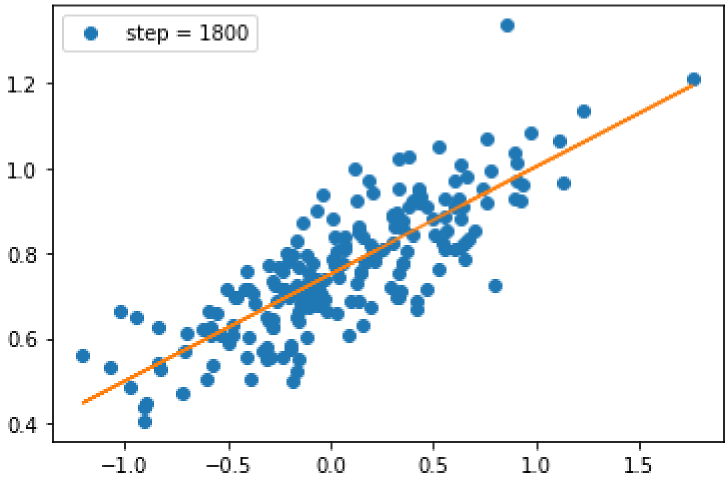
y = [0.25236934] x + [0.7511497]



```
y = [0.25236934] x + [0.7511497]
```




```
y = [0.25236934] x + [0.7511497]
```




```
y = [0.25236934] x + [0.7511497]
```

In [ ]:

Cost minimize

1. Our hypothesis for linear model X * W

hypothesis = X * W

$$H(x) = Wx$$

2. cost/loss function

cost = tf.reducemean(tf.square(hypothesis - Y))

$$cost(W) = \frac{1}{m} \sum^{m} i = 1(H(x^{(i)} - y^{(i)})^2)$$

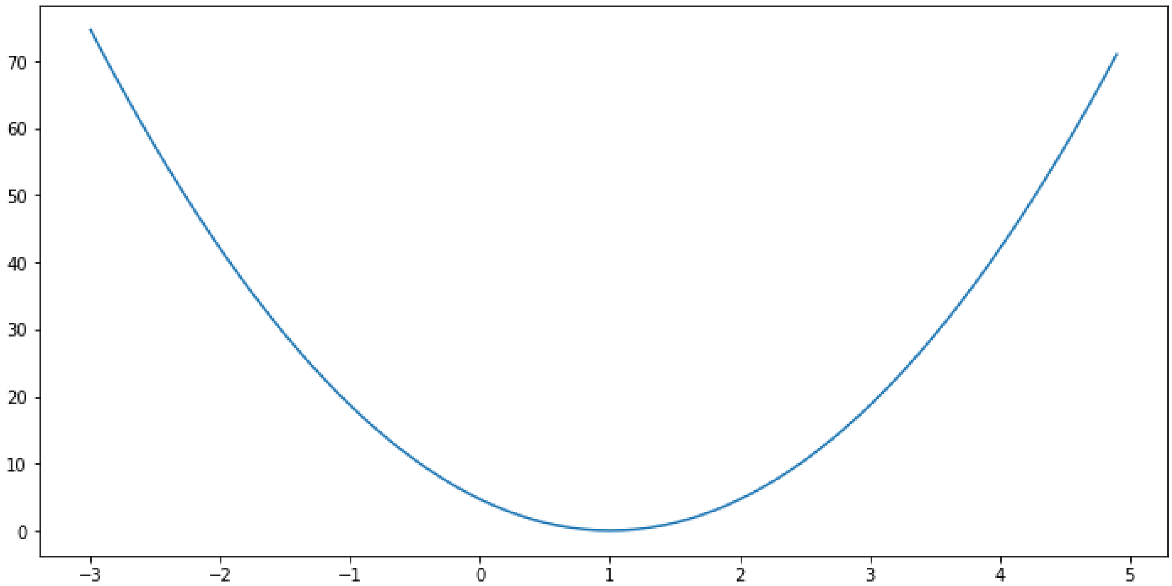
3. Gradient descent

$$cost(W) = \frac{1}{m} \sum_{i=1}^{m} (H(x^{(i)} - y^{(i)})^2$$

$$W := W - a\frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)})x^{(i)}$$

In [12]:

```
## Cost/ loss func.

import tensorflow as tf
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = [12,6]

X = [1, 2, 3]
Y = [1, 2, 3]

W = tf.placeholder(tf.float32)

# Our hypothesis for linear model X * W
hypothesis = X * W

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Launch the graph in a session.
sess = tf.Session()

# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

# Variables for plotting cost function
W_val = []
cost_val = []
for i in range(-30, 50):
    feed_W = i * 0.1
    curr_cost, curr_W = sess.run([cost, W], feed_dict={W: feed_W})
    W_val.append(curr_W)
    cost_val.append(curr_cost)

# Show the cost function
plt.plot(W_val, cost_val)
plt.show()
```



In [ ]:

```

```

In [16]:

```
## Gradient descent
import tensorflow as tf

# tf Graph Input
X = [1, 2, 3]
Y = [1, 2, 3]

# Set wrong model weights
W = tf.Variable(5.0)

# Linear model
hypothesis = X * W

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize: Gradient Descent Magic
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()
```

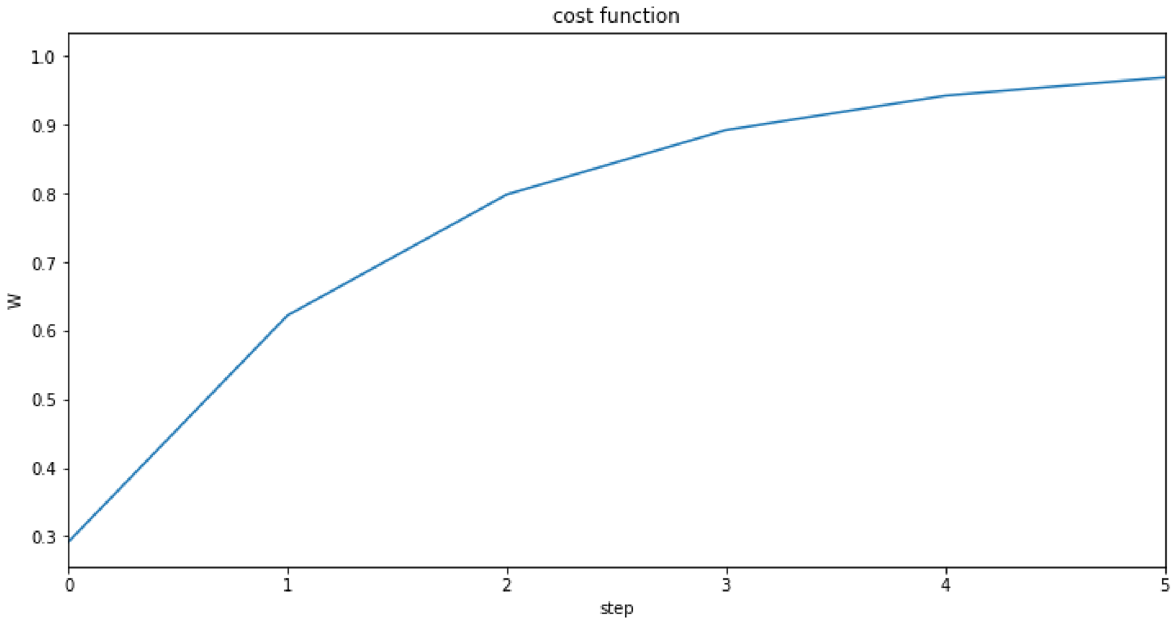
In [17]:

```
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

W_val = []
step_val = []

for step in range(101):
    W_val.append(sess.run(W))
    step_val.append(step)

    if step % 10 == 0 or step < 10:
        # print(step, sess.run(W))
        print("step={step}, ₩t W={W_val}".format(
            step=step, W_val=sess.run(W)
            ));

    sess.run(train)
```

```
step=0,          W=5.0
step=1,          W=1.2666664123535156
step=2,          W=1.0177778005599976
step=3,          W=1.0011851787567139
step=4,          W=1.0000790357589722
step=5,          W=1.0000052452087402
step=6,          W=1.0000003576278687
step=7,          W=1.0
step=8,          W=1.0
step=9,          W=1.0
step=10,         W=1.0
step=20,         W=1.0
step=30,         W=1.0
step=40,         W=1.0
step=50,         W=1.0
step=60,         W=1.0
step=70,         W=1.0
step=80,         W=1.0
step=90,         W=1.0
step=100,        W=1.0
```

In [21]:

```
# Show the cost function
plt.plot(step_val, W_val)
plt.title('cost function')
plt.xlabel('step')
plt.ylabel('W')
plt.xlim(0, 5)
plt.show()
```



In [ ]:

```

```

Ex. Multi-variable matmul linear regression

Hypothesis using matrix

$$w1x1 + w2x2 + w3x3 + \ldots + wnxn$$

[x{1} x{2} x{3}] \times \begin{bmatrix} w{1}\ w{2}\ w{3}

\end{bmatrix}

$$[x_1 w_1 + x_2 w_2 + x_3 w_3]$$

$$H(X) = XW$$

TIP. 행렬식 생각

In [22]:

```
import tensorflow as tf

tf.set_random_seed(777)  # for reproducibility

x_data = [[73., 80., 75.],
          [93., 88., 93.],
          [89., 91., 90.],
          [96., 98., 100.],
          [73., 66., 70.]]
y_data = [[152.], [185.], [180.], [196.], [142.]]
#x_data는 하나의 리스트당 3개의 데이터
#y_data는 하나의 리스트당 하나의 데이터



# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3]) #x_data는 하나의 리스트당 3개의 데이터
Y = tf.placeholder(tf.float32, shape=[None, 1]) #y_data는 하나의 리스트당 하나의 데이터

W = tf.Variable(tf.random_normal([3, 1]), name='weight') #3by1 matrix
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis
hypothesis = tf.matmul(X, W) + b

# Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)
```

In [23]:

```
# Launch the graph in a session.
sess = tf.Session()

# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

for step in tqdm_notebook(range(2001)):
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})

    if step % 100 == 0 or step < 10 :
        print("₩nStep : {} ₩nCost : {} ₩nPrediction :₩n{}".format(step, cost_val, hy_val))
```

```
HBox(children=(IntProgress(value=0, max=2001), HTML(value='')))


Step : 0
Cost : 970.3675537109375
Prediction :
[[131.93364 ]
 [146.66693 ]
 [150.79033 ]
 [164.23979 ]
 [108.565895]]

Step : 1
Cost : 328.1596984863281
Prediction :
[[143.87549]
 [161.02438]
 [164.93475]
 [179.64293]
 [119.51789]]

Step : 2
Cost : 126.84867095947266
Prediction :
[[150.56029]
 [169.06328]
 [172.85341]
 [188.26634]
 [125.65044]]

Step : 3
Cost : 63.735496520996094
Prediction :
[[154.3019 ]
 [173.56467]
 [177.28647]
 [193.09406]
 [129.08473]]

Step : 4
Cost : 43.94036102294922
Prediction :
[[156.39568]
 [176.08546]
 [179.76807]
 [195.79668]
```

```
 [131.00838]]

Step : 5
Cost : 37.722869873046875
Prediction :
[[157.56691]
 [177.49748]
 [181.15714]
 [197.30952]
 [132.08626]]

Step : 6
Cost : 35.761375427246094
Prediction :
[[158.22168]
 [178.2887 ]
 [181.93451]
 [198.15628]
 [132.69064]]

Step : 7
Cost : 35.13386917114258
Prediction :
[[158.58725]
 [178.73235]
 [182.36943]
 [198.63011]
 [133.02992]]

Step : 8
Cost : 34.92456817626953
Prediction :
[[158.79094]
 [178.98141]
 [182.61264]
 [198.89517]
 [133.22078]]

Step : 9
Cost : 34.84640884399414
Prediction :
[[158.90399]
 [179.12152]
 [182.74852]
 [199.04332]
 [133.32854]]

Step : 100
Cost : 33.18610382080078
Prediction :
[[158.84872]
 [179.43661]
 [182.86081]
 [199.18428]
 [133.64833]]

Step : 200
Cost : 31.481945037841797
Prediction :
[[158.63252]
 [179.5852 ]
```

```
 [182.79507]
 [199.13261]
 [133.84686]]

Step : 300
Cost : 29.867481231689453
Prediction :
[[158.42216]
 [179.72986]
 [182.73116]
 [199.08226]
 [134.04015]]

Step : 400
Cost : 28.33807945251465
Prediction :
[[158.21748]
 [179.8706 ]
 [182.66899]
 [199.03316]
 [134.22832]]

Step : 500
Cost : 26.8891658782959
Prediction :
[[158.0183 ]
 [180.00754]
 [182.60846]
 [198.98528]
 [134.4115 ]]

Step : 600
Cost : 25.516590118408203
Prediction :
[[157.82451]
 [180.14082]
 [182.5496 ]
 [198.93863]
 [134.58984]]

Step : 700
Cost : 24.21615982055664
Prediction :
[[157.63593]
 [180.27051]
 [182.49234]
 [198.89313]
 [134.76349]]

Step : 800
Cost : 22.9842529296875
Prediction :
[[157.45244]
 [180.39671]
 [182.43665]
 [198.84877]
 [134.93253]]

Step : 900
Cost : 21.817108154296875
Prediction :
```

```
[[157.2739 ]
 [180.51952]
 [182.38246]
 [198.80553]
 [135.09712]]

Step : 1000
Cost : 20.711444854736328
Prediction :
[[157.10017]
 [180.639  ]
 [182.32974]
 [198.76334]
 [135.25734]]

Step : 1100
Cost : 19.663862228393555
Prediction :
[[156.93115]
 [180.75531]
 [182.27849]
 [198.72223]
 [135.41338]]

Step : 1200
Cost : 18.671432495117188
Prediction :
[[156.76668]
 [180.86845]
 [182.2286 ]
 [198.68211]
 [135.56526]]

Step : 1300
Cost : 17.731136322021484
Prediction :
[[156.60666]
 [180.97858]
 [182.18008]
 [198.643  ]
 [135.71317]]

Step : 1400
Cost : 16.84030532836914
Prediction :
[[156.45094]
 [181.08572]
 [182.13287]
 [198.60486]
 [135.85715]]

Step : 1500
Cost : 15.996325492858887
Prediction :
[[156.29945]
 [181.18999]
 [182.08696]
 [198.56766]
 [135.99734]]

Step : 1600
```

```
Cost : 15.196737289428711
Prediction :
[[156.15205]
 [181.29144]
 [182.04233]
 [198.53139]
 [136.13385]]

Step : 1700
Cost : 14.439193725585938
Prediction :
[[156.00865]
 [181.39015]
 [181.99889]
 [198.496  ]
 [136.26675]]

Step : 1800
Cost : 13.72138786315918
Prediction :
[[155.8691 ]
 [181.4862 ]
 [181.95665]
 [198.46147]
 [136.39616]]

Step : 1900
Cost : 13.041348457336426
Prediction :
[[155.73332]
 [181.57967]
 [181.91554]
 [198.42778]
 [136.52214]]

Step : 2000
Cost : 12.396985054016113
Prediction :
[[155.60126]
 [181.67064]
 [181.8756 ]
 [198.39494]
 [136.64485]]
```

Ex. File input linear regression

In [18]:

```
import tensorflow as tf
import numpy as np

tf.set_random_seed(777)  # for reproducibility

xy = np.loadtxt('./data/data-01-test-score.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# Make sure the shape and data are OK
print("x_data.shape : {}, ₩tlen(x_data) : {} ₩nx_data : ₩n{}".format(x_data.shape, len(x_data), x_da
print("-"*25)
print("y_data.shape : {}  ₩ny_data : ₩n{}".format(y_data.shape, y_data))

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([3, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis
hypothesis = tf.matmul(X, W) + b

# Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)
```

```
x_data.shape : (25, 3),         len(x_data) : 25
x_data :
[[ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]
 [ 53.  46.  55.]
 [ 69.  74.  77.]
 [ 47.  56.  60.]
 [ 87.  79.  90.]
 [ 79.  70.  88.]
 [ 69.  70.  73.]
 [ 70.  65.  74.]
 [ 93.  95.  91.]
 [ 79.  80.  73.]
 [ 70.  73.  78.]
 [ 93.  89.  96.]
 [ 78.  75.  68.]
 [ 81.  90.  93.]
 [ 88.  92.  86.]
 [ 78.  83.  77.]
 [ 82.  86.  90.]
 [ 86.  82.  89.]
 [ 78.  83.  85.]
 [ 76.  83.  71.]
 [ 96.  93.  95.]]
```

```
-------------------------
y_data.shape : (25, 1)
y_data :
[[152.]
 [185.]
 [180.]
 [196.]
 [142.]
 [101.]
 [149.]
 [115.]
 [175.]
 [164.]
 [141.]
 [141.]
 [184.]
 [152.]
 [148.]
 [192.]
 [147.]
 [183.]
 [177.]
 [159.]
 [177.]
 [175.]
 [175.]
 [149.]
 [192.]]
```

In [19]:

```
# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

for step in range(2001):
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})

    if step % 100 == 0 or step < 10 :
        print("\nStep : {} \nCost : {} \nPrediction :\n{}".format(step, cost_val, hy_val))
```

```
Step : 0
Cost : 47829.4609375
Prediction :
[[-66.17106 ]
 [-51.897587]
 [-65.661255]
 [-68.70268 ]
 [-34.906334]
 [-17.936304]
 [-54.824265]
 [-46.26666 ]
 [-38.53811 ]
 [-26.858534]
 [-47.460365]
 [-33.449764]
 [-70.2755  ]
 [-60.52128 ]
 [-50.618046]
 [-52.518322]
 [-52.461407]
 [-70.53982 ]
 [-71.69783 ]
 [-66.57909 ]
 [-61.42079 ]
 [-47.64652 ]
 [-61.698124]
 [-72.529816]
 [-59.497818]]

Step : 1
Cost : 17762.474609375
Prediction :
[[14.325285]
 [44.842545]
 [29.663145]
 [35.111603]
 [38.872536]
 [36.454933]
 [22.885138]
 [11.343676]
 [51.868866]
 [56.876877]
 [27.407148]
 [40.366737]
 [28.208838]
 [21.349142]
```

```
 [27.447058]
 [45.649857]
 [25.508062]
 [22.72038 ]
 [22.194487]
 [17.432928]
 [29.704916]
 [43.108124]
 [25.18418 ]
 [ 8.632261]
 [40.764122]]

Step : 2
Cost : 6645.82080078125
Prediction :
[[ 63.27323 ]
 [103.66438 ]
 [ 87.626015]
 [ 98.23642 ]
 [ 83.73246 ]
 [ 69.52548 ]
 [ 70.13708 ]
 [ 46.374817]
 [106.838524]
 [107.788765]
 [ 72.93045 ]
 [ 85.24921 ]
 [ 88.09355 ]
 [ 71.131905]
 [ 74.91473 ]
 [105.33991 ]
 [ 72.91818 ]
 [ 79.42871 ]
 [ 79.287575]
 [ 68.51839 ]
 [ 85.11443 ]
 [ 98.29034 ]
 [ 78.013885]
 [ 57.985924]
 [101.72813 ]]

Step : 3
Cost : 2535.634765625
Prediction :
[[ 93.03807 ]
 [139.42986 ]
 [122.87113 ]
 [136.61989 ]
 [111.00814 ]
 [ 89.631905]
 [ 98.8693  ]
 [ 67.67661 ]
 [140.26042 ]
 [138.74222 ]
 [100.61091 ]
 [112.53826 ]
 [124.50762 ]
 [101.40367 ]
 [103.77752 ]
 [141.63322 ]
 [101.746635]
```

```
 [113.91157 ]
 [114.00476 ]
 [ 99.582794]
 [118.80662 ]
 [131.8426  ]
 [110.13783 ]
 [ 87.99839 ]
 [138.79689 ]]

Step : 4
Cost : 1015.9318237304688
Prediction :
[[111.13861 ]
 [161.1759  ]
 [144.30266 ]
 [159.9593  ]
 [127.5917  ]
 [101.85548 ]
 [116.34057 ]
 [ 80.63025 ]
 [160.58014 ]
 [157.55992 ]
 [117.44201 ]
 [129.12962 ]
 [146.65033 ]
 [119.81175 ]
 [121.3276  ]
 [163.7001  ]
 [119.27644 ]
 [134.8802  ]
 [135.11629 ]
 [118.473335]
 [139.29355 ]
 [152.24281 ]
 [129.67159 ]
 [106.250404]
 [161.33612 ]]

Step : 5
Cost : 454.00372314453125
Prediction :
[[122.146614]
 [174.39737 ]
 [157.3348  ]
 [174.15115 ]
 [137.67387 ]
 [109.28585 ]
 [126.96456 ]
 [ 88.50777 ]
 [172.93306 ]
 [168.99841 ]
 [127.6761  ]
 [139.2162  ]
 [160.11531 ]
 [131.00606 ]
 [131.99895 ]
 [177.11655 ]
 [129.93608 ]
 [147.63145 ]
 [147.95476 ]
 [129.96153 ]
```

```
 [151.75096 ]
 [164.6458  ]
 [141.54985 ]
 [117.35145 ]
 [175.04059 ]]

Step : 6
Cost : 246.19020080566406
Prediction :
[[128.84203 ]
 [182.43549 ]
 [165.25967 ]
 [182.78084 ]
 [143.8029  ]
 [113.801735]
 [133.4251  ]
 [ 93.298744]
 [180.44176 ]
 [175.95003 ]
 [133.89893 ]
 [145.34755 ]
 [168.30379 ]
 [137.81407 ]
 [138.48769 ]
 [185.27315 ]
 [136.41832 ]
 [155.38614 ]
 [155.76292 ]
 [136.94875 ]
 [159.32605 ]
 [172.18613 ]
 [148.77318 ]
 [124.10435 ]
 [183.37312 ]]

Step : 7
Cost : 169.30404663085938
Prediction :
[[132.91512]
 [187.32181]
 [170.07904]
 [188.0284 ]
 [147.52815]
 [116.54543]
 [137.35399]
 [ 96.21294]
 [185.0049 ]
 [180.17332]
 [137.68262]
 [149.0739 ]
 [173.28383]
 [141.95491]
 [142.43315]
 [190.23138]
 [140.36044]
 [160.10262]
 [160.51227]
 [141.19905]
 [163.93237]
 [176.76965]
 [153.16605]
```

```
 [128.21329]
 [188.4392 ]]

Step : 8
Cost : 140.82476806640625
Prediction :
[[135.39369 ]
 [190.29169 ]
 [173.01013 ]
 [191.21945 ]
 [149.79181 ]
 [118.211555]
 [139.74352 ]
 [ 97.98595 ]
 [187.77698 ]
 [182.73769 ]
 [139.98323 ]
 [151.33788 ]
 [176.31303 ]
 [144.47401 ]
 [144.83217 ]
 [193.24492 ]
 [142.75807 ]
 [162.97171 ]
 [163.40173 ]
 [143.78523 ]
 [166.73357 ]
 [179.55531 ]
 [155.83786 ]
 [130.71461 ]
 [191.51907 ]]

Step : 9
Cost : 130.24295043945312
Prediction :
[[136.90274]
 [192.0963 ]
 [174.79303]
 [193.16005]
 [151.16675]
 [119.22247]
 [141.19705]
 [ 99.06507]
 [189.46002]
 [184.29333]
 [141.38203]
 [152.71268]
 [178.15599]
 [146.00702]
 [146.29091]
 [195.07594]
 [144.21655]
 [164.71751]
 [165.16032]
 [145.35954]
 [168.43715]
 [181.24777]
 [157.46318]
 [132.23842]
 [193.39128]]
```

```
Step : 100
Cost : 116.80328369140625
Prediction :
[[139.6783  ]
 [194.61378 ]
 [177.7034  ]
 [196.23145 ]
 [152.96439 ]
 [120.300644]
 [143.57977 ]
 [100.97266 ]
 [191.503   ]
 [185.89537 ]
 [143.5345  ]
 [154.43626 ]
 [181.24847 ]
 [148.66258 ]
 [148.55147 ]
 [197.6128  ]
 [146.61067 ]
 [167.70482 ]
 [168.25021 ]
 [148.19505 ]
 [171.1487  ]
 [183.56625 ]
 [160.14575 ]
 [135.24135 ]
 [196.17485 ]]

Step : 200
Cost : 109.3571548461914
Prediction :
[[140.14665 ]
 [194.3083  ]
 [177.85834 ]
 [196.29768 ]
 [152.60008 ]
 [119.7725  ]
 [143.72072 ]
 [101.230576]
 [190.89085 ]
 [185.02425 ]
 [143.51662 ]
 [153.99614 ]
 [181.49767 ]
 [148.95721 ]
 [148.552   ]
 [197.2855  ]
 [146.74596 ]
 [168.0107  ]
 [168.63779 ]
 [148.61726 ]
 [171.22578 ]
 [183.23543 ]
 [160.322   ]
 [135.92647 ]
 [196.04254 ]]

Step : 300
Cost : 102.44879913330078
Prediction :
```

```
[[140.59738 ]
 [194.0128  ]
 [178.00659 ]
 [196.36209 ]
 [152.24669 ]
 [119.264404]
 [143.85965 ]
 [101.48512 ]
 [190.30194 ]
 [184.18945 ]
 [143.50049 ]
 [153.57361 ]
 [181.73495 ]
 [149.23631 ]
 [148.55579 ]
 [196.97067 ]
 [146.8695  ]
 [168.3103  ]
 [169.00833 ]
 [149.02188 ]
 [171.30302 ]
 [182.9173  ]
 [160.49423 ]
 [136.58197 ]
 [195.91313 ]]

Step : 400
Cost : 96.03717803955078
Prediction :
[[141.03116 ]
 [193.72693 ]
 [178.14844 ]
 [196.42471 ]
 [151.90382 ]
 [118.775604]
 [143.9965  ]
 [101.73627 ]
 [189.73532 ]
 [183.38943 ]
 [143.486   ]
 [153.16794 ]
 [181.96088 ]
 [149.50055 ]
 [148.56253 ]
 [196.66779 ]
 [146.98196 ]
 [168.60379 ]
 [169.36261 ]
 [149.4096  ]
 [171.38031 ]
 [182.61134 ]
 [160.66248 ]
 [137.2091  ]
 [195.78656 ]]

Step : 500
Cost : 90.08436584472656
Prediction :
[[141.4487  ]
 [193.45041 ]
 [178.28421 ]
```

```
 [196.48564 ]
 [151.57117 ]
 [118.305305]
 [144.13133 ]
 [101.98395 ]
 [189.19014 ]
 [182.62276 ]
 [143.47308 ]
 [152.77841 ]
 [182.176   ]
 [149.75073 ]
 [148.57208 ]
 [196.37637 ]
 [147.08401 ]
 [168.89124 ]
 [169.70135 ]
 [149.78122 ]
 [171.45758 ]
 [182.31708 ]
 [160.82687 ]
 [137.80916 ]
 [195.66283 ]]

Step : 600
Cost : 84.55567169189453
Prediction :
[[141.85059]
 [193.18285]
 [178.41415]
 [196.54489]
 [151.2484 ]
 [117.85276]
 [144.2641 ]
 [102.22815]
 [188.66551]
 [181.88797]
 [143.46164]
 [152.4044 ]
 [182.38075]
 [149.98747]
 [148.58424]
 [196.096  ]
 [147.17618]
 [169.17271]
 [170.0252 ]
 [150.13734]
 [171.53477]
 [182.03406]
 [160.98746]
 [138.38329]
 [195.54184]]

Step : 700
Cost : 79.4189682006836
Prediction :
[[142.23747 ]
 [192.92395 ]
 [178.53848 ]
 [196.60251 ]
 [150.93518 ]
 [117.41728 ]
```

```
 [144.39478 ]
 [102.468834]
 [188.16064 ]
 [181.18375 ]
 [143.45157 ]
 [152.04526 ]
 [182.57565 ]
 [150.21143 ]
 [148.5988  ]
 [195.82617 ]
 [147.25903 ]
 [169.44836 ]
 [170.33484 ]
 [150.47865 ]
 [171.61177 ]
 [181.76178 ]
 [161.14435 ]
 [138.93259 ]
 [195.42351 ]]

Step : 800
Cost : 74.64476776123047
Prediction :
[[142.60992]
 [192.6734 ]
 [178.65746]
 [196.65854]
 [150.6312 ]
 [116.99818]
 [144.52338]
 [102.70597]
 [187.67471]
 [180.50876]
 [143.44281]
 [151.70033]
 [182.76111]
 [150.42322]
 [148.61557]
 [195.56651]
 [147.33313]
 [169.71823]
 [170.63083]
 [150.80576]
 [171.68848]
 [181.49988]
 [161.29759]
 [139.45813]
 [195.30782]]

Step : 900
Cost : 70.20587921142578
Prediction :
[[142.96855]
 [192.43097]
 [178.77135]
 [196.71307]
 [150.33623]
 [116.59482]
 [144.64993]
 [102.93955]
 [187.20706]
```

```
 [179.86182]
 [143.43527]
 [151.36911]
 [182.93762]
 [150.62347]
 [148.63443]
 [195.31664]
 [147.39899]
 [169.98244]
 [170.91385]
 [151.11932]
 [171.76494]
 [181.2479 ]
 [161.44733]
 [139.96101]
 [195.19476]]

Step : 1000
Cost : 66.07707214355469
Prediction :
[[143.31386 ]
 [192.19627 ]
 [178.88031 ]
 [196.7661  ]
 [150.0499  ]
 [116.206566]
 [144.77441 ]
 [103.16957 ]
 [186.75685 ]
 [179.24168 ]
 [143.42888 ]
 [151.05095 ]
 [183.10555 ]
 [150.8127  ]
 [148.65517 ]
 [195.07608 ]
 [147.45708 ]
 [170.24109 ]
 [171.18443 ]
 [151.41985 ]
 [171.84097 ]
 [181.00543 ]
 [161.59355 ]
 [140.44215 ]
 [195.08418 ]]

Step : 1100
Cost : 62.235450744628906
Prediction :
[[143.6464 ]
 [191.96916]
 [178.98462]
 [196.81773]
 [149.772  ]
 [115.83283]
 [144.89687]
 [103.39602]
 [186.32347]
 [178.6473 ]
 [143.42358]
 [150.74539]
```

```
 [183.26534]
 [150.99147]
 [148.67769]
 [194.84459]
 [147.50784]
 [170.49431]
 [171.4431 ]
 [151.70795]
 [171.91661]
 [180.77216]
 [161.73642]
 [140.90253]

 [194.97615]]

Step : 1200
Cost : 58.65964889526367
Prediction :
[[143.96664 ]
 [191.74925 ]
 [179.08443 ]
 [196.86794 ]
 [149.50221 ]
 [115.473045]
 [145.01721 ]
 [103.61887 ]
 [185.90623 ]
 [178.0775  ]
 [143.41928 ]
 [150.45186 ]
 [183.4173  ]
 [151.16026 ]
 [148.7018  ]
 [194.6217  ]
 [147.55174 ]
 [170.74211 ]
 [171.69041 ]
 [151.98409 ]
 [171.99174 ]
 [180.54765 ]
 [161.87595 ]
 [141.343   ]
 [194.87053 ]]

Step : 1300
Cost : 55.3299674987793
Prediction :
[[144.27507]
 [191.53636]
 [179.17995]
 [196.91676]
 [149.24031]
 [115.12663]
 [145.13553]
 [103.83815]
 [185.50449]
 [177.53125]
 [143.41592]
 [150.16988]
 [183.56181]
 [151.31956]
```

```
 [148.72737]
 [194.40712]
 [147.58917]
 [170.98463]
 [171.9268 ]
 [152.2488 ]
 [172.06635]
 [180.33159]
 [162.01222]
 [141.76445]
 [194.76726]]

Step : 1400
Cost : 52.228336334228516
Prediction :
[[144.57219 ]
 [191.33026 ]
 [179.27138 ]
 [196.9643  ]
 [148.98605 ]
 [114.79309 ]
 [145.25183 ]
 [104.053856]
 [185.11766 ]
 [177.0076  ]
 [143.41348 ]
 [149.899   ]
 [183.69926 ]
 [151.46988 ]
 [148.75433 ]
 [194.20053 ]
 [147.62054 ]
 [171.22198 ]
 [172.15285 ]
 [152.5026  ]
 [172.14038 ]
 [180.12366 ]
 [162.14534 ]
 [142.16772 ]
 [194.66635 ]]

Step : 1500
Cost : 49.338050842285156
Prediction :
[[144.85838 ]
 [191.13068 ]
 [179.35887 ]
 [197.01051 ]
 [148.73918 ]
 [114.47191 ]
 [145.3661  ]
 [104.266014]
 [184.74513 ]
 [176.50558 ]
 [143.41185 ]
 [149.63875 ]
 [183.82994 ]
 [151.61162 ]
 [148.7825  ]
 [194.0016  ]
 [147.6462  ]
```

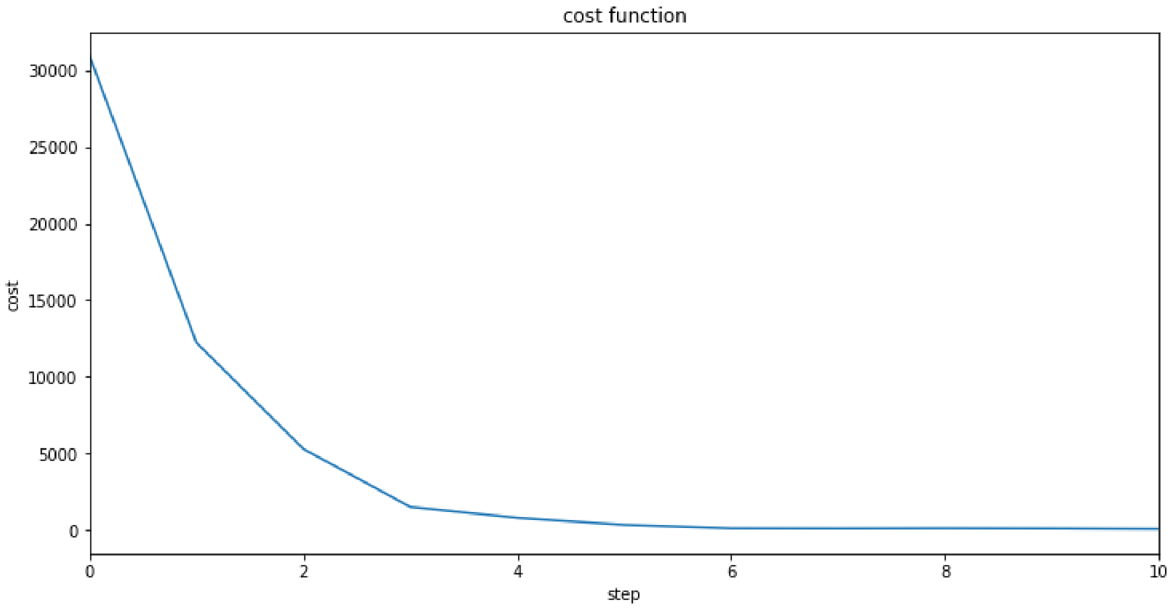
```
 [171.45424 ]
 [172.36891 ]
 [152.74588 ]
 [172.21379 ]
 [179.92348 ]
 [162.27534 ]
 [142.55357 ]
 [194.5677  ]]

Step : 1600
Cost : 46.643707275390625
Prediction :
[[145.13412 ]
 [190.93741 ]
 [179.44261 ]
 [197.05553 ]
 [148.49948 ]
 [114.162605]
 [145.47836 ]
 [104.4746  ]
 [184.38638 ]
 [176.02428 ]
 [143.41101 ]
 [149.38869 ]
 [183.95418 ]
 [151.74521 ]
 [148.8118  ]
 [193.81004 ]
 [147.66653 ]
 [171.68152 ]
 [172.57547 ]
 [152.97913 ]
 [172.28656 ]
 [179.7308  ]
 [162.40231 ]
 [142.92278 ]
 [194.47134 ]]

Step : 1700
Cost : 44.131011962890625
Prediction :
[[145.39977 ]
 [190.75023 ]
 [179.52275 ]
 [197.09929 ]
 [148.26671 ]
 [113.864716]
 [145.58862 ]
 [104.67967 ]
 [184.04082 ]
 [175.56282 ]
 [143.41089 ]
 [149.1484  ]
 [184.07227 ]
 [151.87105 ]
 [148.8421  ]
 [193.62553 ]
 [147.68185 ]
 [171.90388 ]
 [172.77293 ]
 [153.20276 ]
```

```
 [172.35861 ]
 [179.5453  ]
 [162.52632 ]
 [143.27605 ]
 [194.37714 ]]

Step : 1800
Cost : 41.7868766784668
Prediction :
[[145.65575]
 [190.56891]
 [179.59944]
 [197.1419 ]
 [148.04065]
 [113.57778]
 [145.6969 ]
 [104.8812 ]
 [183.70796]
 [175.12035]
 [143.41144]
 [148.91751]
 [184.18446]
 [151.98953]
 [148.8733 ]
 [193.44783]
 [147.69249]
 [172.12141]
 [172.96169]
 [153.41718]
 [172.42995]
 [179.36673]
 [162.64742]
 [143.61407]
 [194.28508]]

Step : 1900
Cost : 39.599151611328125
Prediction :
[[145.90242]
 [190.3933 ]
 [179.67285]
 [197.18335]
 [147.8211 ]
 [113.30139]
 [145.80322]
 [105.07925]
 [183.38731]
 [174.69609]
 [143.41263]
 [148.6956 ]
 [184.29108]
 [152.10101]
 [148.90535]
 [193.27667]
 [147.69873]
 [172.33423]
 [173.14212]
 [153.62277]
 [172.50055]
 [179.19476]
 [162.76567]
```

```
 [143.93748]
 [194.1951 ]]

Step : 2000
Cost : 37.55662536621094
Prediction :
[[146.14015]
 [190.22318]
 [179.7431 ]
 [197.2237 ]
 [147.60788]
 [113.03512]
 [145.90758]
 [105.27379]
 [183.0784 ]
 [174.28926]
 [143.4144 ]
 [148.48235]
 [184.39235]
 [152.20587]
 [148.93811]
 [193.11177]
 [147.70088]
 [172.5424 ]
 [173.31464]
 [153.81992]
 [172.57037]
 [179.02916]
 [162.88115]
 [144.24698]
 [194.10721]]
```

In [20]:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = [12,6]

# Show the cost function
plt.plot(Step_val, Cost_val)
plt.title('cost function')
plt.xlabel('step')
plt.ylabel('cost')
plt.xlim(0,10)
plt.show()
```



In [ ]:

```

```

In [12]:

```
import tensorflow as tf

tf.set_random_seed(777)  # for reproducibility

filename_queue = tf.train.string_input_producer(
    ['./data/data-01-test-score.csv'], shuffle=False, name='filename_queue')
#shuffle=False을 해줌으로써 수들이 섞이지 않게


reader = tf.TextLineReader()
key, value = reader.read(filename_queue)

# Default values, in case of empty columns. Also specifies the type of the decoded result.
# Convert CSV records to tensors. Each column maps to one tensor.
record_defaults = [[0.], [0.], [0.], [0.]]
xy = tf.decode_csv(value, record_defaults=record_defaults)

# collect batches of csv in
train_x_batch, train_y_batch = ₩
    tf.train.batch([xy[0:-1], xy[-1:]], batch_size=10)

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([3, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis
hypothesis = tf.matmul(X, W) + b

# Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)
```

```
WARNING: Logging before flag parsing goes to stderr.
W0911 11:03:52.211034  9272 deprecation.py:323] From <ipython-input-12-4a1d47d98880
>:6: string_input_producer (from tensorflow.python.training.input) is deprecated and
will be removed in a future version.
Instructions for updating:
Queue-based input pipelines have been replaced by `tf.data`. Use `tf.data.Dataset.fr
om_tensor_slices(string_tensor).shuffle(tf.shape(input_tensor, out_type=tf.int64)
[0]).repeat(num_epochs)`. If `shuffle=False`, omit the `.shuffle(...)`.
W0911 11:03:52.222724  9272 deprecation.py:323] From C:₩Users₩202-006₩Anaconda3₩lib
₩site-packages₩tensorflow₩python₩training₩input.py:278: input_producer (from tensorf
low.python.training.input) is deprecated and will be removed in a future version.
Instructions for updating:
Queue-based input pipelines have been replaced by `tf.data`. Use `tf.data.Dataset.fr
om_tensor_slices(input_tensor).shuffle(tf.shape(input_tensor, out_type=tf.int64)
[0]).repeat(num_epochs)`. If `shuffle=False`, omit the `.shuffle(...)`.
W0911 11:03:52.224666  9272 deprecation.py:323] From C:₩Users₩202-006₩Anaconda3₩lib
₩site-packages₩tensorflow₩python₩training₩input.py:190: limit_epochs (from tensorflo
w.python.training.input) is deprecated and will be removed in a future version.
Instructions for updating:
Queue-based input pipelines have been replaced by `tf.data`. Use `tf.data.Dataset.fr
```

```
om_tensors(tensor).repeat(num_epochs)`.
W0911 11:03:52.229549  9272 deprecation.py:323] From C:₩Users₩202-006₩Anaconda3₩lib
₩site-packages₩tensorflow₩python₩training₩input.py:199: QueueRunner.__init__ (from t
ensorflow.python.training.queue_runner_impl) is deprecated and will be removed in a
future version.
Instructions for updating:
To construct input pipelines, use the `tf.data` module.
W0911 11:03:52.233456  9272 deprecation.py:323] From C:₩Users₩202-006₩Anaconda3₩lib
₩site-packages₩tensorflow₩python₩training₩input.py:199: add_queue_runner (from tenso
rflow.python.training.queue_runner_impl) is deprecated and will be removed in a futu
re version.
Instructions for updating:
To construct input pipelines, use the `tf.data` module.
W0911 11:03:52.240292  9272 deprecation.py:323] From <ipython-input-12-4a1d47d98880
>:8: TextLineReader.__init__ (from tensorflow.python.ops.io_ops) is deprecated and w
ill be removed in a future version.
Instructions for updating:
Queue-based input pipelines have been replaced by `tf.data`. Use `tf.data.TextLineDa
taset`.
W0911 11:03:52.250058  9272 deprecation.py:323] From <ipython-input-12-4a1d47d98880
>:18: batch (from tensorflow.python.training.input) is deprecated and will be remove
d in a future version.
Instructions for updating:
Queue-based input pipelines have been replaced by `tf.data`. Use `tf.data.Dataset.ba
tch(batch_size)` (or `padded_batch(...)` if `dynamic_pad=True`).
```

In [13]:

```
# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

# Start populating the filename queue.
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(sess=sess, coord=coord)

Step_val = []
Cost_val = []

for step in tqdm_notebook(range(2001)):

    x_batch, y_batch = sess.run([train_x_batch, train_y_batch])
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train], feed_dict={X: x_batch, Y: y_batch})


    Step_val.append(step)
    Cost_val.append(cost_val)

    if step % 100 == 0 or step < 10 :
        print("₩nStep : {} ₩nCost : {} ₩nPrediction :₩n{}".format(step, cost_val, hy_val))

coord.request_stop()
coord.join(threads)
```

```
W0911 11:04:44.198271  9272 deprecation.py:323] From <ipython-input-13-a0a2b752713a
>:8: start_queue_runners (from tensorflow.python.training.queue_runner_impl) is depr
ecated and will be removed in a future version.
Instructions for updating:
To construct input pipelines, use the `tf.data` module.

HBox(children=(IntProgress(value=0, max=2001), HTML(value='')))


Step : 0
Cost : 30978.400390625
Prediction :
[[-10.5520935]
 [-18.421577 ]
 [-14.914436 ]
 [-19.039238 ]
 [-13.2786255]
 [-14.993626 ]
 [-16.968872 ]
 [-16.157917 ]
 [-22.078262 ]
 [-27.293266 ]]

Step : 1
Cost : 12231.3759765625
Prediction :
[[43.16722 ]
 [38.92281 ]
 [64.13846 ]
 [56.88572 ]
 [42.62716 ]
```

```
 [55.684196]
 [55.79019 ]
 [51.981804]
 [62.46227 ]
 [55.94705 ]]

Step : 2
Cost : 5287.12353515625
Prediction :
[[ 97.59089 ]
 [ 97.0528  ]
 [ 94.32073 ]
 [100.271095]
 [112.16209 ]
 [ 92.96686 ]
 [106.03886 ]
 [107.69688 ]
 [114.504074]
 [ 81.644264]]

Step : 3
Cost : 1493.1212158203125
Prediction :
[[ 74.055824]
 [110.207405]
 [ 78.125465]
 [125.918816]
 [109.81078 ]
 [107.33985 ]
 [102.183716]
 [148.57176 ]
 [127.08399 ]
 [109.53553 ]]

Step : 4
Cost : 783.9133911132812
Prediction :
[[155.31958]
 [134.9293 ]
 [146.63405]
 [157.76044]
 [141.21684]
 [143.87987]
 [143.15866]
 [138.45131]
 [141.47476]
 [163.08902]]

Step : 5
Cost : 321.26654052734375
Prediction :
[[144.10187 ]
 [167.50797 ]
 [168.25812 ]
 [180.46432 ]
 [128.52313 ]
 [ 89.579636]
 [132.37454 ]
 [ 94.56593 ]
 [151.71573 ]
 [133.71931 ]]
```

```
Step : 6
Cost : 93.63517761230469
Prediction :
[[134.21555]
 [128.69669]
 [183.90215]
 [156.44302]
 [137.56572]
 [175.07072]
 [150.60315]
 [165.39822]
 [176.6404 ]
 [158.10988]]

Step : 7
Cost : 84.15323638916016
Prediction :
[[164.65642]
 [163.85036]
 [158.25836]
 [159.95824]
 [185.94057]
 [152.1854 ]
 [177.23401]
 [177.83566]
 [190.89893]
 [135.94019]]

Step : 8
Cost : 97.02232360839844
Prediction :
[[ 96.57832 ]
 [142.3618  ]
 [101.974365]
 [163.34325 ]
 [144.50099 ]
 [138.31552 ]
 [132.7418  ]
 [189.28987 ]
 [160.91902 ]
 [141.84273 ]]

Step : 9
Cost : 84.88079833984375
Prediction :
[[182.67336]
 [156.62245]
 [172.6211 ]
 [183.89616]
 [164.60182]
 [169.27106]
 [168.44783]
 [162.65637]
 [164.0521 ]
 [191.01373]]

Step : 100
Cost : 46.08684539794922
Prediction :
[[157.71341]
```

```
 [184.23744]
 [184.53821]
 [198.35587]
 [141.24089]
 [ 99.24805]
 [145.9032 ]
 [104.77473]
 [167.63348]
 [148.85269]]

Step : 200
Cost : 42.22480010986328
Prediction :
[[157.48172 ]
 [184.32605 ]
 [184.42302 ]
 [198.39262 ]
 [141.26588 ]
 [ 99.5597  ]
 [146.069   ]
 [105.080086]
 [167.99965 ]
 [149.58324 ]]

Step : 300
Cost : 38.713993072509766
Prediction :
[[157.26135]
 [184.40945]
 [184.31299]
 [198.4281 ]
 [141.2879 ]
 [ 99.85633]
 [146.22884]
 [105.37463]
 [168.34818]
 [150.28056]]

Step : 400
Cost : 35.52268600463867
Prediction :
[[157.05174]
 [184.4879 ]
 [184.20782]
 [198.4623 ]
 [141.30714]
 [100.13865]
 [146.38297]
 [105.65879]
 [168.67989]
 [150.94615]]

Step : 500
Cost : 32.622215270996094
Prediction :
[[156.85242]
 [184.56166]
 [184.10733]
 [198.4953 ]
 [141.32378]
 [100.40733]
```

```
 [146.53159]
 [105.93296]
 [168.99554]
 [151.58142]]

Step : 600
Cost : 29.986352920532227
Prediction :
[[156.66289 ]
 [184.631   ]
 [184.01132 ]
 [198.52711 ]
 [141.33794 ]
 [100.662994]
 [146.67491 ]
 [106.19752 ]
 [169.29593 ]
 [152.18773 ]]

Step : 700
Cost : 27.591114044189453
Prediction :
[[156.4827 ]
 [184.69609]
 [183.91959]
 [198.55785]
 [141.34982]
 [100.90624]
 [146.8132 ]
 [106.45283]
 [169.58173]
 [152.76643]]

Step : 800
Cost : 25.414945602416992
Prediction :
[[156.31139 ]
 [184.75716 ]
 [183.83191 ]
 [198.58748 ]
 [141.35953 ]
 [101.13767 ]
 [146.94655 ]
 [106.699234]
 [169.8536  ]
 [153.31871 ]]

Step : 900
Cost : 23.43788719177246
Prediction :
[[156.14856]
 [184.81442]
 [183.74817]
 [198.61607]
 [141.36722]
 [101.35783]
 [147.07523]
 [106.93707]
 [170.11226]
 [153.84583]]
```

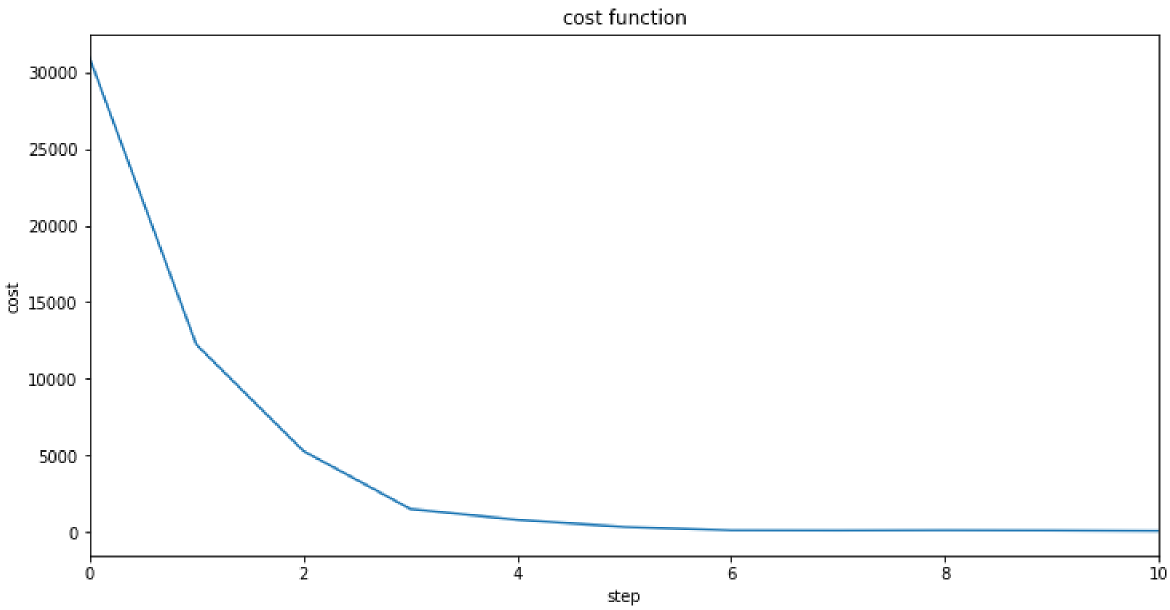
```
Step : 1000
Cost : 21.64202308654785
Prediction :
[[155.99379]
 [184.8681 ]
 [183.66809]
 [198.64366]
 [141.37299]
 [101.56723]
 [147.19936]
 [107.16668]
 [170.35826]
 [154.34885]]

Step : 1100
Cost : 20.01089859008789
Prediction :
[[155.84674]
 [184.91835]
 [183.59161]
 [198.67032]
 [141.37703]
 [101.76639]
 [147.31914]
 [107.38835]
 [170.59225]
 [154.82893]]

Step : 1200
Cost : 18.529691696166992
Prediction :
[[155.70702 ]
 [184.96535 ]
 [183.51852 ]
 [198.69604 ]
 [141.37938 ]
 [101.95579 ]
 [147.43474 ]
 [107.602394]
 [170.81473 ]
 [155.28706 ]]

Step : 1300
Cost : 17.184703826904297
Prediction :
[[155.57428]
 [185.00931]
 [183.44868]
 [198.72089]
 [141.38022]
 [102.13587]
 [147.54633]
 [107.80908]
 [171.02629]
 [155.72427]]

Step : 1400
Cost : 15.963668823242188
Prediction :
[[155.44821 ]
 [185.05032 ]
```

```
 [183.38194 ]
 [198.74487 ]
 [141.37961 ]
 [102.307076]
 [147.65402 ]
 [108.00869 ]
 [171.22742 ]
 [156.14148 ]]

Step : 1500
Cost : 14.855279922485352
Prediction :
[[155.32846 ]
 [185.08858 ]
 [183.31816 ]
 [198.768   ]
 [141.37764 ]
 [102.46983 ]
 [147.75798 ]
 [108.201485]
 [171.4186  ]
 [156.5396  ]]

Step : 1600
Cost : 13.849319458007812
Prediction :
[[155.21475 ]
 [185.12424 ]
 [183.25722 ]
 [198.79036 ]
 [141.37442 ]
 [102.62454 ]
 [147.85835 ]
 [108.387726]
 [171.60033 ]
 [156.91948 ]]

Step : 1700
Cost : 12.936464309692383
Prediction :
[[155.10681]
 [185.15741]
 [183.19896]
 [198.81197]
 [141.37006]
 [102.77156]
 [147.95526]
 [108.56764]
 [171.77303]
 [157.28198]]

Step : 1800
Cost : 12.10825252532959
Prediction :
[[155.00435]
 [185.18828]
 [183.14334]
 [198.83284]
 [141.36464]
 [102.91127]
 [148.04887]
```

```
 [108.74147]
 [171.93715]
 [157.62788]]

Step : 1900
Cost : 11.356977462768555
Prediction :
[[154.90709 ]
 [185.21689 ]
 [183.09015 ]
 [198.85301 ]
 [141.3582  ]
 [103.044014]
 [148.13925 ]
 [108.90945 ]
 [172.09306 ]
 [157.9579  ]]

Step : 2000
Cost : 10.675619125366211
Prediction :
[[154.81482]
 [185.24344]
 [183.03932]
 [198.8725 ]
 [141.35086]
 [103.17011]
 [148.22656]
 [109.07177]
 [172.24118]
 [158.27278]]
```

In [14]:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = [12,6]

# Show the cost function
plt.plot(Step_val, Cost_val)
plt.title('cost function')
plt.xlabel('step')
plt.ylabel('cost')
plt.xlim(0,10)
plt.show()
```



학습을시킨후

prediction

In [16]:

```
# Ask score
print("Your score ₩t: ₩n", sess.run(hypothesis, feed_dict={X: [[100, 70, 101]]}))
```

```
Your score      :
 [[180.75392]]
```

In [17]:

```
# Ask score many
print("₩nOther scores ₩t: ₩n", sess.run(hypothesis,
                                        feed_dict={X: [[60, 70, 110], [90, 100, 80]]}))
```

```
Other scores    :
 [[155.70262]
 [185.67659]]
```