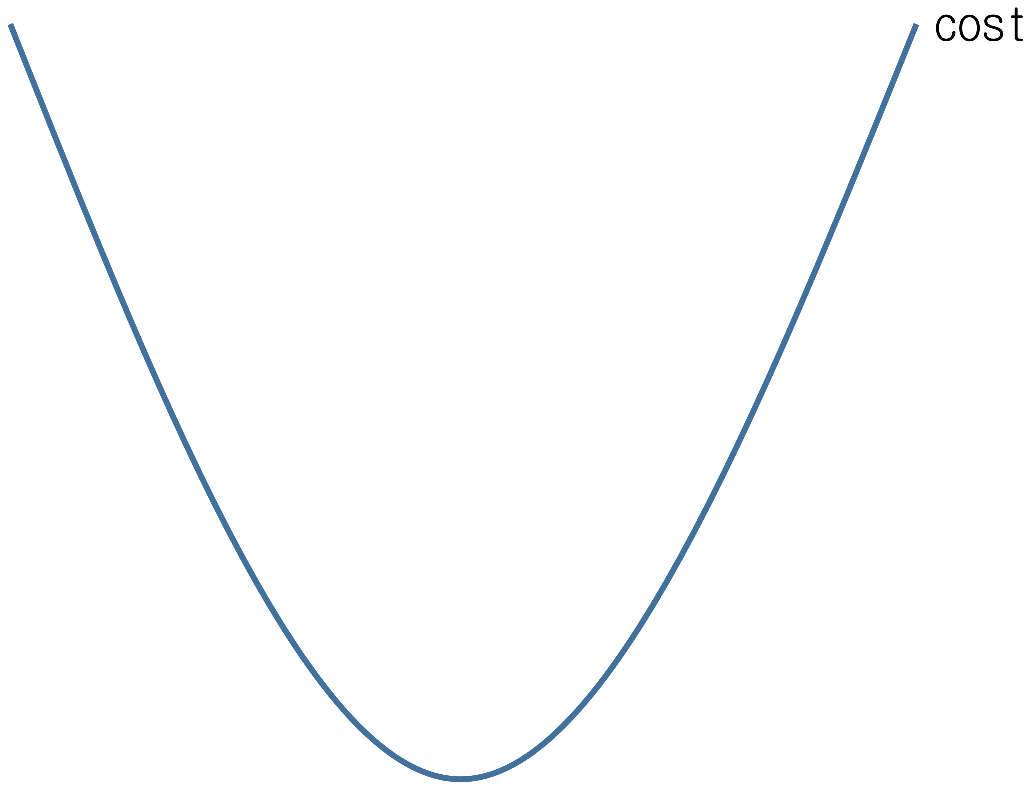
딥러닝 구현을 위한 텐서플로우 개발

김성균

5강
머신러닝 실용과 MNIST 데이터셋

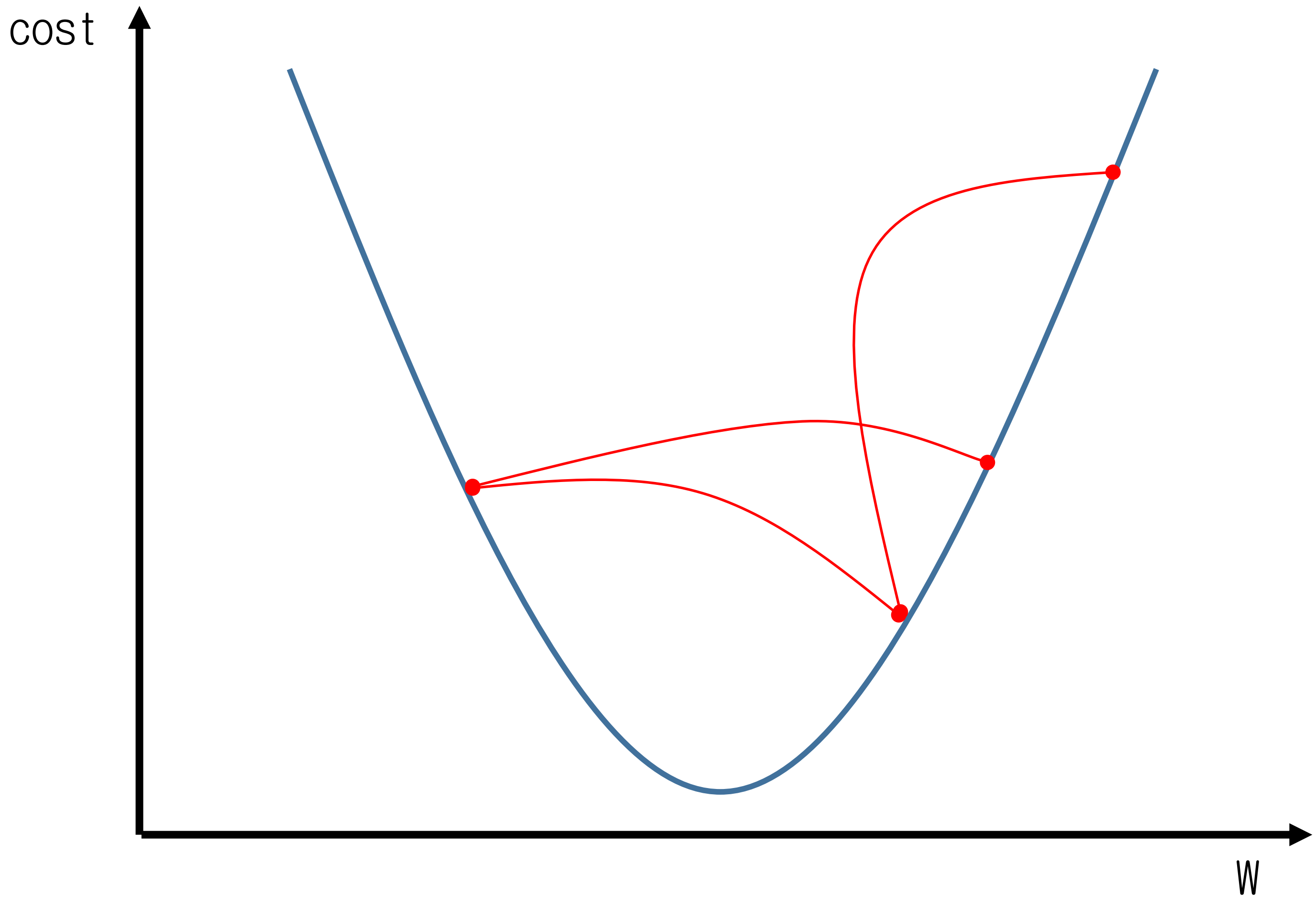
01.Learning rate

```
LEARNING_RATE = 0.01

# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate = LEARNING_RATE).minimize(cost)
```



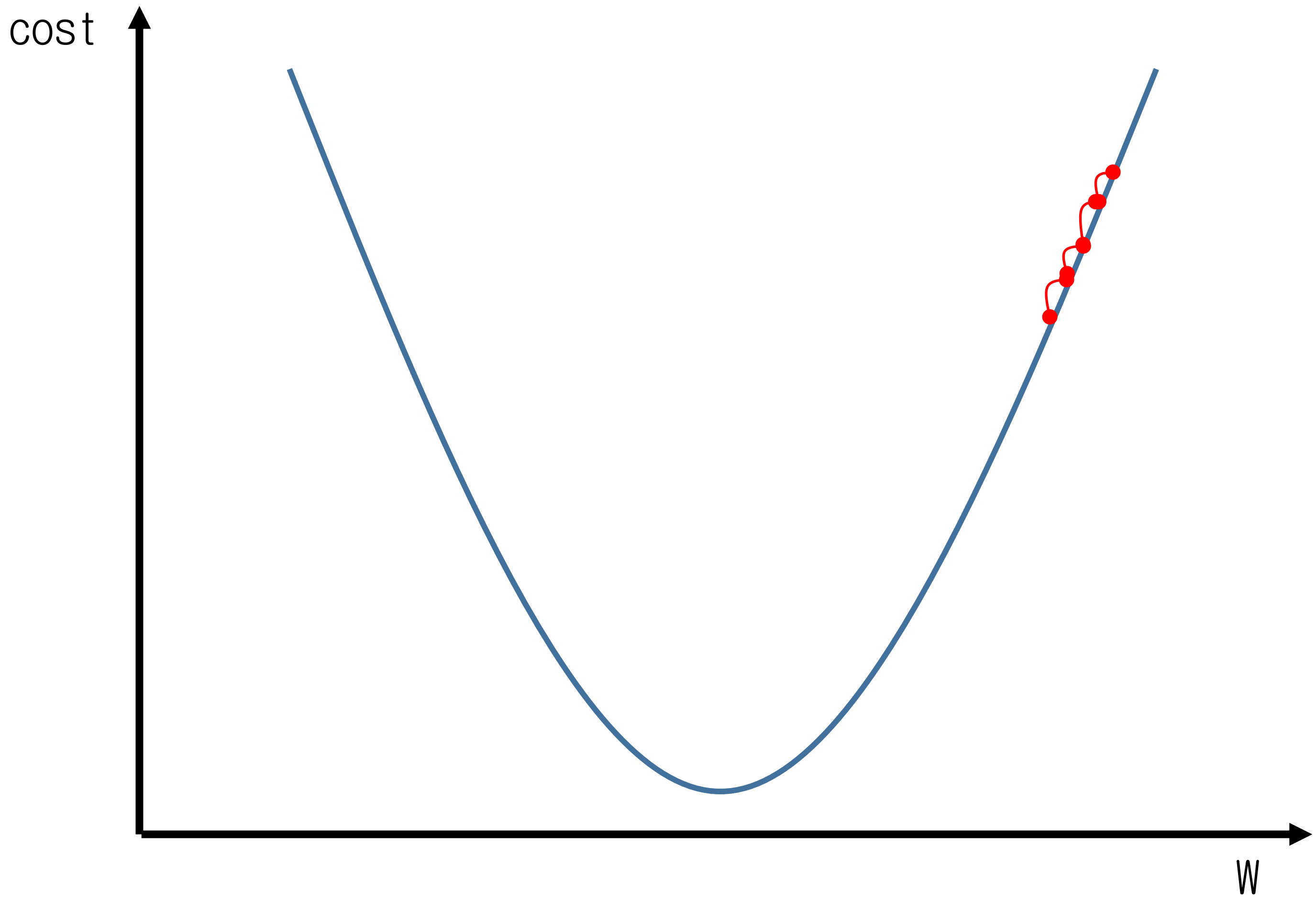
02. Large learning rate: overshooting

- cost가 증가하는 overshooting 현상



03. Small learning rate: takes too long, stops at local minimum

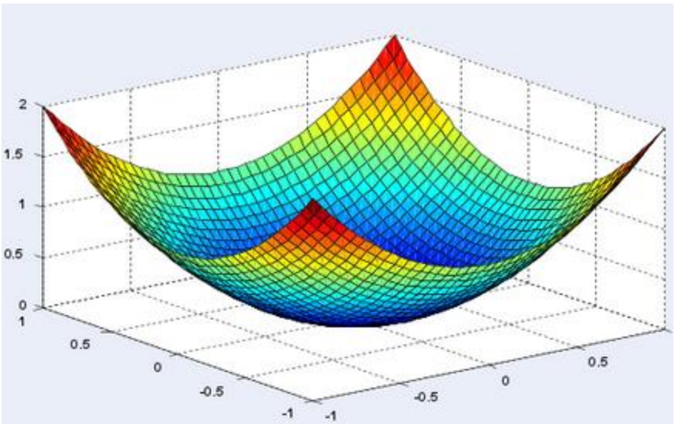
- 너무 조금씩 감소하는 현상

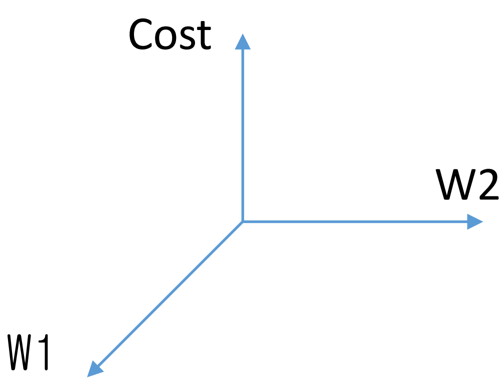


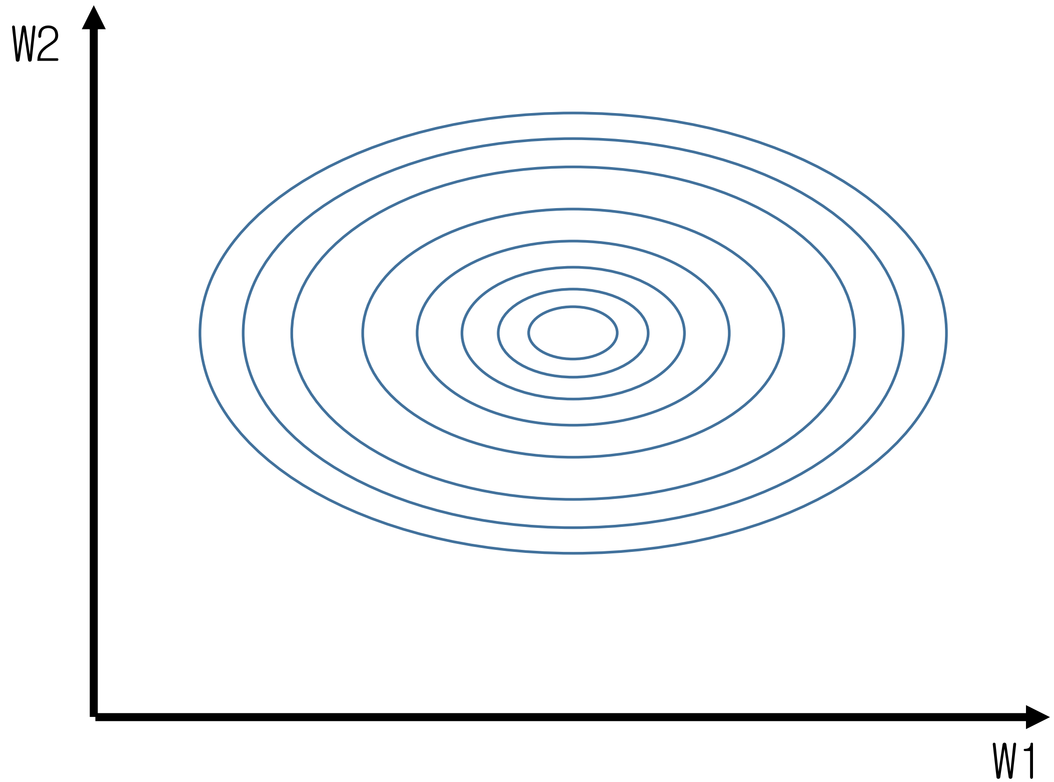
04. Try several learning rates

- cost function관찰
- 경사도 하강속도 확인
- 다양한 learning rate을 사용해서 여러 번에 걸쳐 실행하는 것이 최선

05.경사도 하강을 위한 전처리(preprocessing)

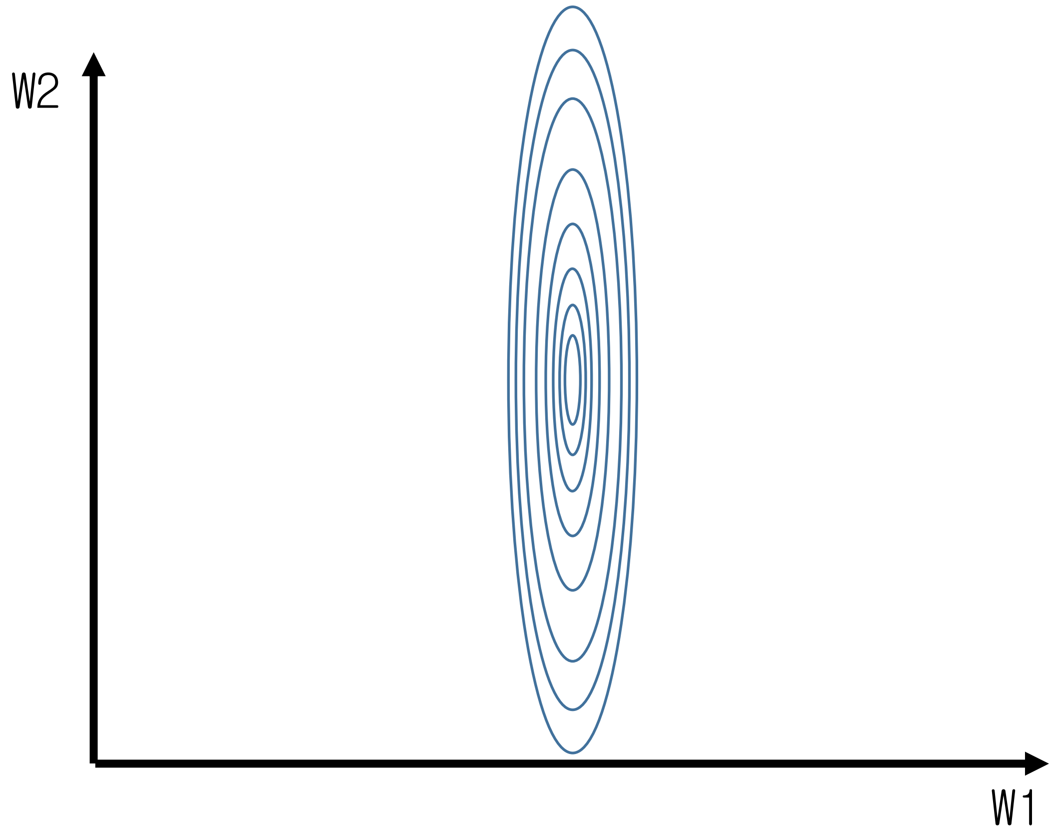




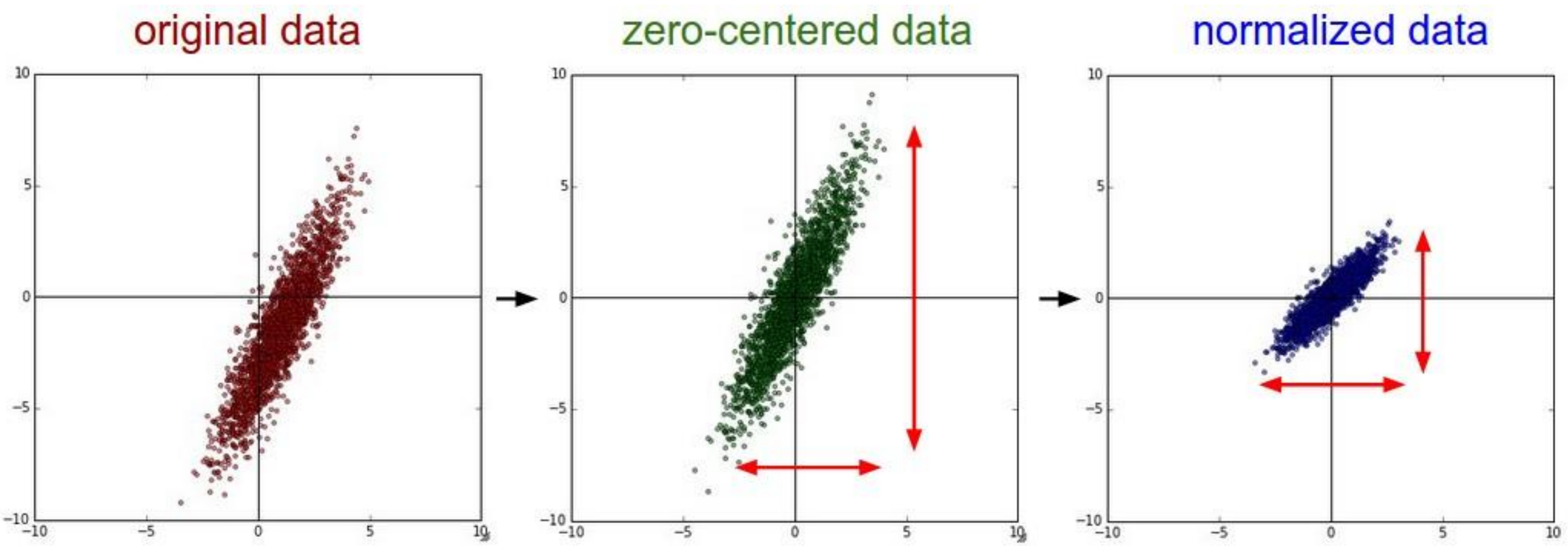


05.경사도 하강을 위한 전처리(preprocessing)

x1	x2	Y
1	9000	A
2	-5000	A
4	-2000	B
6	8000	B
9	9000	C



05.경사도 하강을 위한 전처리(preprocessing)



06.표준화

- Normalization :
  - 해당값을 0~1사이의 값으로 나타내는 척도법

$$x_j' = \frac{x_j - x_{min}}{x_{max} - x_{min}}$$

- Standardization:
  - 속성값이 편균으로부터의 위치를 표준편차다위로 표현

$$x_j' = \frac{x_j - \mu}{\sigma}$$

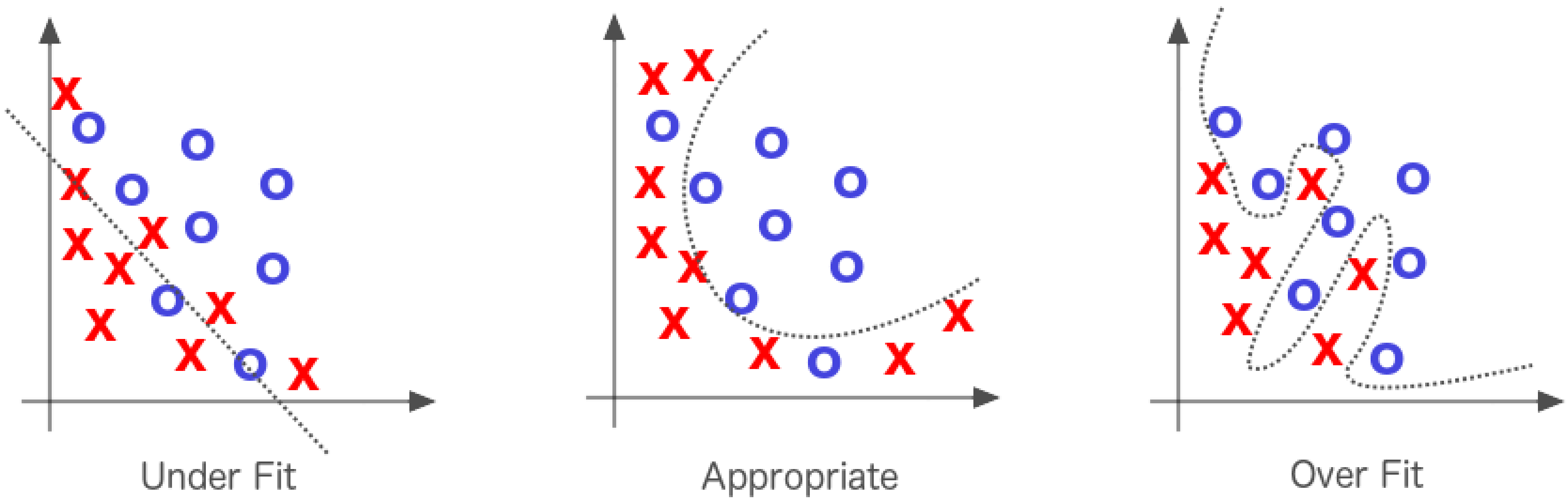
μ_j:평균

σ_j: 표준편차

```
X_std[:,0] = ( X[:,0] – X[:,0].mean() ) / X[:,0].std()
```

07. Overfitting

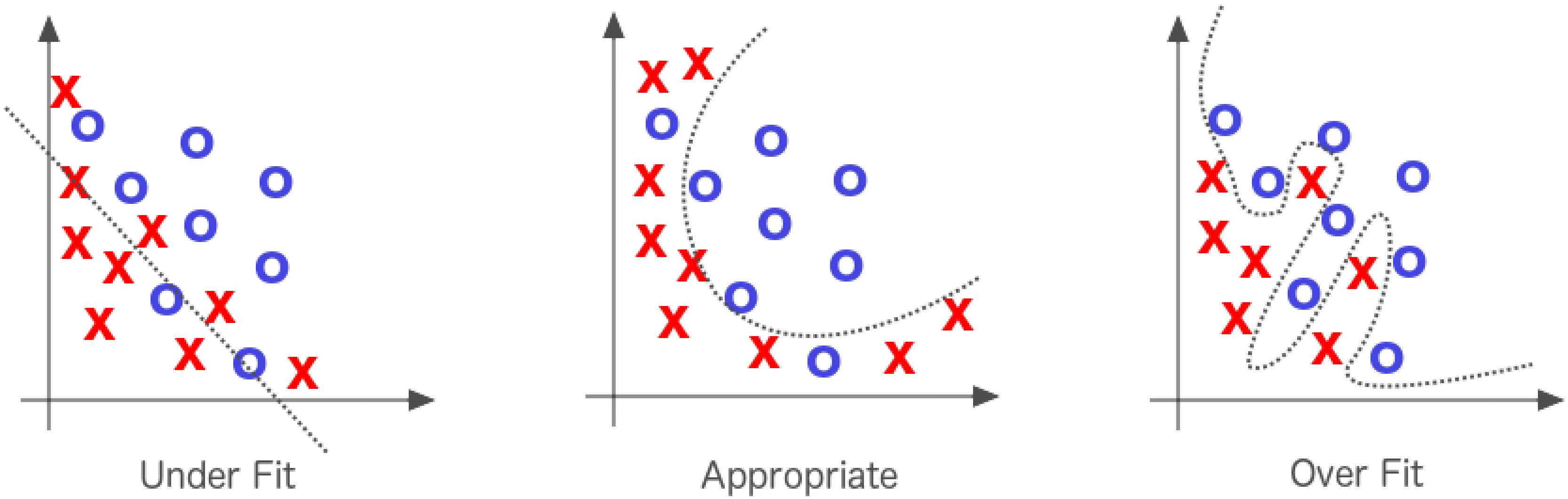
- 모델이 학습데이터에 너무 잘 맞아서 학습데이터에만 최적화된 경우
- 정말 잘 맞추기 위해 과도하게 복잡해져서 실제로 사용할 때는 오히려 맞지 않는 현상



학습이 덜되어
error가 많이 발생함

학습데이터에 최적화
되어 error이 발생하지
않음

07. Overfitting

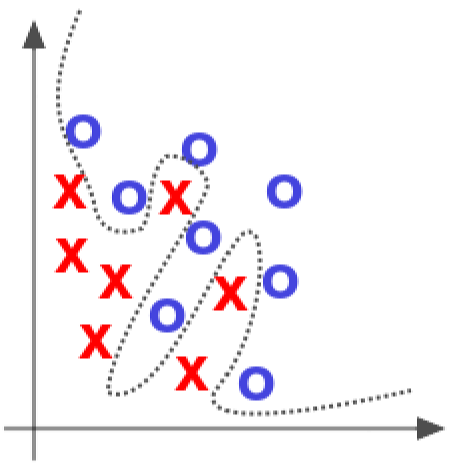


08. Overfitting의 해결방법

- 많은 학습데이터
- 입력으로 들어오는 변수(feature, x)의 갯수를 줄이기
- 정규화(Regularization)을 사용

09. 정규화(Regularization)

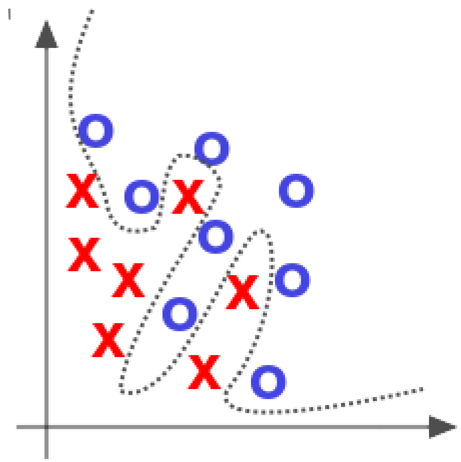
- W(weight)가 너무 큰 값들을 갖지 않도록 하는 것
- 비정상적이거나 애매한 위치에 있는 데이터를 올바르게 예측하는 것을 '맞았다'라고 얘기할 수는 없음
- 오히려 '틀렸다'라고 얘기하는 것이 더욱 좋을 수 있음
- 최소한의 에러를 인정하는 'Just right'



Over Fit

09. 정규화(Regularization)

- W(weight)가 너무 큰 값들을 갖지 않도록 하는 것
- 비정상적이거나 애매한 위치에 있는 데이터를 올바르게 예측하는 것을 '맞았다'라고 얘기할 수는 없음
- 오히려 '틀렸다'라고 얘기하는 것이 더욱 좋을 수 있음
- 최소한의 에러를 인정하는 'Just right‘



Over Fit

- W(weight)가 너무 큰 값들을 갖지 않도록 하는 것
- cost 함수가 틀렸을 때 높은 비용이 발생할 수 있도록 벌점(penalty)을 부과

```
L2reg = 0.001 * tf.reduce_sum(tf.square(W))
```

$$Cost = \frac{1}{N}\sum_i \underbrace{D(\underbrace{S(WX_i + b)}_{\text{예측값}}, L_i)}_{\text{실제값과의 차}} + \lambda \sum W^2$$

X_i, L_i 입력값

W, b 학습해야할 값

10. Training and Test datasets

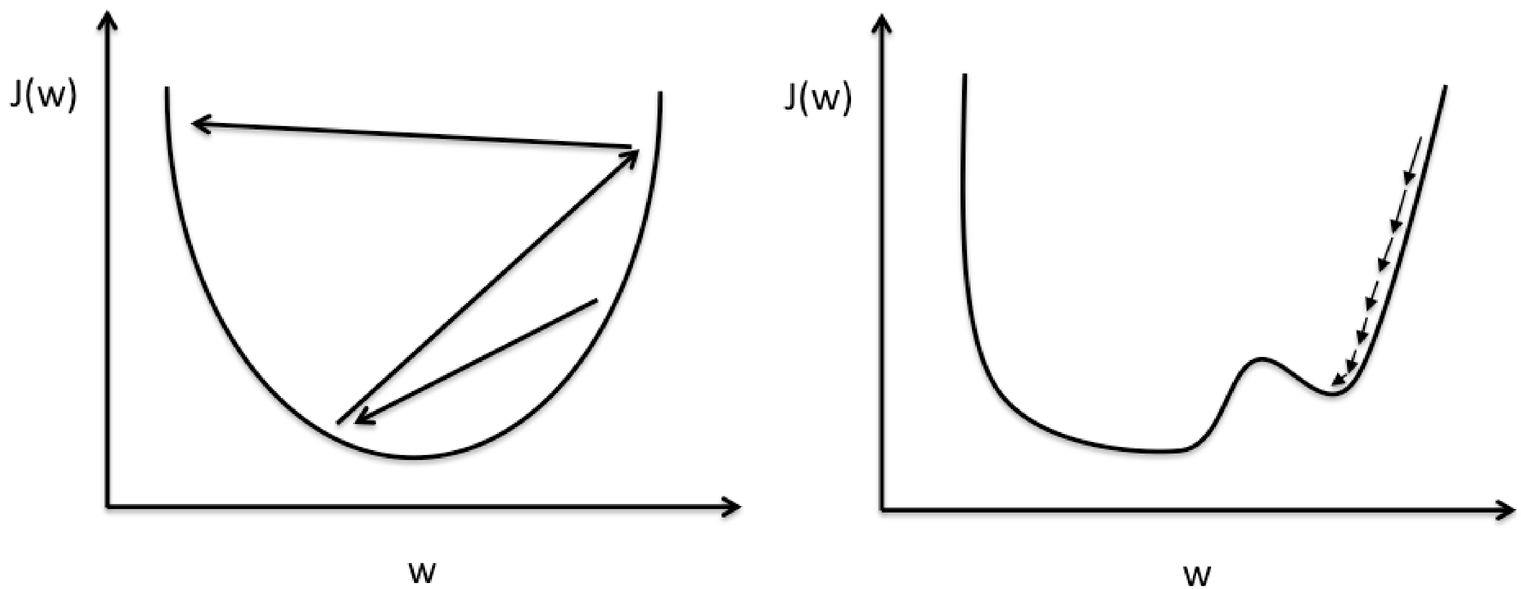
- 학습용 데이터 세트와 테스트용 데이터 세트 구분

```python
x_data = [[1, 2, 1], [1, 3, 2], [1, 3, 4], [1, 5, 5], [1, 7, 5], [1, 2, 5], [1, 6, 6], [1, 7, 7]]
y_data = [[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 1, 0], [0, 1, 0], [1, 0, 0], [1, 0, 0]]
# Evaluation our model using this test dataset
x_test = [[2, 1, 1], [3, 1, 2], [3, 3, 4]]
y_test = [[0, 0, 1], [0, 0, 1], [0, 0, 1]]
```

```python
X = tf.placeholder("float", [None, 3])
Y = tf.placeholder("float", [None, 3])

W = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.random_normal([3]))

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
# Try to change learning_rate to small numbers
optimizer = tf.train.GradientDescentOptimizer(
    learning_rate=LEARNING_RATE).minimize(cost)

# Correct prediction Test model
prediction = tf.arg_max(hypothesis, 1)
is_correct = tf.equal(prediction, tf.arg_max(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(201):
        cost_val, W_val, _ = sess.run(
            [cost, W, optimizer], feed_dict={X: x_data, Y: y_data})
        print(step, cost_val, W_val)

    # predict
    print("Prediction:", sess.run(prediction, feed_dict={X: x_test}))
    # Calculate the accuracy
    print("Accuracy: ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))
```

```
199 0.672261 [[-1.15377033  0.28146935  1.13632679]
 [ 0.37484586  0.18958236  0.33544877]
 [-0.35609841 -0.43973011 -1.25604188]]
200 0.670909 [[-1.15885413  0.28058422  1.14229572]
 [ 0.37609792  0.19073224  0.33304682]
 [-0.35536593 -0.44033223 -1.2561723 ]]
Prediction: [2 2 2]
Accuracy:  1.0
```

11.Learning rate: NaN!



Large learning rate: Overshooting.

Small learning rate: Many iterations until convergence and trapping in local minima.

http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

```python
X = tf.placeholder("float", [None, 3])
Y = tf.placeholder("float", [None, 3])

W = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.random_normal([3]))

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
# Try to change learning_rate to small numbers
optimizer = tf.train.GradientDescentOptimizer(
    learning_rate=LEARNING_RATE).minimize(cost)

# Correct prediction Test model
prediction = tf.arg_max(hypothesis, 1)
is_correct = tf.equal(prediction, tf.arg_max(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(201):
        cost_val, W_val, _ = sess.run(
            [cost, W, optimizer], feed_dict={X: x_data, Y: y_data})
        print(step, cost_val, W_val)

    # predict
    print("Prediction:", sess.run(prediction, feed_dict={X: x_test}))
    # Calculate the accuracy
    print("Accuracy: ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))
```

```
LEARNING_RATE = 1.5
```

```
199 nan [[ nan  nan  nan]
 [ nan  nan  nan]
 [ nan  nan  nan]]
200 nan [[ nan  nan  nan]
 [ nan  nan  nan]
 [ nan  nan  nan]]
Prediction: [0 0 0]
Accuracy:  0.0
```

Not-A-Number

```python
X = tf.placeholder("float", [None, 3])
Y = tf.placeholder("float", [None, 3])

W = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.random_normal([3]))

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
# Try to change learning_rate to small numbers
optimizer = tf.train.GradientDescentOptimizer(
    learning_rate=LEARNING_RATE).minimize(cost)

# Correct prediction Test model
prediction = tf.arg_max(hypothesis, 1)
is_correct = tf.equal(prediction, tf.arg_max(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(201):
        cost_val, W_val, _ = sess.run(
            [cost, W, optimizer], feed_dict={X: x_data, Y: y_data})
        print(step, cost_val, W_val)

    # predict
    print("Prediction:", sess.run(prediction, feed_dict={X: x_test}))
    # Calculate the accuracy
    print("Accuracy: ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))
```

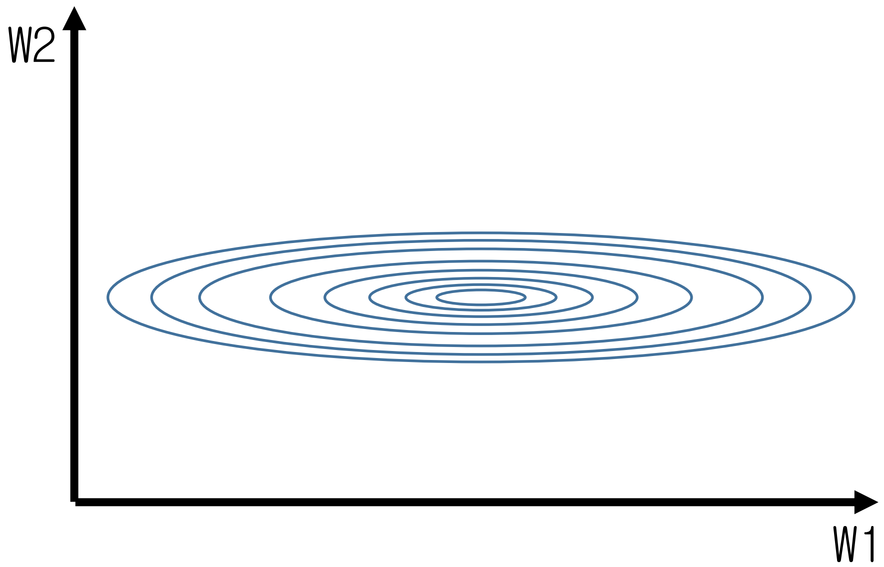
```python
LEARNING_RATE = 1e-10
```

```
198 5.73203 [[ 0.80269563  0.67861295 -1.21728313]
 [-0.3051686  -0.3032113   1.50825703]
 [ 0.75722361 -0.7008909  -2.10820389]]
199 5.73203 [[ 0.80269563  0.67861295 -1.21728313]
 [-0.3051686  -0.3032113   1.50825703]
 [ 0.75722361 -0.7008909  -2.10820389]]
200 5.73203 [[ 0.80269563  0.67861295 -1.21728313]
 [-0.3051686  -0.3032113   1.50825703]
 [ 0.75722361 -0.7008909  -2.10820389]]
Prediction: [0 0 0]
Accuracy:  0.0
```

전혀 학습이 되지 않음

12. Non-normalized inputs

```
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```



```python
xy = …
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 4])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([4, 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)

# 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(101):
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
    print(step, "Cost: ", cost_val, "₩nPrediction:₩n", hy_val)
```

```
100 Cost:  nan
Prediction:
 [[ nan]
 [ nan]
 [ nan]
 [ nan]
 [ nan]
 [ nan]
 [ nan]
 [ nan]]
```

13.Normalized inputs (min-max scale)

```python
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```

```python
xy = MinMaxScaler(xy)
print(xy)
```

$$x'_j = \frac{x_j - x_{min}}{x_{max} - x_{min}}$$

```
[[ 0.99999999  0.99999999  0.          1.          1.        ]
 [ 0.70548491  0.70439552  1.          0.71881782  0.83755791]
 [ 0.54412549  0.50274824  0.57608696  0.606468    0.6606331 ]
 [ 0.33890353  0.31368023  0.10869565  0.45989134  0.43800918]
 [ 0.51436     0.42582389  0.30434783  0.58504805  0.42624401]
 [ 0.49556179  0.42582389  0.31521739  0.48131134  0.49276137]
 [ 0.11436064  0.          0.20652174  0.22007776  0.18597238]
 [ 0.          0.07747099  0.5326087   0.          0.        ]]
```

```python
def MinMaxScaler(data):
    numerator = data - np.min(data, 0)
    denominator = np.max(data, 0) - np.min(data, 0)
    # noise term prevents the zero division
    return numerator / (denominator + 1e-7)
```

```python
xy = ...
xy = MinMaxScaler(xy)
print(xy)

x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 4])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([4, 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)

# 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(101):
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
    print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

```
100 Cost:  0.152254
Prediction:
 [[ 1.63450289]
 [ 0.06628087]
 [ 0.35014752]
 [ 0.67070574]
 [ 0.61131608]
 [ 0.61466062]
 [ 0.23175186]
 [-0.13716528]]
```