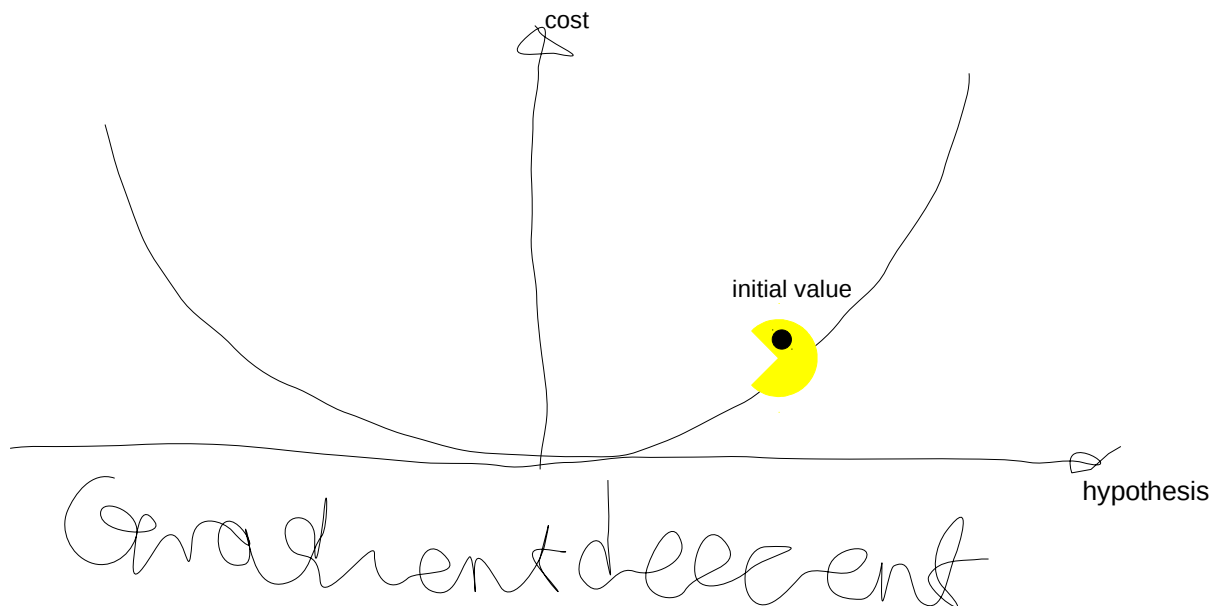


Lecture7

Rate & Overfitting

사전 점검 퀴즈



AI 모델을 만들던 김폴폴군은 운이 좋게도 cost 그래프가 $Y = X^2$ 과 정확하게 일치하는 것을 발견했다. 그래프의 X 축인 hypothesis 의 초기 랜덤값이 1로 주어졌을 때, 다음 보기 중 learning rate 로 가장 바람직한 것은 ?

1. 클 수록 좋으니까 1860
2. 작을 수록 좋다. $e^{(-80)}$
3. 적당히 0.5 쯤 ?

Lecture7

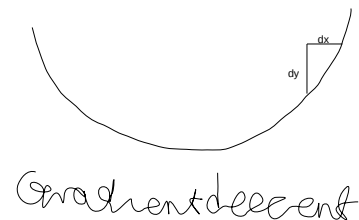
Rate & Overfitting

simple linear regression code

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

```
hypothesis = tf.matmul(X, W) + b  
cost = tf.reduce_mean(tf.square(hypothesis - Y))
```

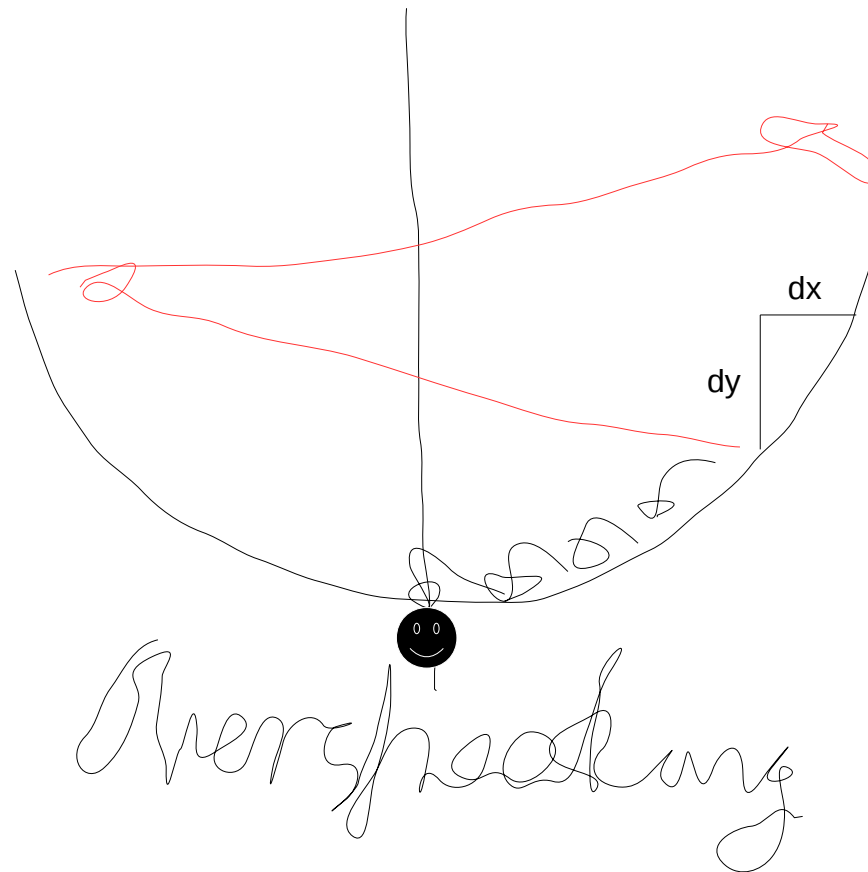
```
optimizer = tf.train.GradientDescentOptimizer(learning_rate = 0.01)  
train = optimizer.minimize(cost)
```



중요함

Lecture7

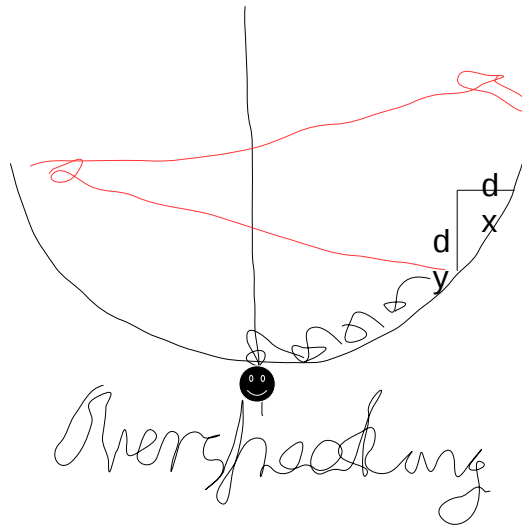
Rate & Overfitting



Learning rate 가 너무 커서 hypothesis - $\{d(\text{cost}) / d(\text{hypothesis}) * \text{learning rate}\}$ 가 한 점으로 수렴하지 않고 , 오히려 발산하는 현상

Lecture7

Rate & Overfitting



hypothesis - $\{d(\text{cost}) / d(\text{hypothesis}) * \text{learning rate}\}$ 가 한 점으로 수렴하지 않고 , 오히려 발산하는 현상 ← 왜 문제인가 ?

hypothesis' = hypothesis - $\{d(\text{cost}) / d(\text{hypothesis}) * \text{learning rate}\}$
cost' = f(hypothesis')

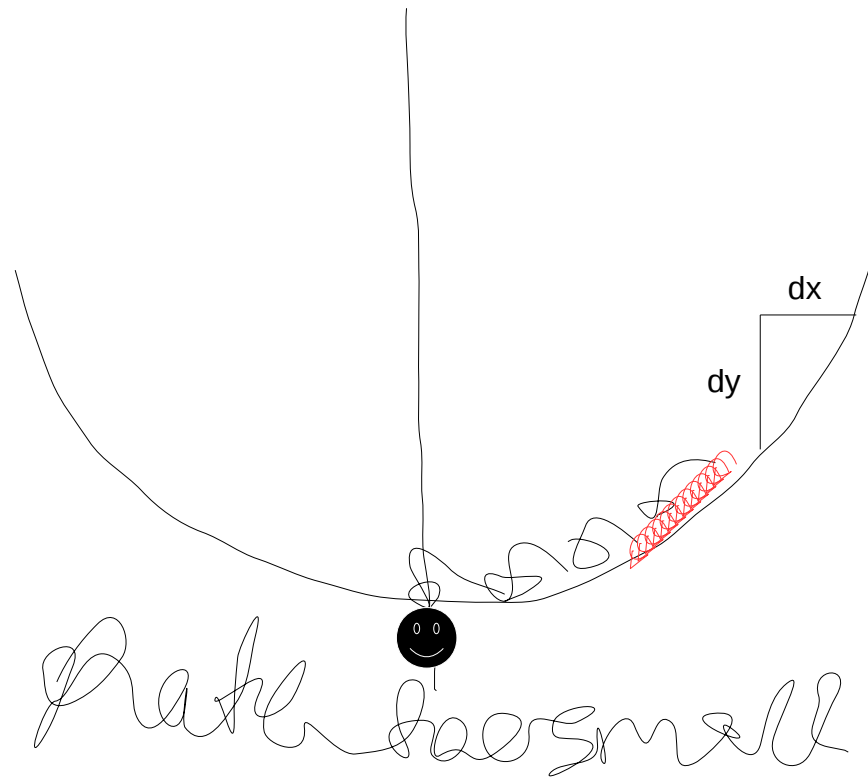
hypothesis' 가 발산한다면 , cost' 역시 발산하거나 부정확한 값이 나올 것이다 .

→ 이에 따라 수정되는 weight 와 bias 값 역시 부정확할 것이다 .

→ 정확한 모델을 만들 수 없어서 , classification 이던 regression 이던 불가능

Lecture7

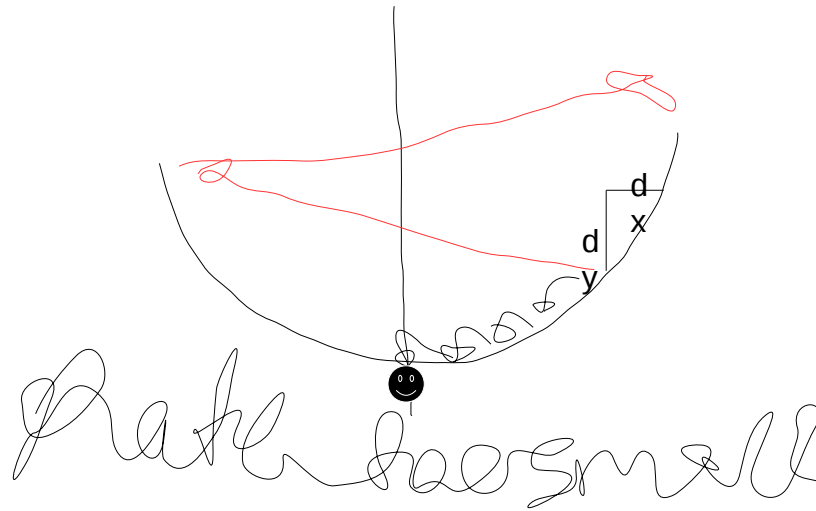
Rate & Overfitting



Learning rate 가 너무 작으 hypothesis - $\{d(\text{cost}) / d(\text{hypothesis}) * \text{learning rate}\}$ 이상적인 값까지 가지 못하는 현상이 발생

Lecture7

Rate & Overfitting



hypothesis - $\{d(\text{cost}) / d(\text{hypothesis}) * \text{learning rate}\}$ 가 이상적인 x 까지 가지 못하거나 너무 오래걸리는 현상 ← 왜 문제인가 ?

hypothesis' = hypothesis - $\{d(\text{cost}) / d(\text{hypothesis}) * \text{learning rate}\}$
cost' = f(hypothesis')

hypothesis' 가 이상값까지 가지 못 한다면 , cost' 역시 부정확한 값이 나올 것이다 .

→ 리소스의 낭비 (시간 짱 오래 걸림)

→ 완성되지 않은 모델로 인한 큰 오차값 (cost) 발생 , 부정확한 output

Lecture7

Rate & Overfitting



Learning Rate 실제로 문제가 되는가 ?

```
In [1]: 1 import tensorflow as tf
        2 import numpy as np
        3 import seaborn as sns
```

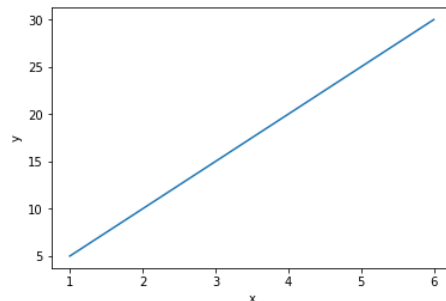
```
In [2]: 1 x = np.array([[1],[2],[3],[4],[5],[6]])
        2 y = np.array([[5],[10],[15],[20],[25],[30]])
        3
        4 x_test = np.array([[10],[20],[30],[40]])
        5
        6 X = tf.placeholder(tf.float32, [None, 1])
        7 Y = tf.placeholder(tf.float32, [None, 1])
        8
        9 W = tf.Variable(tf.random_normal([1, 1]), name = 'weight')
       10 b = tf.Variable(tf.random_normal([1]), name = 'bias')
       11
       12 hypothesis = tf.matmul(X, W) + b
       13 cost = tf.reduce_mean(tf.square(hypothesis - Y))
```

Y = 5*X 의 간단한 수식에 대한
linear regression 에서 learning rate
변화에 따른 output 변화 분석

WARNING:tensorflow:From /home/pirl/anaconda3/lib/python3.7/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

```
In [3]: 1 import pandas as pd
        2 test_tmp = pd.DataFrame({'x':[k[0] for k in x], 'y':[k[0] for k in y]})
        3 sns.lineplot(x=test_tmp['x'], y=test_tmp['y'])
```

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb254b060b8>



Lecture7

Rate & Overfitting

```
In [4]: 1 lr = 0.01
2
3 optimizer = tf.train.GradientDescentOptimizer(lr)
4 train = optimizer.minimize(cost)
5
6 with tf.Session() as sess:
7     sess.run(tf.global_variables_initializer())
8
9     for step in range(101):
10         _, cost_val = sess.run([train, cost], feed_dict = {X:x, Y:y})
11
12         if step % 10 == 0:
13             print(f'Step: {step} Cost: {cost_val}')
14
15     w_val, b_val = sess.run([W, b])
16
17     print('*80')
18     print('Report')
19     print('*80')
20     print(f'Linear Regression Formula: Y = {w_val}x + {b_val}')
21     print('*80')
22     print('*80')
23     exp_val1 = sess.run(hypothesis, feed_dict = {X:x_test})
24     print(exp_val1)
25
26
```

```
Step: 0 Cost: 209.36463928222656
Step: 10 Cost: 0.23946623504161835
Step: 20 Cost: 0.13491608202457428
Step: 30 Cost: 0.1253630816936493
Step: 40 Cost: 0.11652344465255737
Step: 50 Cost: 0.1083071231842041
Step: 60 Cost: 0.10067000985145569
Step: 70 Cost: 0.09357129782438278
Step: 80 Cost: 0.08697342872619629
Step: 90 Cost: 0.08084077388048172
Step: 100 Cost: 0.07514046877622604
```

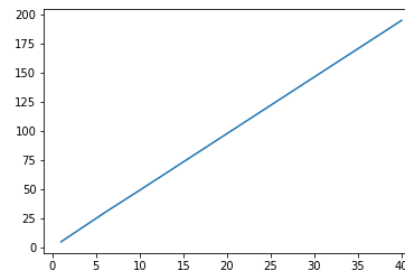
=====
Report

=====
Linear Regression Formula: Y = [[4.854571]]x + [0.6226092]
=====

```
[[ 49.168316]
 [ 97.71403 ]
 [146.25974 ]
 [194.80544 ]]
```

```
In [5]: 1 graph1_x = np.concatenate([[k[0] for k in x], [k[0] for k in x_test]])
2       2 graph1_y = np.concatenate([[k[0] for k in y], [k[0] for k in exp_val1]])
3       sns.lineplot(x = graph1_x, y = graph1_y)
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24ff65ba8>



Lecture7

Rate & Overfitting

```
In [6]: 1 lr = 15
2
3 optimizer = tf.train.GradientDescentOptimizer(lr)
4 train = optimizer.minimize(cost)
5
6 with tf.Session() as sess:
7     sess.run(tf.global_variables_initializer())
8
9     for step in range(101):
10         _, cost_val = sess.run([train, cost], feed_dict = {X:x, Y:y})
11
12         if step % 10 == 0:
13             print(f'Step: {step} Cost: {cost_val}')
14
15     w_val, b_val = sess.run([W, b])
16
17     print('='*80)
18     print('Report')
19     print('='*80)
20     print(f'Linear Regression Formula: Y = {w_val}x + {b_val}')
21     print('='*80)
22     print('='*80)
23     exp_val2 = sess.run(hypothesis, feed_dict = {X:x_test})
24     print(exp_val2)
25
```

```
Step: 0 Cost: 378.2164611816406
Step: 10 Cost: inf
Step: 20 Cost: nan
Step: 30 Cost: nan
Step: 40 Cost: nan
Step: 50 Cost: nan
Step: 60 Cost: nan
Step: 70 Cost: nan
Step: 80 Cost: nan
Step: 90 Cost: nan
Step: 100 Cost: nan
```

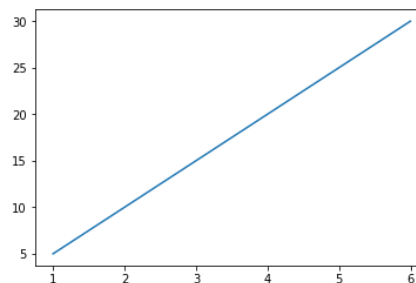
Report

Linear Regression Formula: Y = [[nan]]x + [nan]

[[nan]
[nan]
[nan]
[nan]]

```
In [7]: 1 graph2_y = np.concatenate([[k[0] for k in y], [k[0] for k in exp_val2]])
2       2 sns.lineplot(x = graph1_x, y = graph2_y)
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24c676048>



Lecture7

Rate & Overfitting

```
In [8]: 1 lr = 0.0000000001
2
3 optimizer = tf.train.GradientDescentOptimizer(lr)
4 train = optimizer.minimize(cost)
5
6 with tf.Session() as sess:
7     sess.run(tf.global_variables_initializer())
8
9     for step in range(101):
10         _, cost_val = sess.run([train, cost], feed_dict = {X:x, Y:y})
11
12         if step % 10 == 0:
13             print(f'Step: {step} Cost: {cost_val}')
14
15         w_val, b_val = sess.run([w, b])
16
17         print('='*80)
18         print('Report')
19         print('='*80)
20         print(f'Linear Regression Formula: Y = {w_val}x + {b_val}')
21         print('='*80)
22         print('='*80)
23         exp_val3 = sess.run(hypothesis, feed_dict = {X:x_test})
24         print(exp_val3)
25
```

```
Step: 0 Cost: 314.02099609375
Step: 10 Cost: 314.02099609375
Step: 20 Cost: 314.02099609375
Step: 30 Cost: 314.02099609375
Step: 40 Cost: 314.02099609375
Step: 50 Cost: 314.02099609375
Step: 60 Cost: 314.02099609375
Step: 70 Cost: 314.02099609375
Step: 80 Cost: 314.02099609375
Step: 90 Cost: 314.02099609375
Step: 100 Cost: 314.02099609375
```

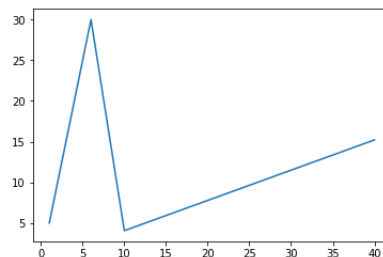
Report

Linear Regression Formula: Y = [[0.37176648]]x + [0.3386665]

```
[[ 4.056331]
 [ 7.773996]
 [11.491661]
 [15.209326]]
```

```
In [9]: 1 graph3_y = np.concatenate(((k[0] for k in y), [k[0] for k in exp_val3]))
2       sns.lineplot(x = graph1_x, y = graph3_y)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24c5b39e8>



Lecture7

Rate & Overfitting

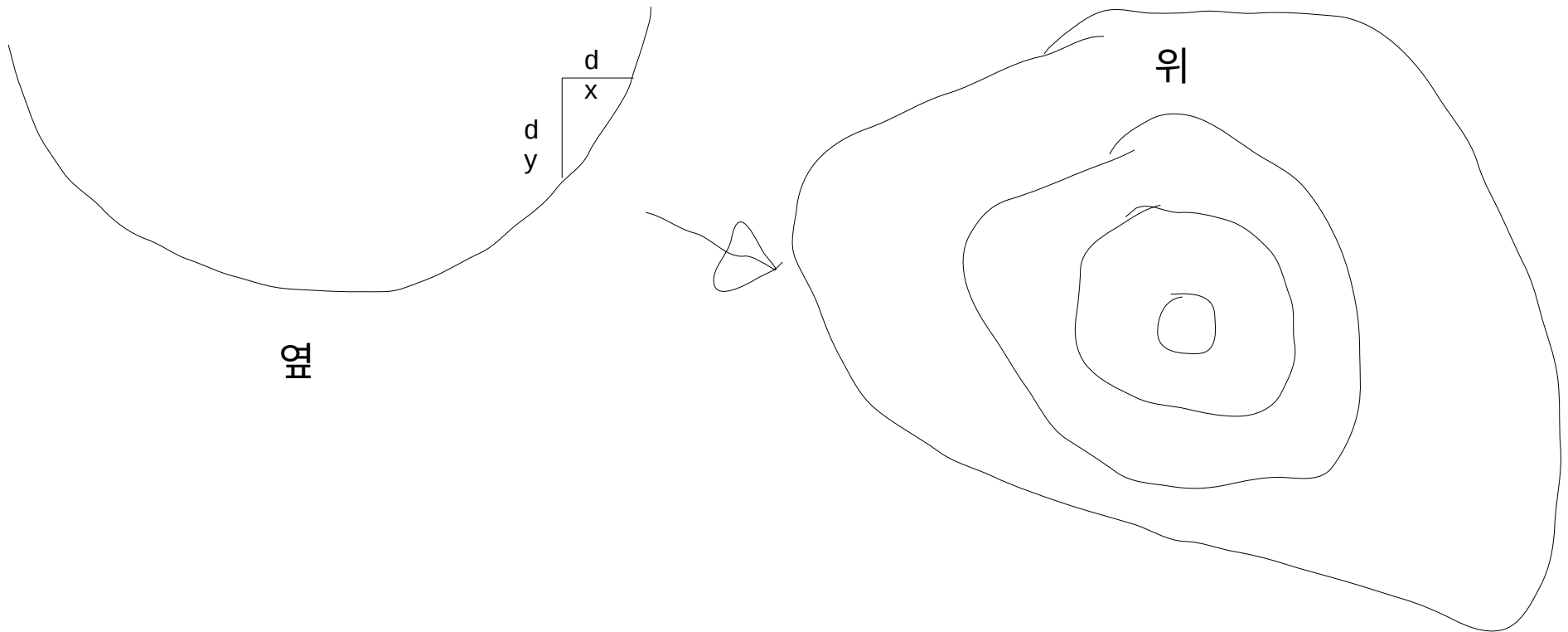


그렇다면 어떻게 좋은 learning rate 를 결정할 수 있는가 ?

→ 반복적인 시행 착오와 관찰로 직접 알아내는 방법 밖에 없음

Lecture7

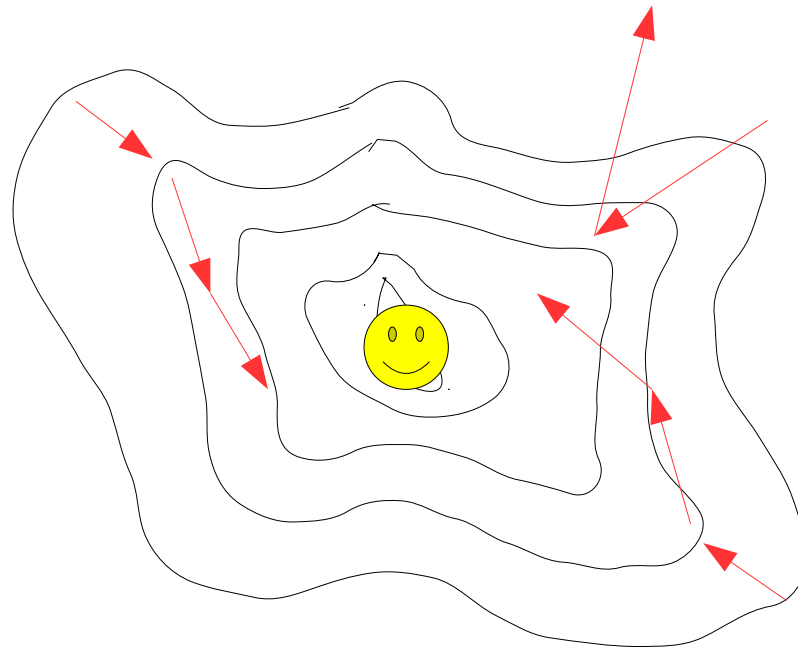
Normalisation



실제로 대부분의 데이터는 예시와는 다르게 고차원의 데이터로 되어있음 .
2 개의 변수만 있더라도 cost 함수의 그래프는 3 차원 그래프가 됨

Lecture7

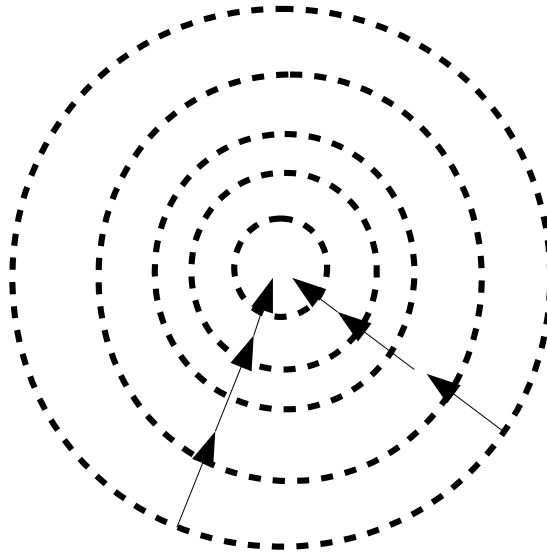
Normalisation



차원이 높아지고 변수의 range 가 달라짐에 따라 그래프는 점점 더 정규적이지 않은 모습이 되고 , cost 가 최소가 되는 hypothesis 의 이상값을 찾기 어려워짐

Lecture7

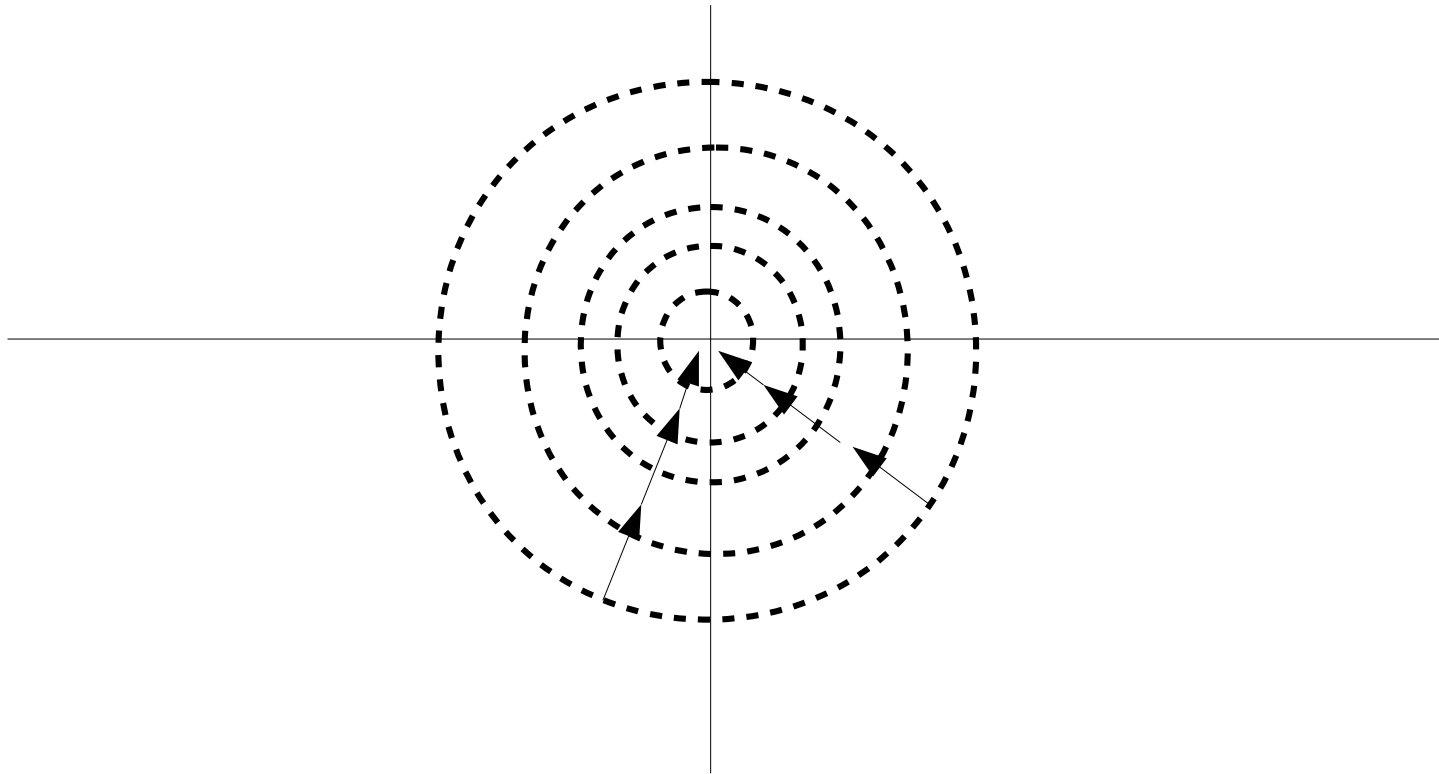
Normalisation



이상적인 형태의 그래프라면 어느 방향에서던 기울기가 극단적인 값을 띄지 않으므로 안전하게 최저의 cost 를 출력하는 최적의 hypothesis 값을 찾아갈 수 있음

Lecture7

Normalisation



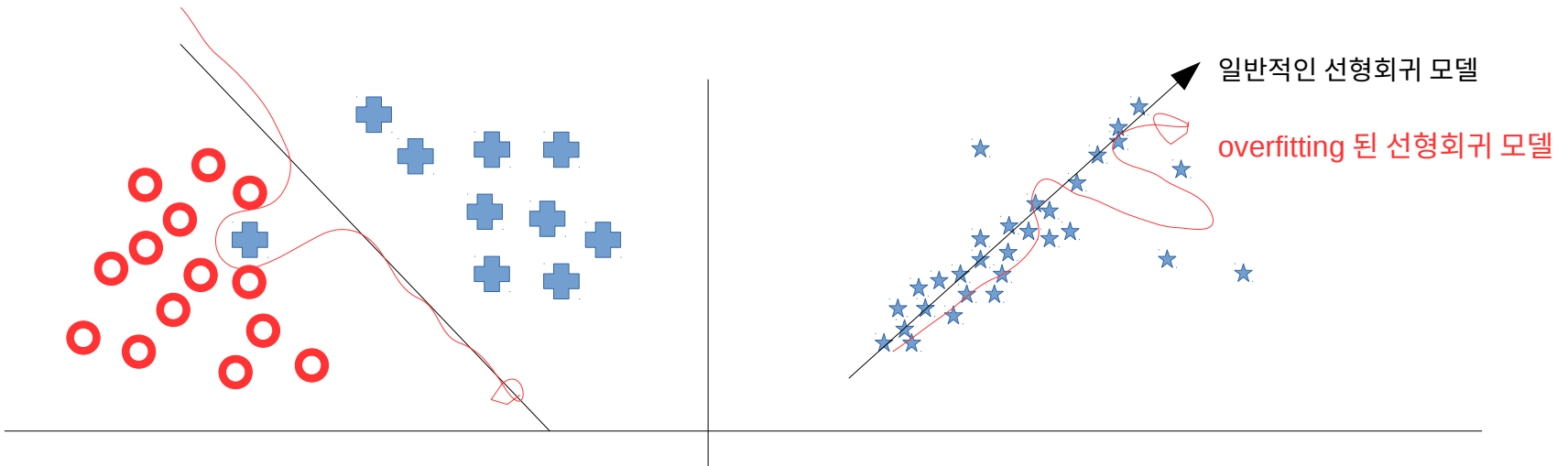
방법 1: 데이터의 중심이 0에 위치하도록 한다 .

방법 2: 데이터를 정규화시킨다 . (변수의 range 를 다른 변수들과 흡사하게 변환시킨다 .)

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

Lecture 7

Overfitting




Overfitting 된 모델의 경우 학습한 데이터 (training set) 에 한해서 높은 일치율 (낮은 cost) 를 보여주지만 , 실제 데이터 (test set) 에서는 상대적으로 낮은 일치율 (높은 cost) 를 보인다 . Test set 의 outlier 에 민감하게 반응하기 때문

해결 방법)

1. 학습할 데이터를 더 많이 준다 .
2. 변수를 줄여준다 . (통계에서 파라미터를 줄이는 것과 같음)
3. 데이터를 정규화 한다 .

Lecture7

Regularisation

$$\mathcal{L} = \frac{1}{N} \sum_i \mathcal{D}(s(w x_i + b), L_i) + \lambda \sum W^2$$


The diagram illustrates the components of the cost function. A blue arrow points from the text 'TRAINING SET' to the index i in the summation \sum_i . Another blue arrow points from 'TRAINING SET' to the input x_i in the hypothesis $s(w x_i + b)$. A third blue arrow points from 'TRAINING SET' to the target L_i in the loss function $\mathcal{D}(s(w x_i + b), L_i)$. The loss function \mathcal{D} is written in orange, the hypothesis $s(w x_i + b)$ is written in orange, and the target L_i is written in blue. The regularization term $\lambda \sum W^2$ is written in blue, with λ in red and $\sum W^2$ in green.

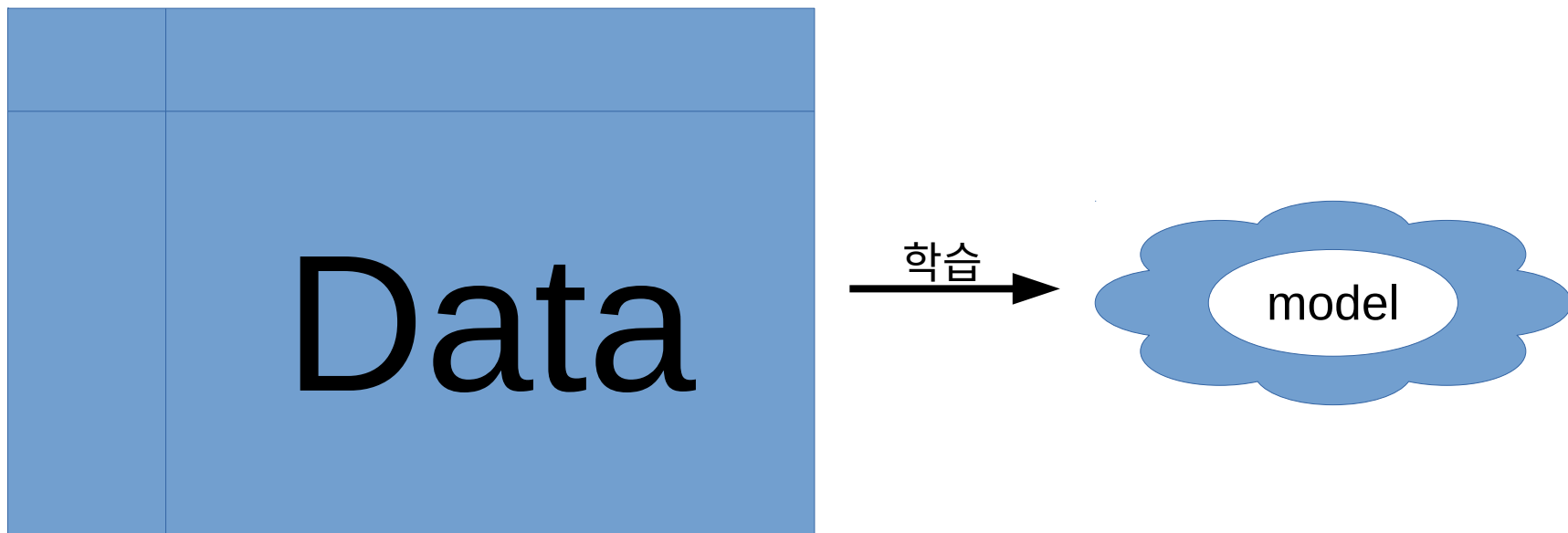
Cost 함수에 가중치 W 값의 제곱 (서로 상쇄되는 것을 막기 위해 제곱) 을 더해서 가중치 역시 줄어드는 방향으로 hypothesis 를 수렴시킨다 .

앞의 람다는 상수로 가중치를 얼마나 줄일지 정하는 도구 , 연구자가 재량껏 적당히 바뀌가며 설정해야한다 .

Lecture7

Training set and

Test set



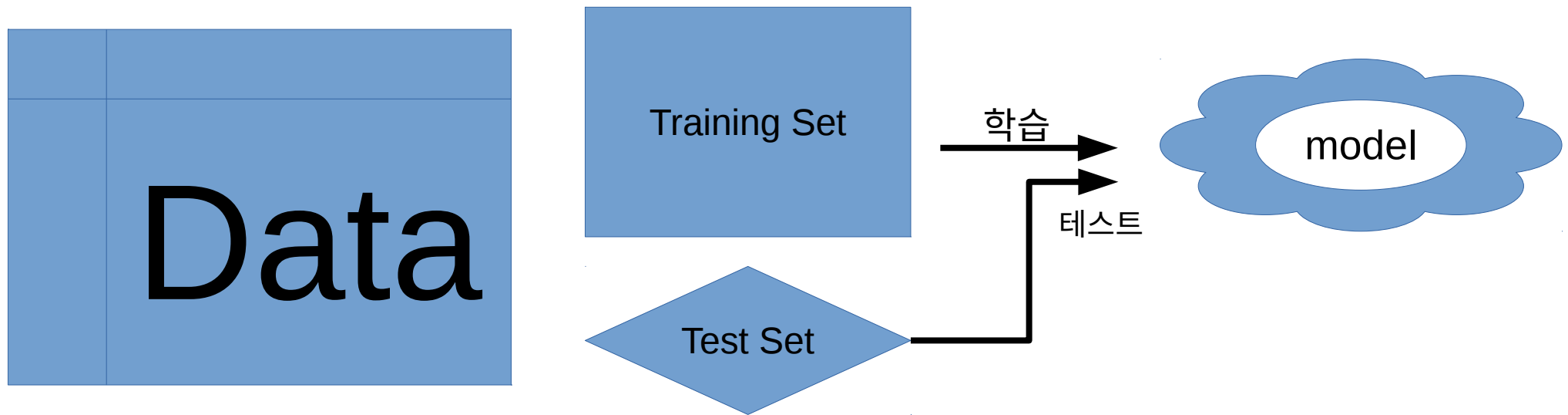
주어진 데이터 전부로 모델을 학습시키는 경우)

주어진 데이터를 모델에 다시 입력시키면 , 답을 알고 있으므로 100% 에 가까운 정답률을 보여서 모델이 얼마나 좋은지를 확인 할 수 없

Lecture7

Training set and

Test set

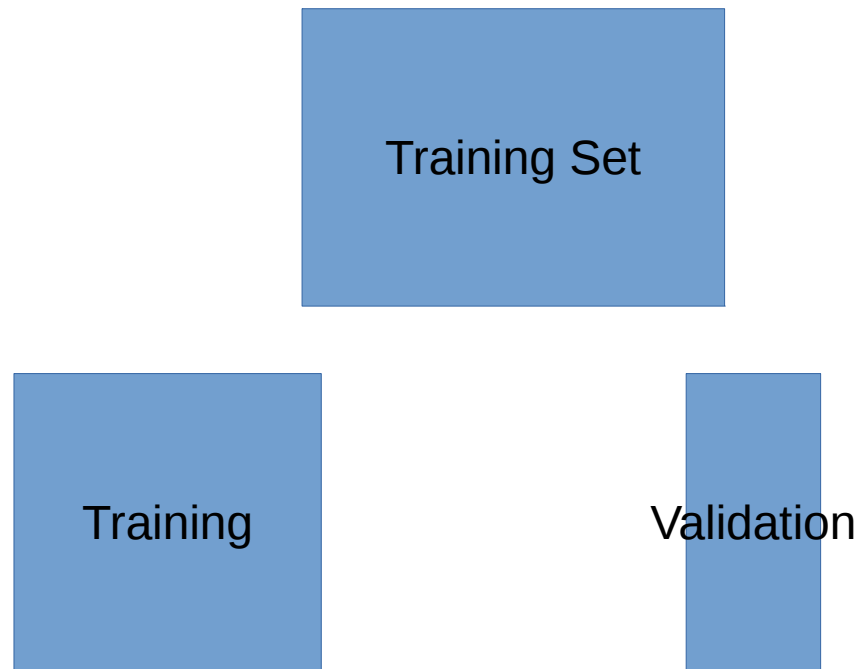


데이터를 training set 과 test set 으로 나누어 training set 으로는 model 을 학습시키고 , test set 의 변수를 model 에 주입한 후 출력되는 값과 테스트 셋의 종속변수를 비교하여 model 의 성능을 검증한다 .

Lecture7

Training set and

Test set



Training set 은 다시 training data 와 validation data 로 나눌 수 있다 .

Training data 는 model 을 학습시키는데 사용되며 , validation data 는 각종 상수 (learning rate 나 Regularisation 의 램다 값) 을 정하는데 사용된다 .