Learning

1. 선형회귀(Linear Regression)
2. 가설(Hypothesis)
3. 비용함수(Cost function)
4. Gradient descent algorithm (minimize cost)
5. XOR – Neural Network
6. Multiple-Layer Perceptron 학습

참고

- 모두를 위한 머신러닝/딥러닝 강의 (홍콩과기대 김성훈)
  https://hunkim.github.io/ml/
  https://github.com/hunkim/DeepLearningZeroToAll

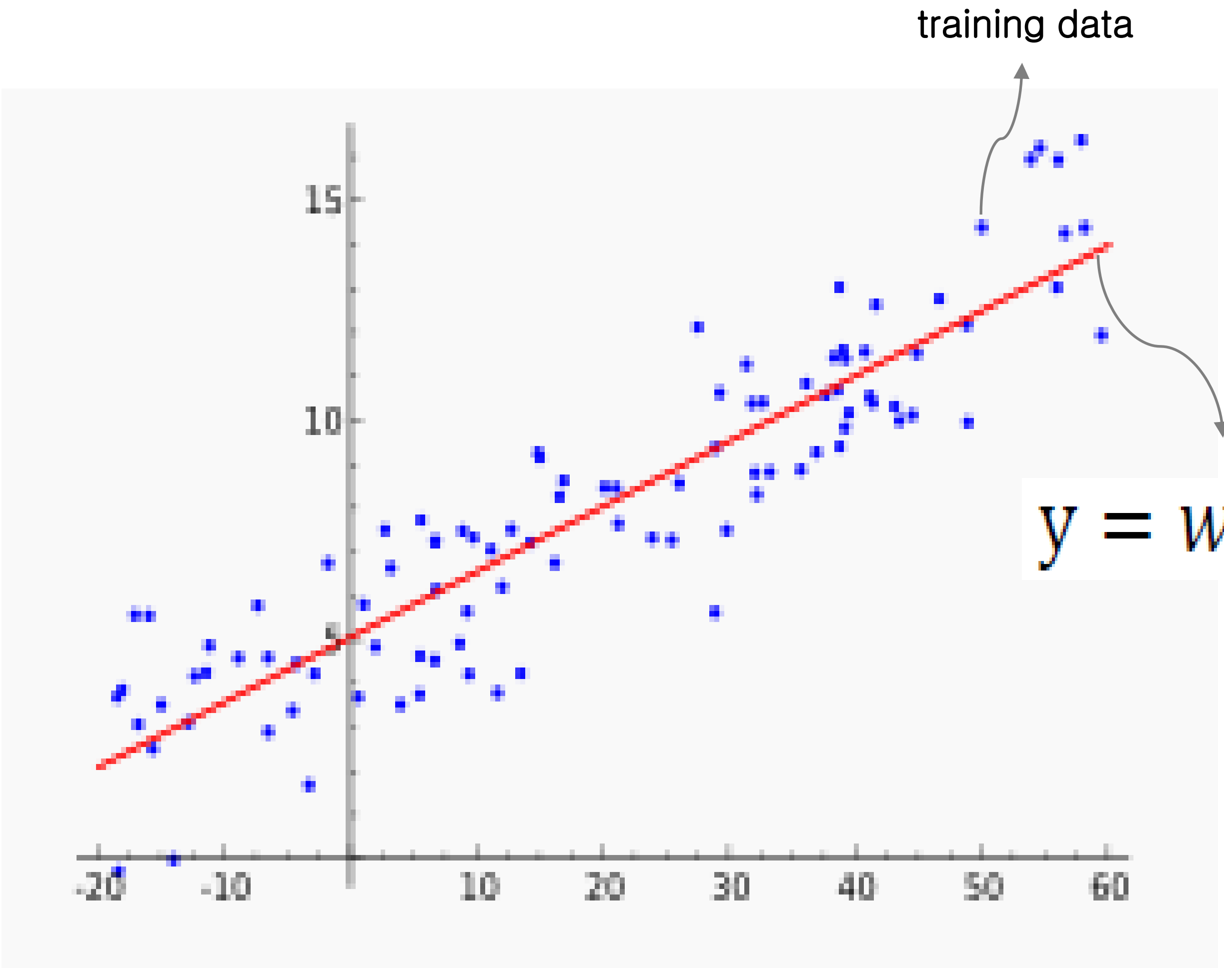
- [부스트코스] 텐서플로우로 시작하는 딥러닝 기초
  - edwith에서 제공되는 개발자인증코스
  - Tensorflow 2.X 코드반영

  https://www.edwith.org/boostcourse-dl-tensorflow/
  https://github.com/deeplearningzerotoall/TensorFlow/tree/master/tf_2.x

- Stanford Machine Learning
  http://www.holehouse.org/mlclass/

Linear Regression

training data(학습데이터) 를 잘 표현하는 직선방정식을 구하는 문제
최적의 w, b를 찾는 문제



$$y = wx + b$$

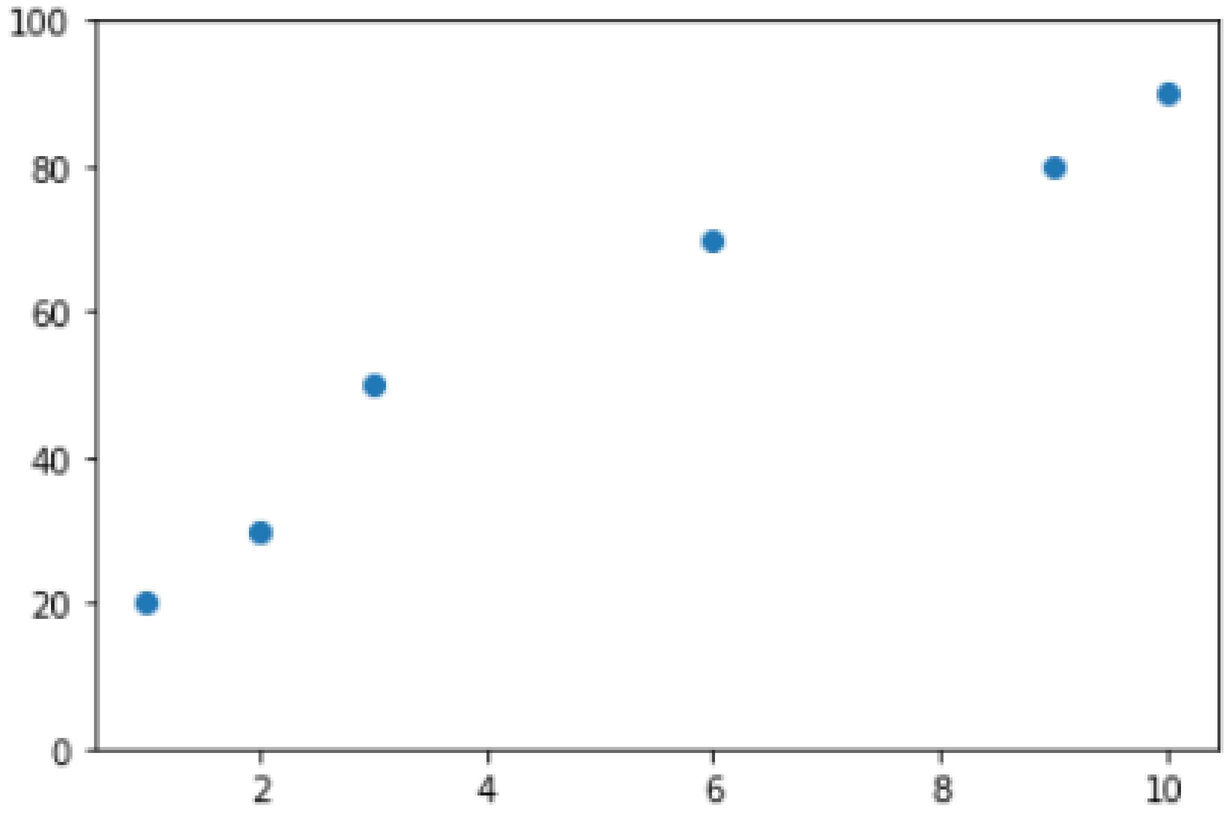
Regression 문제

공부한 시간에 따른 점수 데이터

x (hours)	y (score)
10	90
9	80
3	50
2	30

Regression 문제

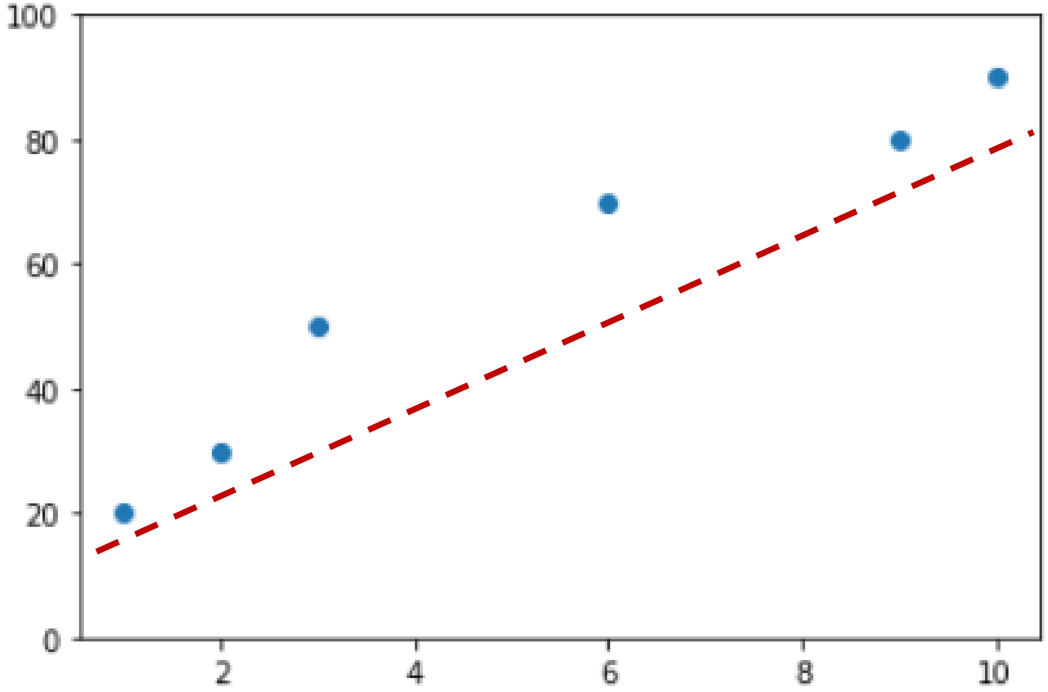
x (hours)	y (score)
10	90
9	80
3	50
2	30



가설(Hypothesis linear)

$$H(x) = Wx + b$$

- 가설(Hypothesis)
  - 데이터로 가장 잘 설명할 수 있는 직선 방정식
  - 기울기(W, weight)와 절편(b, bias)으로 표현



가설(Hypothesis linear)

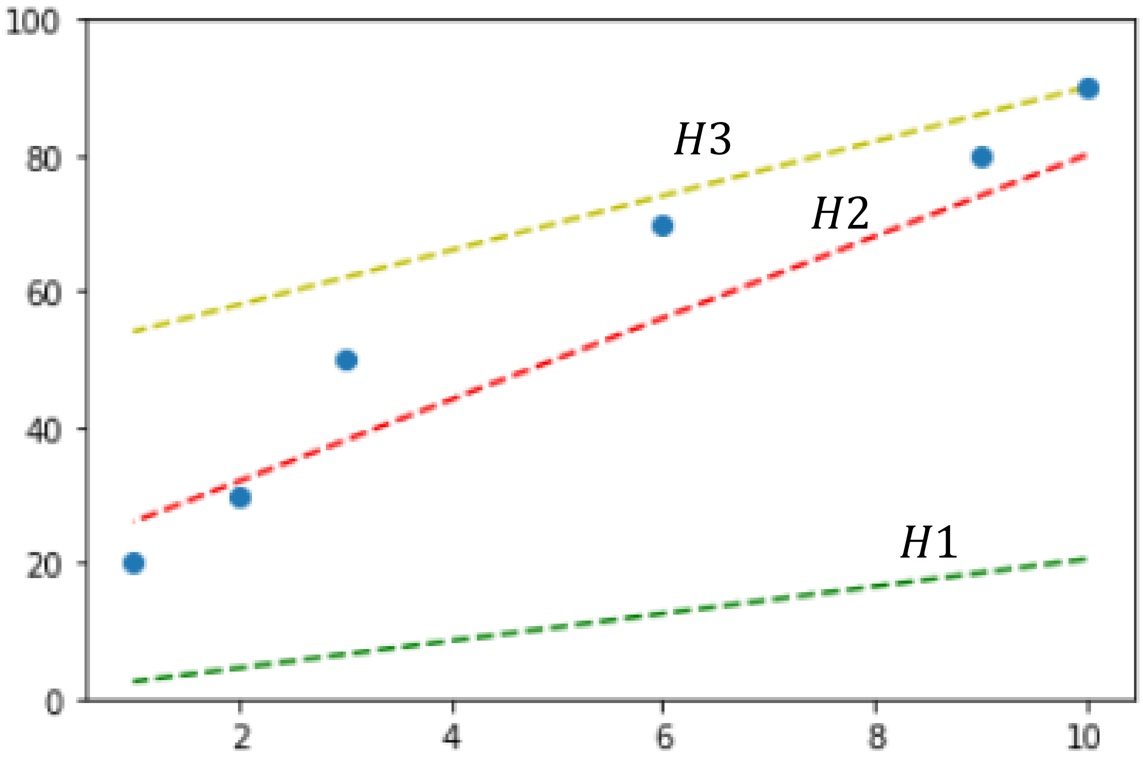
$$H(x) = Wx + b$$

$H1 = 2x + 0.5$

$H2 = 6x + 20$

$H3 = 4x + 50$

어떤 가설이 더 좋은가 ??



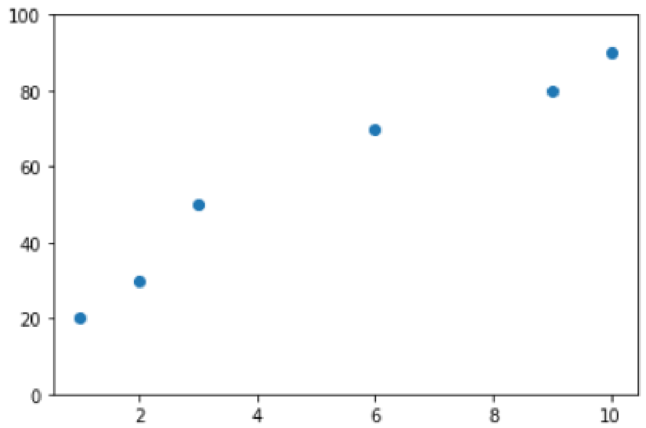
가설(Hypothesis linear)

Data

```
x = np.array([10, 9, 6, 3, 2, 1], dtype=float)
y = np.array([90, 80, 70, 50, 30, 20], dtype=float)
```

```
import matplotlib.pyplot as plt
plt.plot(x, y, 'o')
plt.ylim(0, 100)
```

(0, 100)



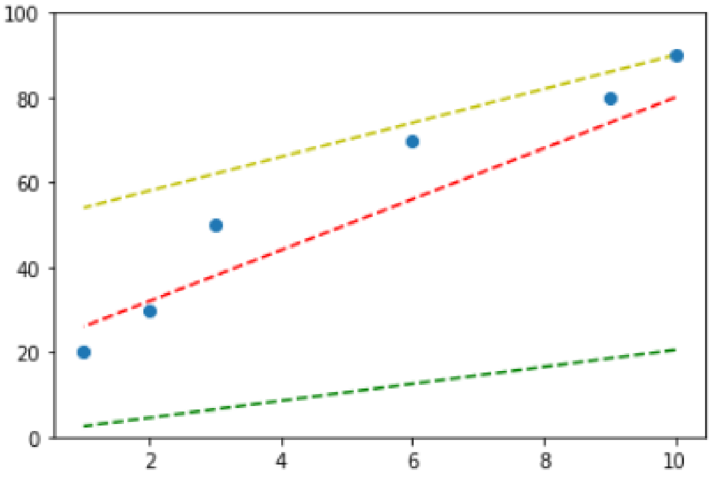
Hypothesis

데이터로 가장 잘 표현할 수 있는 직선 방정식

$H(x) = Wx + b$

```
#hypothesis

W, b = 2, 0.5
hypothesis1 = W * x + b
W, b = 6, 20
hypothesis2 = W * x + b
W, b = 4, 50
hypothesis3 = W * x + b
```

```
plt.plot(x, hypothesis1, 'g--')
plt.plot(x, hypothesis2, 'r--')
plt.plot(x, hypothesis3, 'y--')
plt.plot(x, y, 'o')
plt.ylim(0, 100)
plt.show()
```



Cost (loss, error)

cost(비용)

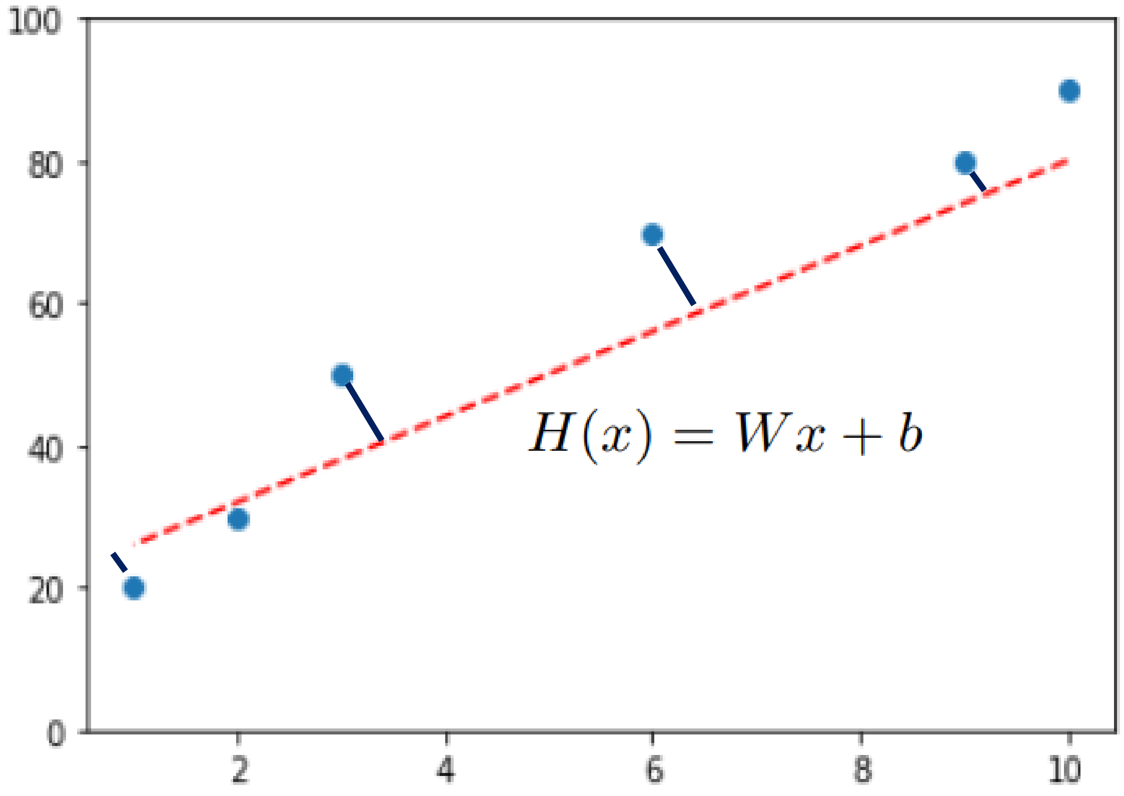
가설 (H)과 실제값(y)와의 오차(error)

$$H(x) - y$$

$\boldsymbol{H = 6x + 20}$

x = 2, y=30
cost = 6*2+20 – 30 = 32−30 = 2

x = 6, y = 70
cost = 6*6+20 – 70 = 56−70 = −24



Cost function (loss, object)

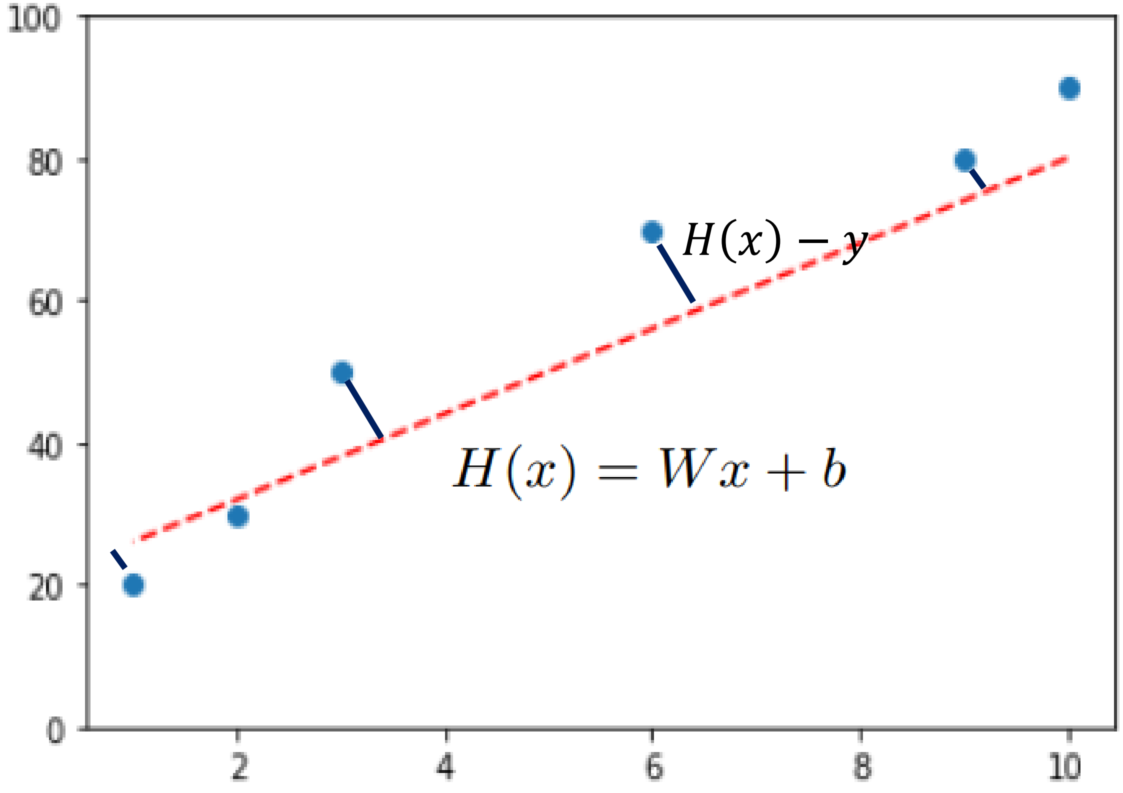
- cost function

  오차를 측정하는 함수

  mean squared error (MSE)

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

$$\frac{((H(x^{(1)}) - y^{(1)}) + (H(x^{(2)}) - y^{(2)}) + ..(H(x^{(6)}) - y^{(6)}))^2}{6}$$



Minimize cost

$$H(x) = Wx + b$$

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

$$cost(W,b) = \frac{1}{2m}\sum_{i=1}^{m}(Wx_i + \ b) - y_i)^2$$

$$\underset{W,b}{minimize}\ cost(W,b)$$

Simplified hypothesis

$$H(x) = Wx$$

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

Cost function

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

x	Y
1	1
2	2
3	3

- W=1, cost(W)=0

$$\frac{1}{3}((1 * 1 - 1)^2 + (1 * 2 - 2)^2 + (1 * 3 - 3)^2)$$

- W=0, cost(W)=4.67

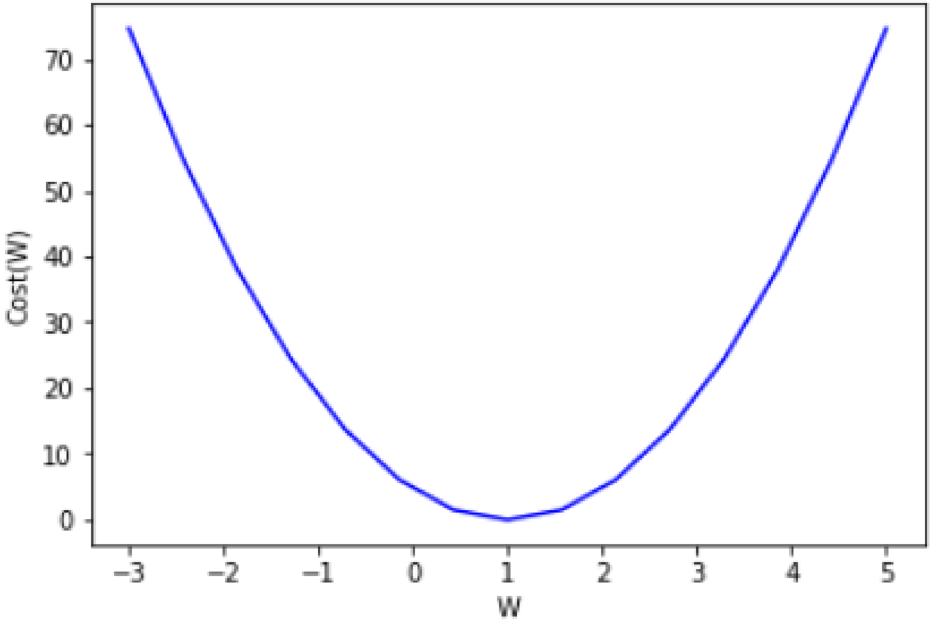
$$\frac{1}{3}((0 * 1 - 1)^2 + (0 * 2 - 2)^2 + (0 * 3 - 3)^2)$$

- W=2, cost(W)=?

Cost function

```
import numpy as np

X = np.array([1, 2, 3])
Y = np.array([1, 2, 3])

def cost_func(W, X, Y):
    c = 0
    for i in range(len(X)):
        c += (W * X[i] - Y[i]) ** 2
    return c / len(X)

for feed_W in np.linspace(-3, 5, num=15):
    curr_cost = cost_func(feed_W, X, Y)
    print("{:6.3f} | {:10.5f}".format(feed_W, curr_cost))
```

```
import matplotlib.pyplot as plt

plt.plot(W_values, cost_values, "b")
plt.ylabel('Cost(W)')
plt.xlabel('W')
plt.show()
```



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

QUIZ!

점수 데이터로 W(0~22)에 대한 cost function 그리기

x (hours)	y (score)
10	90
9	80
3	50
2	30

Cost function in TensorFlow

- Tensorflow
  - 텐서플로우는 머신러닝과 딥 뉴럴 네트워크 연구를 목적으로 구글의 인공지능 연구 조직인 구글 브레인 팀의 연구자와 엔지니어들에 의해 개발
  - https://www.tensorflow.org/tutorials
  - https://tensorflowkorea.gitbooks.io/tensorflow-kr/content/

```
# 텐서플로우 최신 버전 설치
!pip install --upgrade tensorflow

#import

import tensorflow as tf

#버전확인
print("tensorflow version: ", tf.__version__)
```

Cost function in TensorFlow

```
## tensorflow 사용 예
x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 4, 5]

W = tf.Variable(2.0)  #변수 설정, tf.Variable
b = tf.Variable(0.5)

print("x=", x)

print("W=", W)
print("W.numpy()=", W.numpy())

meanX = tf.reduce_mean(x)  #x 평균, 값 형태
print("mean(x)=", meanX)
print("mean(x).numpy()=", meanX.numpy())
squareY = tf.square(y)   #y 제곱, 배열형태, tf.Tensor
print("square(y)=", squareY)

hypothesis = W * x_data + b
print("hypothesis=", hypothesis) #tf.Tensor
print("hypothesis.numpy())=", hypothesis.numpy())
```

```
tensorflow version:  2.0.0
x= [1, 2, 3, 4, 5]
W= <tf.Variable 'Variable:0' shape=() dtype=float32, numpy=2.0>
W.numpy()= 2.0
mean(x)= tf.Tensor(3, shape=(), dtype=int32)
mean(x).numpy()= 3
square(y)= tf.Tensor([ 1  4  9 16 25], shape=(5,), dtype=int32)
hypothesis= tf.Tensor([ 2.5  4.5  6.5  8.5 10.5], shape=(5,), dtype=float32)
hypothesis.numpy())= [ 2.5  4.5  6.5  8.5 10.5]
```

Cost function in TensorFlow

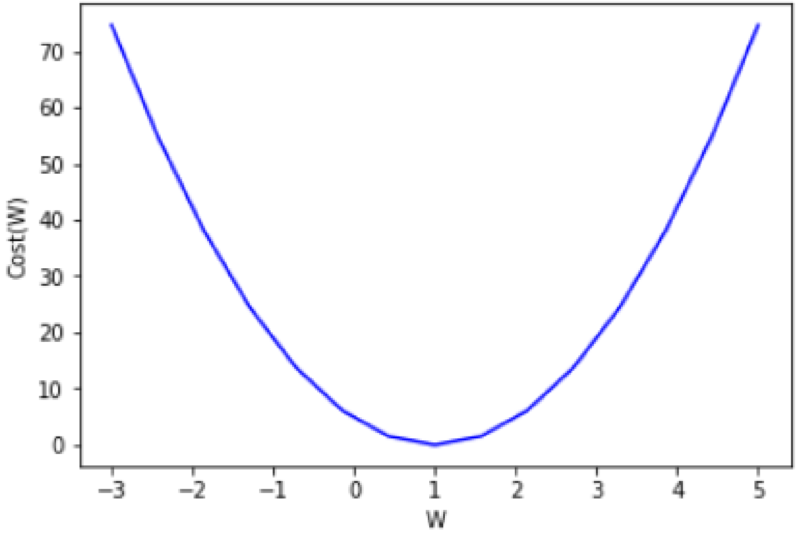
```
X = np.array([1, 2, 3])
Y = np.array([1, 2, 3])

def cost_func(W, X, Y):
  hypothesis = X * W
  return tf.reduce_mean(tf.square(hypothesis - Y))

W_values = np.linspace(-3, 5, num=15)
cost_values = []

for feed_W in W_values:
    curr_cost = cost_func(feed_W, X, Y)
    cost_values.append(curr_cost)
    print("{:6.3f} | {:10.5f}".format(feed_W, curr_cost))
```

Gradient descent algorithm

- 경사하강(Gradient descent) 알고리즘
  - 비용함수(cost function)를 최소화하기 위한 최적화(optimization) 알고리즘
  - 경사를 따라 내려오면서 비용함수의 최소점을 찾는 알고리즘
  - Cost (W, b)를 최소로 하는 W, b를 찾는 알고리즘
  - cost function의 일반적인 형식 : cost (w1, w2, ...)

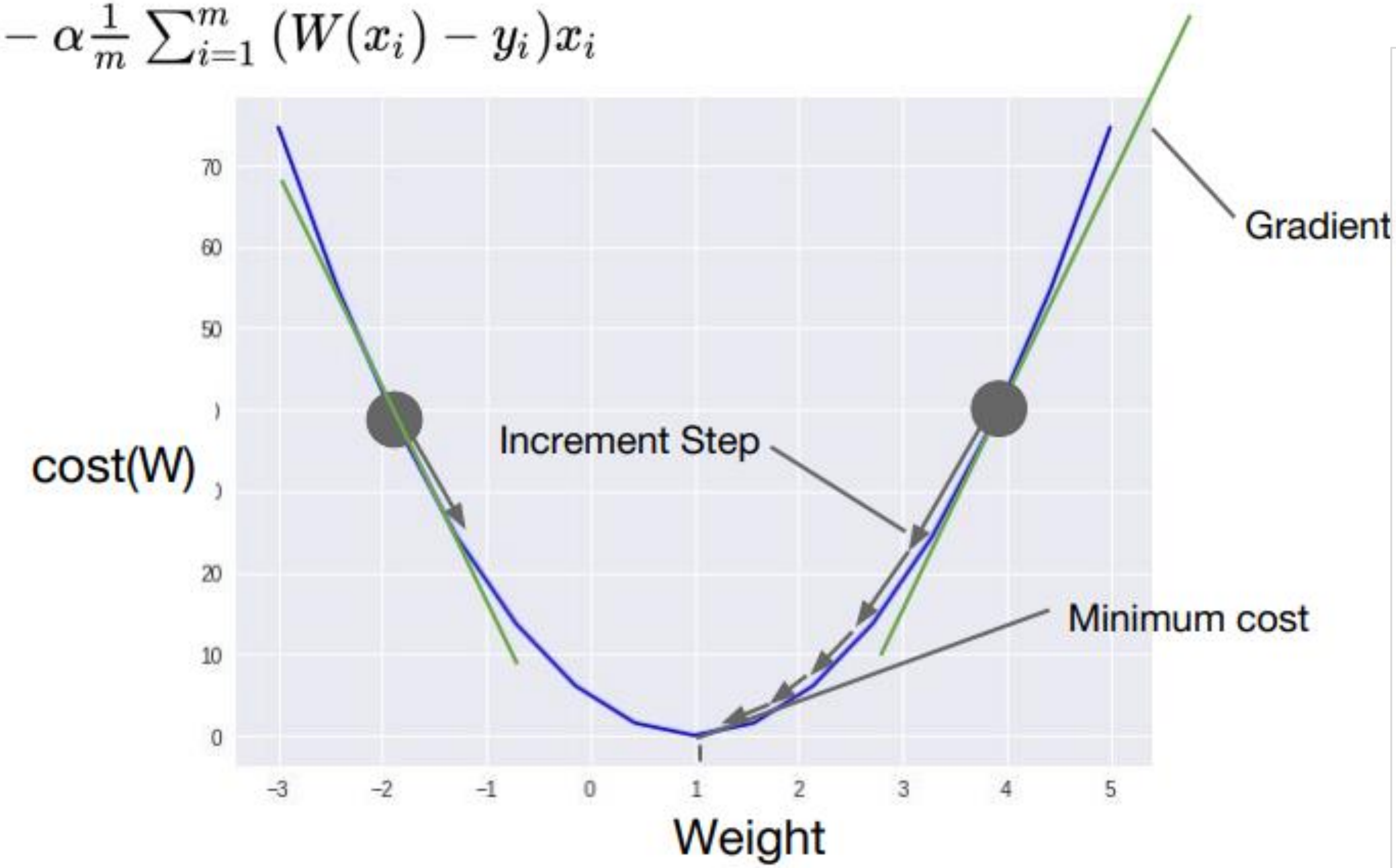
- 동작방식

  - W, b 초기화
  - cost(W, b)가 줄어들도록 W, b를 지속적으로 변경
  - 기울기를 구하여 cost(W, b)가 최소화되는 방향으로 수정
  - 최소점에 도달할 때 까지 반복



Gradient descent algorithm

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (W(x_i) - y_i)x_i$$



[경사하강(Gradient descent) 알고리즘]

- 경사를 따라 내려오면서 최저점을 찾는 알고리즘
- Cost 함수 에서의 한 점인 W에서의 기울기(gradient)를 구하여, 기울기와 학습률 α를 곱한값을 W에서 빼주면서 다음 W계산
  반복해서 cost가 최소가 되는 지점을 찾아감
- Cost함수의 W지점의 기울기는 미분으로 구함

https://www.edwith.org/boostcourse-dl-tensorflow/lecture/41845/

Gradient descent algorithm

- 비용함수의 일반적인 정의

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H(x_i) - y_i\right)^2$$

$$\Downarrow$$

$$cost(W,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(H(x_i) - y_i\right)^2$$

Gradient descent algorithm

- Gradient descent algorithm 정의

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

α : learning rate (학습률), 0.01~0.0001

$\frac{\partial cost(W)}{\partial W}$: cost(W)함수를 W에 대해 편미분 (기울기)

$$W := W - \alpha \frac{\partial}{\partial W} \frac{1}{2m} \sum_{i=1}^{m} (W(x_i) - y_i)^2$$

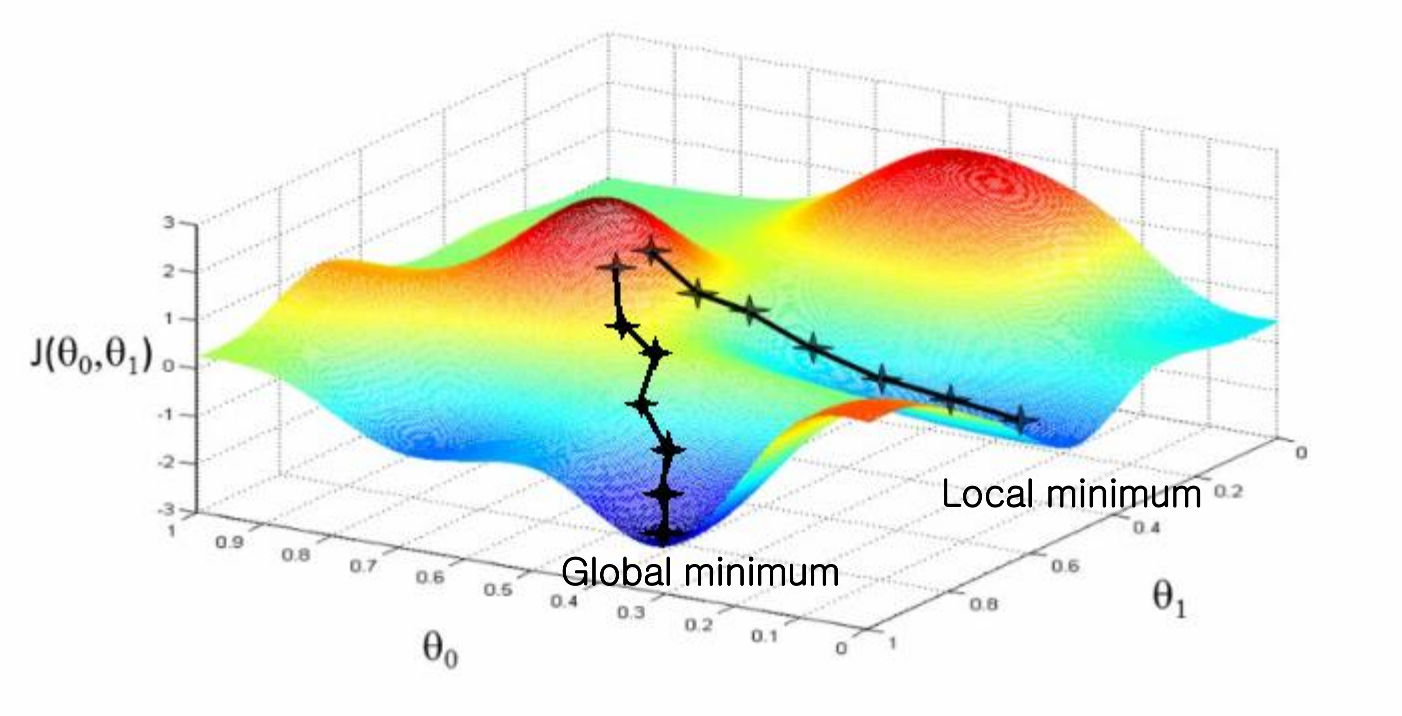
$$\Rightarrow \quad W := W - \alpha \frac{1}{2m} \sum_{i=1}^{m} 2(W(x_i) - y_i)x_i$$

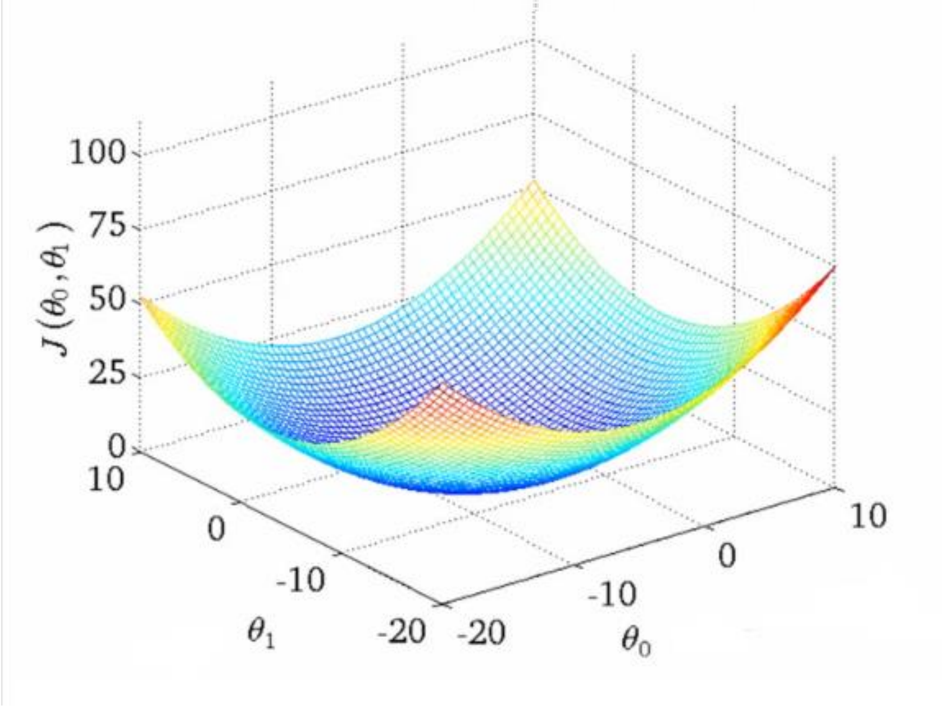
$$\Rightarrow \quad \boxed{W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (W(x_i) - y_i)x_i}$$

Gradient descent algorithm

- 기울기 경사 하강법(Gradient descent algorithm)을 사용하려면,
  비용함수 cost(W,b)가 볼록 함수(Convex function)이어야 함.

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





http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html

Gradient descent algorithm 구현

Gradient descent algorithm in pure Python

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

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

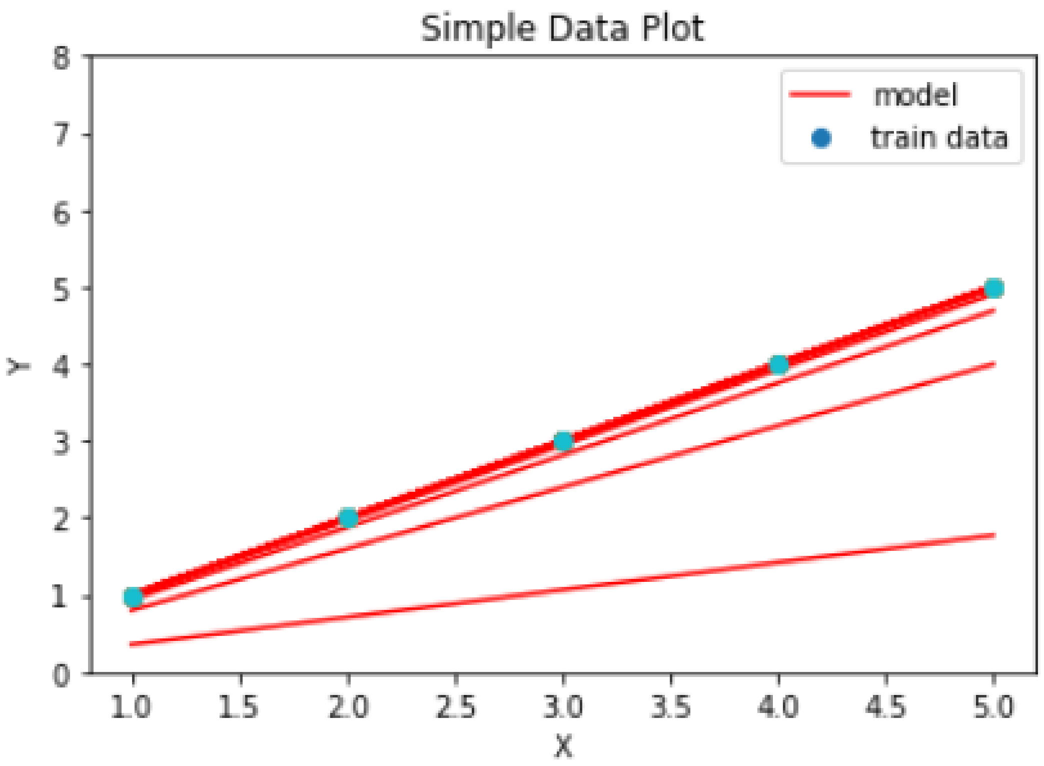
$$W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx^i - y^i)x^i$$

```
import numpy as np

X = np.array([1, 2, 3, 4, 5])
Y = np.array([1, 2, 3, 4, 5])

# W 초기화
# 평균 -100, 표준편차 100인 임의값으로 초기화
# 평균 0, 표준편차 100인 임의값으로 초기화
#np.random.normal(mean, std, size)
W = np.random.normal(-100, 100, 1)

print('step  |     W      | cost')

#300번 반복,  hypothesis, cost, gradient를 계산하여 W 갱신
for step in range(300):
    hypothesis = W * X
    cost =  np.mean((hypothesis - Y) ** 2)

    alpha = 0.01
    gradient = np.mean((W * X  - Y) * X)
    descent = W - (alpha * gradient)
    W = descent

    if step % 10 == 0:
        print('{:5} | {:10.4f} | {:10.6f}'.format(step, W[0], cost))
```

```
step  |     W      | cost
    0 |  -193.8869 | 527446.046157
   10 |   -59.7691 | 51283.561018
   20 |   -17.9488 | 4986.298883
   30 |    -4.9086 | 484.817670
   40 |    -0.8424 |  47.138805
   50 |     0.4255 |   4.583304
   60 |     0.8209 |   0.445635
   70 |     0.9441 |   0.043329
   80 |     0.9826 |   0.004213
   90 |     0.9946 |   0.000410
  100 |     0.9983 |   0.000040
  110 |     0.9995 |   0.000004
  120 |     0.9998 |   0.000000
  130 |     0.9999 |   0.000000
  140 |     1.0000 |   0.000000
  150 |     1.0000 |   0.000000
  160 |     1.0000 |   0.000000
  170 |     1.0000 |   0.000000
  180 |     1.0000 |   0.000000
  190 |     1.0000 |   0.000000
  200 |     1.0000 |   0.000000
  210 |     1.0000 |   0.000000
  220 |     1.0000 |   0.000000
  230 |     1.0000 |   0.000000
  240 |     1.0000 |   0.000000
  250 |     1.0000 |   0.000000
  260 |     1.0000 |   0.000000
  270 |     1.0000 |   0.000000
  280 |     1.0000 |   0.000000
  290 |     1.0000 |   0.000000
```

Gradient descent algorithm 구현

```
# hypothesis graph
        plt.plot(X, hypothesis, 'r-')
        plt.plot(X, Y, 'o')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title("Simple Data Plot")
        plt.legend(["model", "train data"], loc="best")
        plt.ylim(0, 8)
```



Gradient descent algorithm 구현

Gradient descent algorithm in TensorFlow

```
# Gradient descent algorithm in TensorFlow
X = [1, 2, 3, 4, 5]
Y = [1, 2, 3, 4, 5]

# W 초기화
# 평균 0, 표준편차 100인 임의값으로 초기화
W = tf.Variable(tf.random.normal([1], 0., 100.))

print('step  |     W       | cost')

#300번 반복,  hypothesis, cost, gradient decent 계산하여 W갱신
for step in range(300):
    hypothesis = W * X
    cost = tf.reduce_mean(tf.square(hypothesis - Y))

    alpha = 0.01
    gradient = tf.reduce_mean(tf.multiply(tf.multiply(W, X) - Y, X))
    descent = W - tf.multiply(alpha, gradient)
    W.assign(descent)  #W =  descent

    if step % 10 == 0:
        print('{:5} | {:10.4f} | {:10.6f}'.format(
            step,  W.numpy()[0], cost.numpy()))
```

Simple Linear Regression in TensorFlow

Hypothesis

$$H(x) = Wx + b$$

Cost

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

Gradient descent algorithm

$$\underset{W,b}{minimize}\ cost(W, b)$$

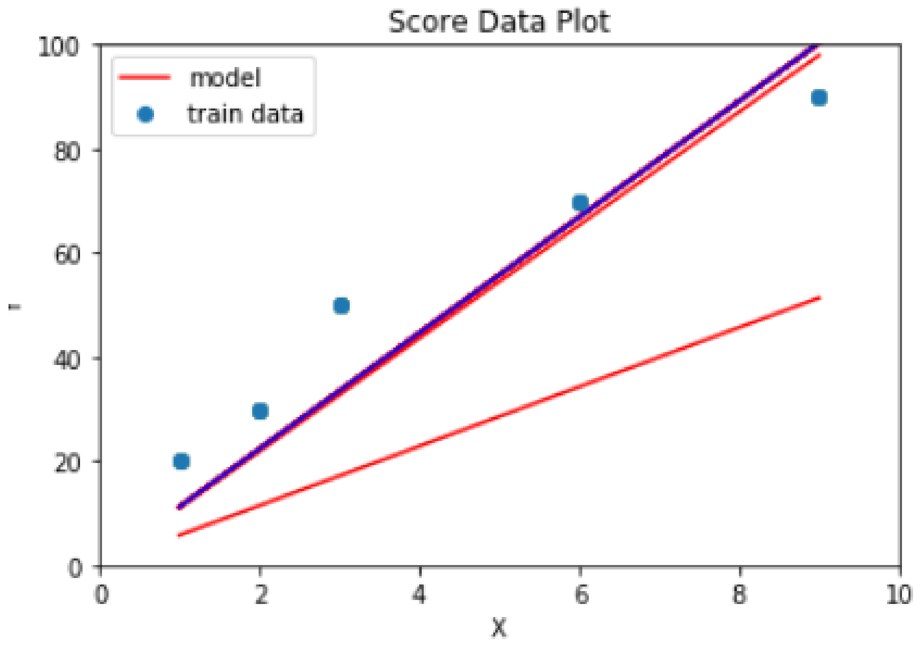
```
    0|    2.4520|    0.3760| 45.660004
   10|    1.1036|    0.0034|  0.206336
   20|    1.0128|   -0.0209|  0.001026
   30|    1.0065|   -0.0218|  0.000093
   40|    1.0059|   -0.0212|  0.000083
   50|    1.0057|   -0.0205|  0.000077
   60|    1.0055|   -0.0198|  0.000072
   70|    1.0053|   -0.0192|  0.000067
   80|    1.0051|   -0.0185|  0.000063
   90|    1.0050|   -0.0179|  0.000059
tf.Tensor(5.0066934, shape=(), dtype=float32)
tf.Tensor(2.4946523, shape=(), dtype=float32)
```

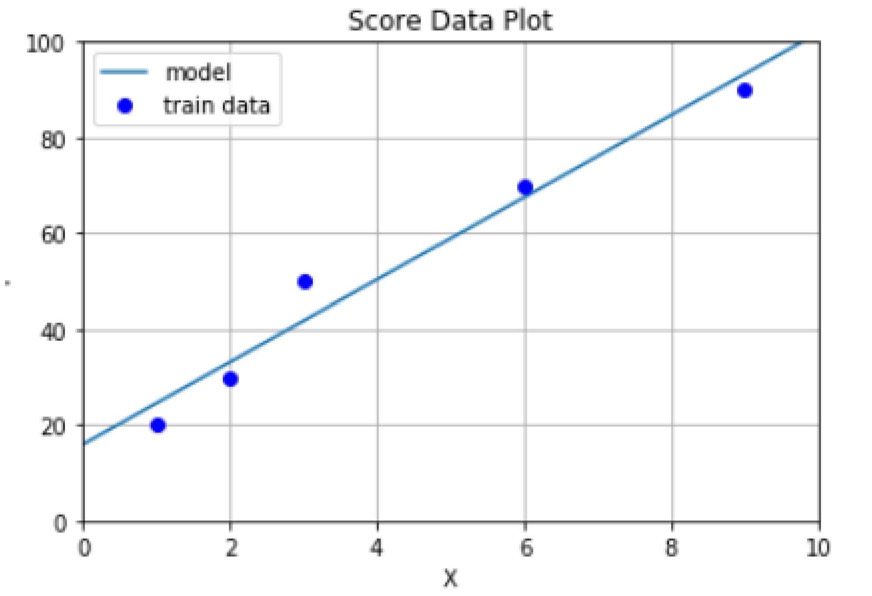
```
#simple linear regression in TensorFlow

import tensorflow as tf
import numpy as np

# Data
x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 4, 5]

learning_rate = 0.01

# W, b initialize
W = tf.Variable(2.9)
b = tf.Variable(0.5)

# W, b update
for i in range(100):
    # Gradient descent
    # tf.GradientTape() 안에서 변수의 정보를 tape에 저장
    with tf.GradientTape() as tape:
        hypothesis = W * x + b
        cost = tf.reduce_mean(tf.square(hypothesis - y))
    #gradient 계산
    W_grad, b_grad = tape.gradient(cost, [W, b])  #cost에서의 W, b의 기울기 반환
    #W, b 갱신
    W.assign_sub(learning_rate * W_grad)  #W = W - (learning_rate * W_grad)
    b.assign_sub(learning_rate * b_grad)
    if i % 10 == 0:
      print("{:5}|{:10.4f}|{:10.4f}|{:10.6f}".format(i, W.numpy(), b.numpy(), cost))

# predict
print(W * 5 + b)
print(W * 2.5 + b)
```

QUIZ!

1. 점수 데이터를 이용한 **linear regression**
2. 학습 과정 중 생성된 중간 모델 그래프 그리기
3. 새로운 데이터 **0, 4, 8**에 대한 결과 확인
4. **Sklearn**의 **LinearRegression** 결과와 비교

x (hours)	y (score)
10	90
9	80
3	50
2	30





XOR – Neural Network

기본 Library 선언 및 Tensorflow 버전 확인

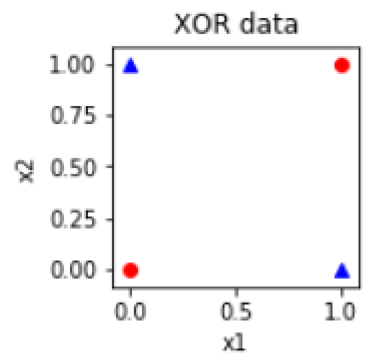
```
#기본 Library 선언 및 Tensorflow 버전 확인
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

print(tf.__version__)
```

```
2.0.0
```

XOR Data

- x_data가 2차원 배열이기에 2차원 공간에 표현하여 x1과 x2를 기준으로 y_data 0과 1로 구분하는 예제
- 붉은색원과 푸른색세모로 0과 1을 표시.

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

x_data = [[0, 0],
          [0, 1],
          [1, 0],
          [1, 1]]
y_data = [[0],
          [1],
          [1],
          [0]]

plt.figure(figsize= (2,2))
plt.scatter(x_data[0][0],x_data[0][1], c='red' , marker='o')
plt.scatter(x_data[3][0],x_data[3][1], c='red' , marker='o')
plt.scatter(x_data[1][0],x_data[1][1], c='blue' , marker='^')
plt.scatter(x_data[2][0],x_data[2][1], c='blue' , marker='^')

plt.xlabel("x1")
plt.ylabel("x2")
plt.title ("XOR data")
plt.show()
```



XOR – Neural Network

```python
#Tensorflow data API로 데이터 설정
#tf.data.Dataset.from_tensor_slices() : x_data, y_data를 이용하여 텐서플로우 데이터집합 생성
#batch() 한번에 학습시킬 크기

dataset = tf.data.Dataset.from_tensor_slices((x_data, y_data)).batch(len(x_data))

#features, label 타입변경
def preprocess_data(features, labels):
    features = tf.cast(features, tf.float32)
    labels = tf.cast(labels, tf.float32)
    return features, labels
```

```python
#3-Layer의 Neural Network 구조에 대한  weight, bias 초기값 랜덤으로 설정
W1 = tf.Variable(tf.random.normal([2, 1]), name='weight1')
b1 = tf.Variable(tf.random.normal([1]), name='bias1')

W2 = tf.Variable(tf.random.normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random.normal([1]), name='bias2')

W3 = tf.Variable(tf.random.normal([2, 1]), name='weight3')
b3 = tf.Variable(tf.random.normal([1]), name='bias3')
```

XOR – Neural Network

```python
#3-layer Neural Network 구조에 대한  출력(hypothesis)
def neural_net(features):
    layer1 = tf.sigmoid(tf.matmul(features, W1) + b1)
    layer2 = tf.sigmoid(tf.matmul(features, W2) + b2)
    layer3 = tf.concat([layer1, layer2],-1)
    layer3 = tf.reshape(layer3, shape = [-1,2])
    hypothesis = tf.sigmoid(tf.matmul(layer3, W3) + b3)
    return hypothesis

#cost(loss) function (MSE)
def loss_fn(hypothesis, labels):
    cost = -tf.reduce_mean(labels * tf.math.log(hypothesis) + (1 - labels) * tf.math.log(1 - hypothesis))
    return cost

#optimization (Gradient decent algorithm)
optimizer = tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.01)

# 예측결과와 출력 결과가 같은것의 평균으로 정확도 계산
def accuracy_fn(hypothesis, labels):
    predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
    accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, labels), dtype=tf.float32))
    return accuracy

#기울기 계산
def grad(hypothesis, features, labels):
    with tf.GradientTape() as tape:
        loss_value = loss_fn(neural_net(features),labels)
    return tape.gradient(loss_value, [W1, W2, W3, b1, b2, b3])
```

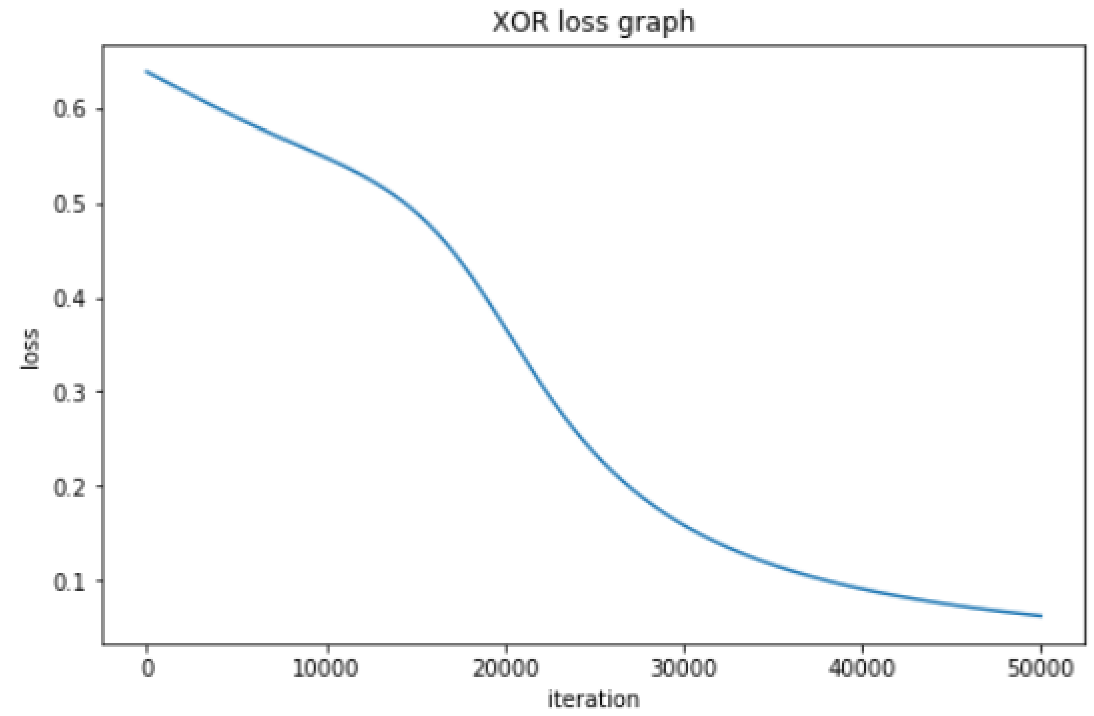
XOR – Neural Network

```python
#반복횟수 설정
EPOCHS = 50000

train_loss_list = []
for step in range(EPOCHS):
    for features, labels  in iter(dataset):
        features, labels = preprocess_data(features, labels) #타입변경
        grads = grad(neural_net(features), features, labels) #기울기 계산
        optimizer.apply_gradients(grads_and_vars=zip(grads,[W1, W2, W3, b1, b2, b3])) #기울기와 계수를 최적화 알고리즘에 적용
        loss = loss_fn(neural_net(features),labels) # 손실함수
        train_loss_list.append(loss)                # 학습 과정 기록
        if step % 5000 == 0:                        # 500번에 한번씩
            print("Iter: {}, Loss: {:.4f}".format(step, loss))

x_data, y_data = preprocess_data(x_data, y_data)
test_acc = accuracy_fn(neural_net(x_data),y_data)
print("Testset Accuracy: {:.4f}".format(test_acc))
```

```
Iter: 0, Loss: 0.6381
Iter: 5000, Loss: 0.5901
Iter: 10000, Loss: 0.5479
Iter: 15000, Loss: 0.4906
Iter: 20000, Loss: 0.3686
Iter: 25000, Loss: 0.2356
Iter: 30000, Loss: 0.1590
Iter: 35000, Loss: 0.1168
Iter: 40000, Loss: 0.0912
Iter: 45000, Loss: 0.0743
Testset Accuracy: 1.0000
```

XOR – Neural Network

```
# 그래프 그리기
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
markers = {'train': 'o', 'test': 's'}

x_loss = np.arange(len(train_loss_list))
plt.plot(x_loss, train_loss_list)
plt.xlabel("iteration")
plt.ylabel("loss")
plt.title("XOR loss graph")
plt.show()
```



Multiple-Layer Perceptron 학습

- Multiple Layer Perceptron 학습

1) 순전파(Forward Propagation)

입력층에서 출력층을 향하여 출력값 계산

(1)입력층과 은닉층사이의 가중치(w1~w4)를 이용하여 은닉층 가중치 합(z1, z2) 계산

(2)시그모이드 함수에 z값을 적용하여 h1, h2 출력값 계산

(3)은닉층 출력(h1, h2)과 출력층 사이의 가중치(w5~w8)로 출력층 가중치 합(z3,z4) 계산

(4)시그모이드 함수에 z값을 적용하여 o1, o2 출력값 계산

2) 역전파(BackPropagation)

출력층에서 입력층을 향하면서 오차를 계산하여 가중치를 수정

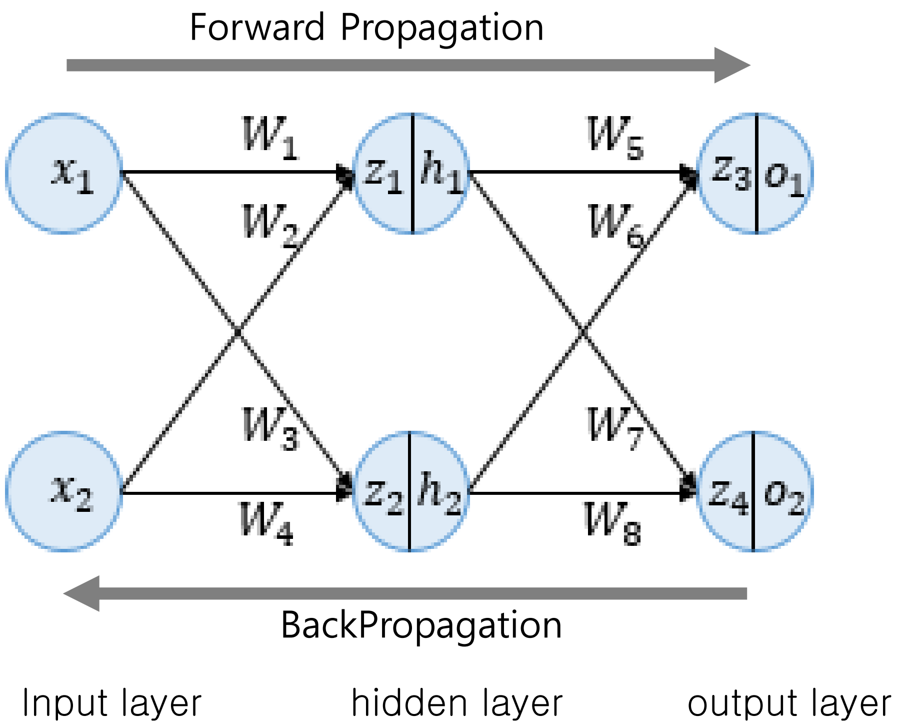
(1)출력층과 은닉층 사이의 가중치(w5~w8) 수정

$$E_{total} = \frac{1}{2}(target_{o1} - output_{o1})^2 + \frac{1}{2}(target_{o2} - output_{o2})^2$$

$$\frac{\partial E_{total}}{\partial W_5} = \frac{\partial E_{total}}{\partial o_1} \times \frac{\partial o_1}{\partial z_3} \times \frac{\partial z_3}{\partial W_5}$$

$$W_5^+ = W_5 - \alpha\frac{\partial E_{total}}{\partial W_5}$$

(2)은닉층과 입력층사이의 가중치 (w1~w4) 수정



https://wikidocs.net/37406