

ML/DL for Everyone Season2

with  TensorFlow

02 - Simple Linear Regression LAB

Code: <https://github.com/deeplearningzerotoall/TensorFlow>

Slides: <http://bit.ly/2LQMKvk>

Lecturer: 이승준 plusjune@financedata.kr



Hypothesis and Cost

Hypothesis

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

Cost

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

Build hypothesis and cost

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

```
x_data = [1, 2, 3, 4, 5]
y_data = [1, 2, 3, 4, 5]

W = tf.Variable(2.9)
b = tf.Variable(0.5)

# hypothesis = W * x + b
hypothesis = W * x_data + b
```

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

```
cost = tf.reduce_mean(tf.square(hypothesis - y_data))
```

Build hypothesis and cost

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

```
cost = tf.reduce_mean(tf.square(hypothesis - y_data))
```

- `tf.reduce_mean()`

```
v = [1., 2., 3., 4.]  
tf.reduce_mean(v) # 2.5
```

- `tf.square()`

```
tf.square(3) # 9
```

Gradient descent

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

```
# learning_rate initialize
learning_rate = 0.01

# Gradient descent
with tf.GradientTape() as tape:
    hypothesis = W * x_data + b
    cost = tf.reduce_mean(tf.square(hypothesis - y_data))

W_grad, b_grad = tape.gradient(cost, [W, b])

W.assign_sub(learning_rate * W_grad)
b.assign_sub(learning_rate * b_grad)
```

A.assign_sub(B)

A = A - B

A -= B

Parameter(W,b) Update

```
W = tf.Variable(2.9)
b = tf.Variable(0.5)

for i in range(100):
    # Gradient descent
    with tf.GradientTape() as tape:
        hypothesis = W * x_data + b
        cost = tf.reduce_mean(tf.square(hypothesis - y_data))
    W_grad, b_grad = tape.gradient(cost, [W, b])
    W.assign_sub(learning_rate * W_grad)
    b.assign_sub(learning_rate * b_grad)
    if i % 10 == 0:
        print("{:5}|{:10.4}|{:10.4}|{:10.6f}".format(i, W.numpy(), b.numpy(), cost))
```

Full Code (less than 20 lines)

```
import tensorflow as tf
tf.enable_eager_execution()

# Data
x_data = [1, 2, 3, 4, 5]
y_data = [1, 2, 3, 4, 5]

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

learning_rate = 0.01

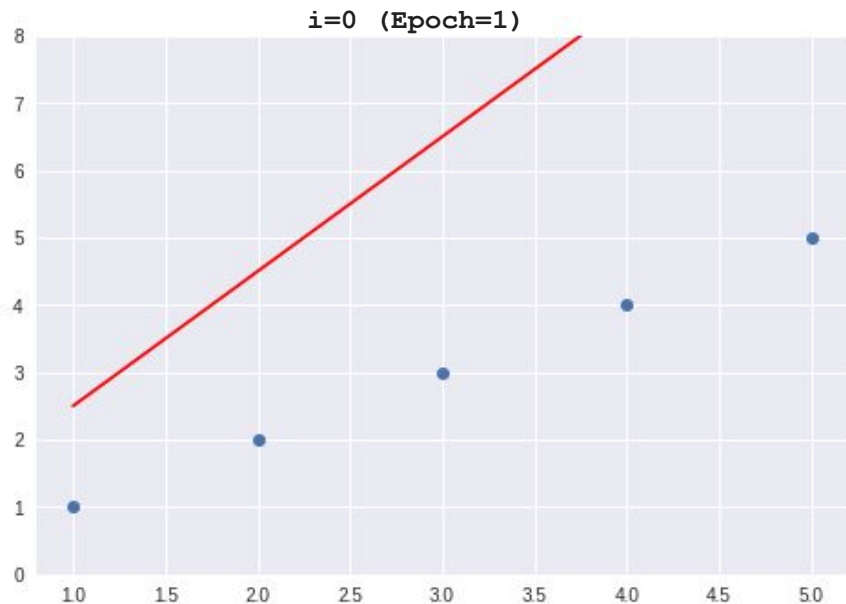
for i in range(100+1): # W, b update
    # Gradient descent
    with tf.GradientTape() as tape:
        hypothesis = W * x_data + b
        cost = tf.reduce_mean(tf.square(hypothesis - y_data))
    W_grad, b_grad = tape.gradient(cost, [W, b])
    W.assign_sub(learning_rate * W_grad)
    b.assign_sub(learning_rate * b_grad)
    if i % 10 == 0:
        print("{:5}|{:10.4f}|{:10.4f}|{:10.6f}".format(i, W.numpy(), b.numpy(), cost))
```

	i	W	b	cost
	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

Training

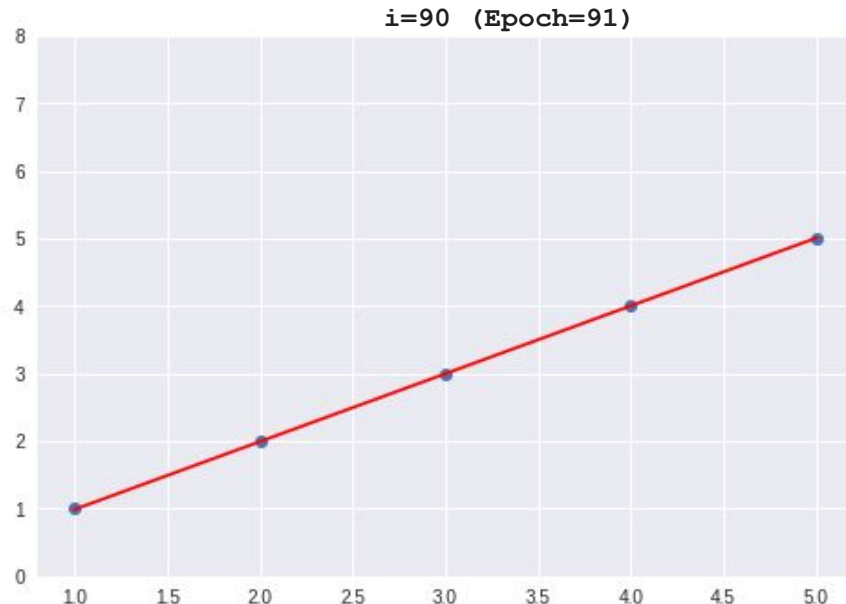
`W = tf.Variable(2.9)`

`b = tf.Variable(0.5)`



`W = 2.4520`

`b = 0.3760`



`W = 1.0050`

`b = -0.0179`

Predict

Hypothesis $H(x) = Wx + b$

Data

`x_data = [1, 2, 3, 4, 5]`

`y_data = [1, 2, 3, 4, 5]`

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

`tf.Tensor(5.0066934, shape=(), dtype=float32)`

`tf.Tensor(2.4946523, shape=(), dtype=float32)`

What's Next?

- How to minimize cost