ML_Lec 01.

About ML.

1. Limitations of Explicit Programming.
   Spam filter : many rules
   Automatic driving : too many rules → 고려할 사항이 너무 많음.

2. Machine Learning. //빨간색: 주요 개념
   "Field of study that gives computers the ability to learn without being explicitly programmed." Arthur Samuel(1959)

머신러닝은 일종의 소프트웨어로, 프로그램 자체가 학습하여 배우는 명령을 갖는 소프트웨어이다.
//분명하지 않은 경우 재정리

Learning.

1. Supervised Learning : Learning with labeled examples – Training Set

   Most common problem type in ML
   1) Image Labeling
   2) Email Spam Filter
   3) Predicting Exam Score

   Types of Supervised Learning
   1) 0~100 까지의 점수 Regression
   2) Pass/Non-pass Binary Classification
   3) A, B, C, D, E grade Multi-Label Classification

2. Unsupervised Learning : un-labeled data
   Google news grouping
   Word clustering

ML_Lab 01.

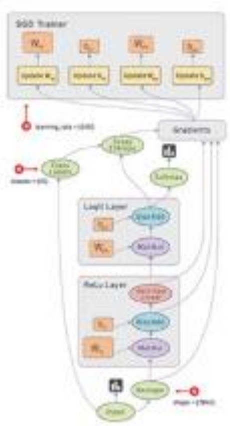
TensorFlow.

TensorFlow is an open source software library for numerical computation using data flow graphs.

Data Flow Graph.

Data Flow Graph 는 노드, 엣지로 연산이 일어나 어떤 작업을 할 수 있는 것이다.

1. Nodes in the graph represent mathematical operations. //연산
2. Edges represent the multidimensional data arrays(tensors) communicated between them. //Tensor



Tensor.

임의의 차원을 갖는 배열들을 의미한다.

1. Ranks : n 차원

Rank	Math entity	Python example
0	Scalar (magnitude only)	`s = 483`
1	Vector (magnitude and direction)	`v = [1.1, 2.2, 3.3]`
2	Matrix (table of numbers)	`m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	3-Tensor (cube of numbers)	`t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]`
n	n-Tensor (you get the idea)	`....`

2. Shapes : [가장 바깥 요소의 개수, 가장 안쪽 요소의 개수]

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

3. Types : 자료형

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.

Test.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
hello = tf.constant("Hello, Tensorflow!") #.constant()
sess = tf.Session() #.Session()
print(sess.run(hello)) #.run()
```

```
b'Hello, Tensorflow!'
```

```
node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0)
node3 = tf.add(node1, node2)  #.add()

sess = tf.Session()
print("sess.run(node1, node2): ", sess.run([node1, node2]))
print("sess.run(node3): ", sess.run(node3))
```

```
sess.run(node1, node2):  [3.0, 4.0]
sess.run(node3):  7.0
```

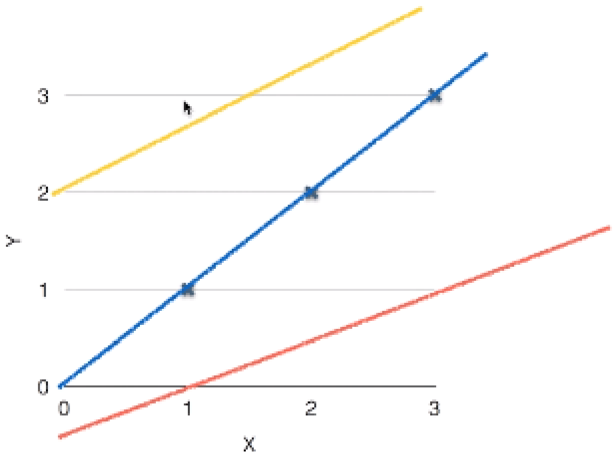
```
a_node = tf.placeholder(tf.float32) #.placeholeder()와 feed_dict={a:a_data}
b_node = tf.placeholder(tf.float32)
add_node = a_node + b_node

print(sess.run(add_node, feed_dict={a_node:2, b_node:13}))
```

```
15.0
```

ML_Lec 02.

Linear Regression. //선형 회귀

$\boldsymbol{H(x) = Wx + b}$



대략 직선(Linear) 형태에 맞을 것이라 가설하고 진행한다.

Cost function. //비용함수

1. How fit the line to our (training) data

   $(H(x) = -y)^2$ //± 때문에 제곱

   $$\frac{\left(H\left(x^{(1)}\right) - y^{(1)}\right)^2 + \left(H\left(x^{(2)}\right) - y^{(2)}\right)^2 + \left(H\left(x^{(3)}\right) - y^{(3)}\right)^2}{3}$$

   $$\rightarrow\ Cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2\ \ (m: \textit{데이터 수})$$

2. Goal : Minimize Cost

   minimize $Cost(W,b)$

ML_Lab 02.

Test.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
#H(x) = ₩x + b
x_train = [1, 2, 3]
y_train = [1, 2, 3]
w = tf.Variable(tf.random_normal([1]), name='weight') #.Variable(): 변수
b = tf.Variable(tf.random_normal([1]), name='bias') #.random_normal(Shapes)
hypothesis = x_train * w + b

#Cost(₩,b)=∑...
cost = tf.reduce_mean(tf.square(hypothesis - y_train)) #.reduce_mean(): 평균 #.square()

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

sess = tf.Session()
sess.run(tf.global_variables_initializer()) #.global_variables_initializer()

for step in range(2001):
  sess.run(train)
  if step % 20 == 0:
    print("step:", step, "cost:", sess.run(cost), "w:", sess.run(w), "b:", sess.run(b))
```

```
step: 0 cost: 1.6658787 w: [1.1718934] b: [-1.6268224]
step: 20 cost: 0.2775497 w: [1.5546099] b: [-1.3784562]
step: 40 cost: 0.24087375 w: [1.5657564] b: [-1.2973045]
```

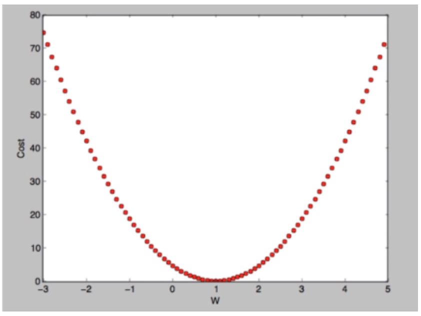
```
#H(x) = ₩x + b
x = tf.placeholder(tf.float32, shape=[None])
y = tf.placeholder(tf.float32, shape=[None])
w = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
hypothesis = x * w + b

#Cost(₩,b)=∑...
cost = tf.reduce_mean(tf.square(hypothesis - y))

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

sess = tf.Session()
sess.run(tf.global_variables_initializer())

for step in range(2001):
  cost_val, w_val, b_val, _ = sess.run([cost, w, b, train],
      feed_dict={x: [2, 2.1, 2.2], y: [11, 12, 13]})
  if step % 20 == 0:
    print("step:", step, "cost:", cost_val, "w:", w_val, "b:", b_val)

#11.97 = 2.1 * 4.7 + 2.1
#y = x * w + b
```

```
step: 0 cost: 136.22298 w: [0.3994273] b: [0.78138727]
step: 20 cost: 1.5937748 w: [4.030239] b: [2.5011532]
step: 40 cost: 0.22041719 w: [4.399387] b: [2.6696613]
```

ML_Lec 03.

Simplified Hypothesis.

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



Minimize. //최소화

Cost 가 제일 낮은 값을 찾는 것이다.

Gradient Descent Algorithm. //경사하강법

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

1. Minimize cost function.
2. Gradient descent is used many minimization problems.
3. For a given cost function, cost(W, b), it will find W, b to minimize cost.
4. It can be applied to more general function: cost(w1, w2, …).

Gradient Descent Algorithm Works.

1. Start with initial guesses
   1) Start at 0,0 (or any other value)
   2) Keeping changing W and b a little bit to try and reduce cost(W, b)
2. Each time you change the parameters, you select the gradient which reduces cost(W, b) the most possible

3. Repeat
4. Do so until you converge to a local minimum
5. Has an interesting property
    - ♠ Where you start can determine which minimum you end up

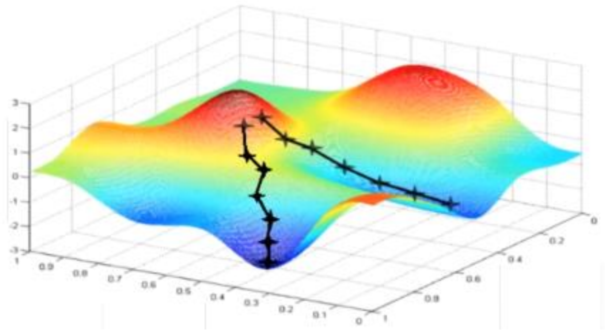
Formal Definition. //미분

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

$$W := W - \alpha\frac{\partial}{\partial W}Cost(W)\ (\alpha = 0.1)$$

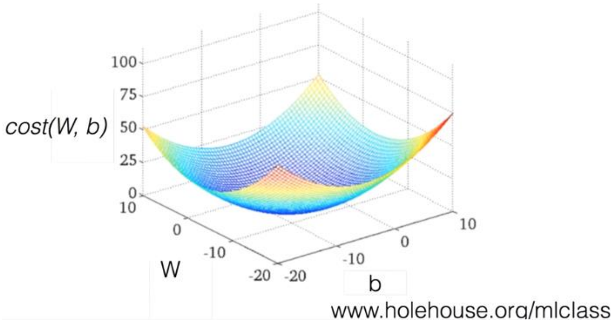
$$\rightarrow\ W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(Wx^{(i)} - y^{(i)}\right)x^{(i)}$$

Convex Function. //볼록함수



Cost W b

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



ML_Lab 03.

Test.

```python
x = [1, 2, 3]
y = [1, 2, 3]
W = tf.placeholder(tf.float32)
hypothesis = x * W

cost = tf.reduce_mean(tf.square(hypothesis - y))

sess = tf.Session()
sess.run(tf.global_variables_initializer())

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) #.append()
  cost_val.append(curr_cost)

plt.plot(W_val, cost_val) #.plot()
plt.show() #.show()
```


```python
x_data = [1, 2, 3]
y_data = [1, 2, 3]
W = tf.Variable(tf.random_normal([1]), name='weight')
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
hypothesis = x * W

cost = tf.reduce_sum(tf.square(hypothesis -y)) #.reduce_sum()

learning_rate = 0.1
gradient = tf.reduce_mean((W * x - y) * x)
descent = W - learning_rate * gradient
update = W.assign(descent) #.assign()

sess = tf.Session()
sess.run(tf.global_variables_initializer())

for step in range(21):
  sess.run(update, feed_dict={x: x_data, y: y_data})
  print("step:", step, "cost:", sess.run(cost, feed_dict={x: x_data, y: y_data}), "W:", sess.run(W))
```

```
step: 0 cost: 18.648413 W: [-0.15413582]
step: 1 cost: 5.3044367 W: [0.38446093]
step: 2 cost: 1.5088172 W: [0.6717125]
step: 3 cost: 0.42917478 W: [0.8249133]
step: 4 cost: 0.12207642 W: [0.90662044]
step: 5 cost: 0.034723975 W: [0.9501976]
step: 6 cost: 0.009877007 W: [0.97343874]
step: 7 cost: 0.0028094668 W: [0.985834]
step: 8 cost: 0.00079912913 W: [0.9924448]
```

```python
x = [1, 2, 3]
y = [1, 2, 3]
W = tf.Variable(5.0)
hypothesis = x * W

cost = tf.reduce_mean(tf.square(hypothesis -y))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train = optimizer.minimize(cost)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

for step in range(100):
  print("step:", step, "W:", sess.run(W))
  sess.run(train)
```

```
step: 0 W: 5.0
step: 1 W: 1.2666664
step: 2 W: 1.0177778
step: 3 W: 1.0011852
step: 4 W: 1.000079
step: 5 W: 1.0000052
step: 6 W: 1.0000004
step: 7 W: 1.0
step: 8 W: 1.0
step: 9 W: 1.0
step: 10 W: 1.0
step: 11 W: 1.0
```

```python
x = [1, 2, 3]
y = [1, 2, 3]
W = tf.Variable(5.0)
hypothesis = x * W
gradient = tf.reduce_mean((W * x - y) * x) * 2

cost = tf.reduce_mean(tf.square(hypothesis -y))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
gvs = optimizer.compute_gradients(cost, [W]) #.compute_gradients()
apply_gradients = optimizer.apply_gradients(gvs) #.apply_gradients()

sess = tf.Session()
sess.run(tf.global_variables_initializer())

for step in range(100):
  print(step, sess.run([gradient, W, gvs]))
  sess.run(apply_gradients)
```

```
0 [37.333332, 5.0, [(37.333336, 5.0)]]
1 [33.84889, 4.6266665, [(33.84889, 4.6266665)]]
2 [30.689657, 4.2881775, [(30.689657, 4.2881775)]]
3 [27.825289, 3.981281, [(27.825289, 3.981281)]]
4 [25.228264, 3.7030282, [(25.228264, 3.7030282)]]
5 [22.873625, 3.4507456, [(22.873627, 3.4507456)]]
6 [20.738754, 3.2220094, [(20.738754, 3.2220094)]]
7 [18.803139, 3.014622, [(18.803139, 3.014622)]]
8 [17.04818, 2.8265905, [(17.04818, 2.8265905)]]
9 [15.457016, 2.6561089, [(15.457016, 2.6561089)]]
```

Memo

.constant()	//상수
.Session()	//tensor 에 데이터를 넣어 흐르게 함.
.run()	//실행
.add()	//더하기
.placeholeder(), feed_dict={a:a_data}	//변수, 값을 나중에 할당.
.Variable()	//변수, 자동으로 업데이트.
.random_normal(Shapes)	//랜덤 값 반환.
.reduce_mean()	//평균
.square()	//제곱
.GradientDescentOptimizer()	//미니 배치 확률적 경사하강법(SGD) 구현.
.minimize()	//최소화
.global_variables_initializer()	//.Variable()를 초기화.
.append()	//append
.plot()	//plot
.show()	//show
.reduce_sum()	//총합
.assign()	//.Variable()의 값 변경.
.compute_gradients()	//compute_gradients
.apply_gradients()	//apply_gradients

//Lab 으로 연결 https://colab.research.google.com/drive/1gaTpEufmhoK2CsEsNyfDDtyynQ_HRpSu

//14 폰트, 12 폰트, 10 폰트

//1. 1) a. *♠

//0.71 1.34