Machine Learning in Practice
#2-2: TensorFlow Basics
"*First Contact with TensorFlow*", Ch. 2-3

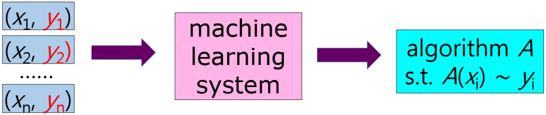
Sang-Hyun Yoon

Summer 2019

Recall: Training data 획득 방식에 따른 분류

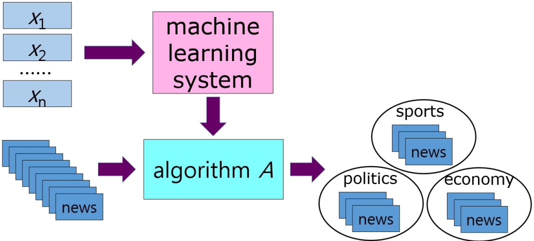
Supervised learning (지도 학습) (가장 널리 사용)

- Input과 output을 모두 주고 최적의 $\mathcal{A} \in \mathcal{H}$를 찾음
- Classification, **regression** 문제에 주로 사용



Unsupervised learning (비지도 학습)

- Input에 대응되는 output이 주어지지 않는 경우
- **Clustering** 문제, feature 추출, 차원 줄이기에 사용



Example Problems for TensorFlow API

Linear Regression Problem (a supervised learning problem, P)

- Input: $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\} \subseteq \mathbb{R}^2$
  - i.e. set of points in the plane
- Output: $W, b \in \mathbb{R}$ that minimizes $\boxed{\sum_{i=1}^{n}(W \cdot x_i + b - y_i)}$
  - i.e. line $y = Wx + b$ that best fits points of T

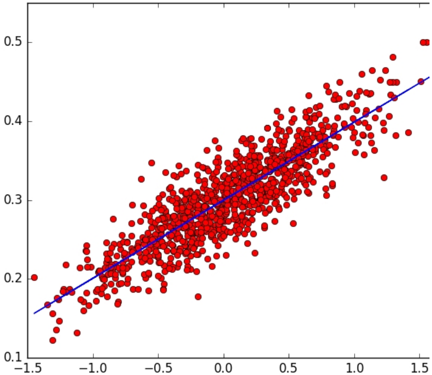
k-Clustering Problem (an unsupervised one, NP-hard)

- Input: $T \subseteq \mathbb{R}^2$ (i.e. set of points in the plane)
- Output: k points $p_1, p_2, \cdots, p_k \in \mathbb{R}^2$ that minimizes
$$\boxed{\max_{p \in T} \min_{1 \le k \le n} dist(p, p_k)}$$
  - each p belongs to cluster with center p_k with min $dist(p, p_k)$

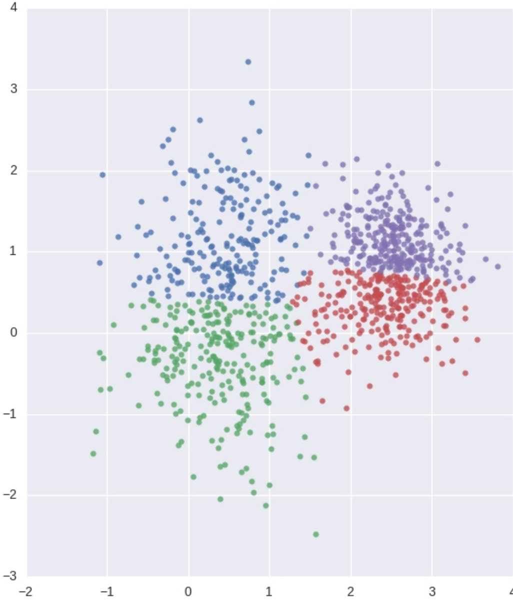
TensorFlow의 API를 이용해서 (Python API, C++ API)

- $\sum_{i=1}^{n}(W \cdot x_i + b - y_i)$ 와 같은 cost function을 표현하고
- 이 cost function을 **min/maximize**할 수 있음 (이게 중요)

linear regression



k-clustering



Outline

TensorFlow Code for Linear Regression Problem (Ch. 2)

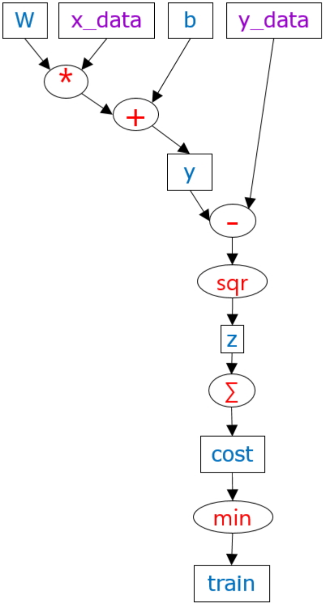
- Input: $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\} \subseteq \mathbb{R}^2$
- Output: $W, b \in \mathbb{R}$ that minimizes $\boxed{\sum_{i=1}^{n} (W \cdot x_i + b - y_i)}$

```
import tensorflow as tf

T = ...
x_data, y_data = [p[0] for p in T], [p[1] for p in T]

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b            # symbolic computation (graph)

cost = tf.reduce_sum(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(cost)

sess = tf.Session()  ...
for step in range(100): sess.run(train)  # actual computation
```

Computation Graph

Find W, b that minimizes $\sum_{i=1}^{n}(W \cdot x_i + b - y_i)$

```
x_data, y_data = ...

W = tf.Variable(...)
b = tf.Variable(...)
y = W * x_data + b   # symbolic computation

z = tf.square(y - y_data)
cost = tf.reduce_sum(z)
optimizer = tf.train.GradientDescentOptimizer
train = optimizer.minimize(cost)

sess = tf.Session()  ...
for step in range(100):
    sess.run(train)  # actual computation
```



Symbolic Computation / Numerical Optimization

```
x_data, y_data = ...
W = tf.Variable(...)
b = tf.Variable(...)
y = W * x_data + b   # symbolic computation

z = tf.square(y - y_data)
cost = tf.reduce_sum(z)
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(cost)

sess = tf.Session()  ...
for step in range(100):
    sess.run(train)  # actual computation
```

- `train` 까지는 symbolic 계산만 이루어짐 (graph 구성)
  - 사용자는 텐서플로 API를 활용하여 이 부분만 채우면 됨!
- `run(·)`에 와서야 실제로 계산 (numerical optimization)
  - 이의 내부가 가장 복잡한데 사용자는 불러 쓰기만 하면 됨!
- `tf.Variable`로 선언된 `W`, `b`을 변화시키면서 최적화 계산
  - `x_data`, `y_data`는 텐서플로 변수가 아니므로 건드리지 않음

Outline

Tensor Data Types (tensor_type.py)

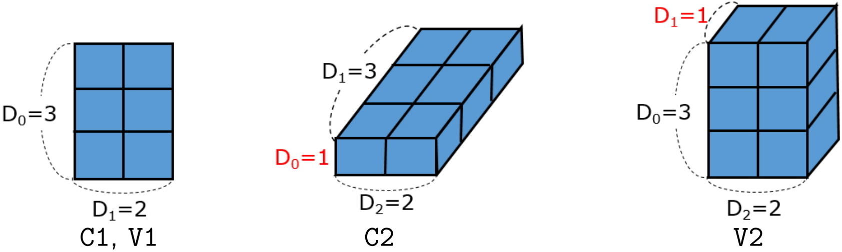
- TensorFlow의 모든 데이터는 Tensor(텐서) type
- 텐서는 multi-dimensional array로 보면 됨
- 텐서의 차원/크기는 동적으로 변경 가능
- 텐서의 차원을 rank로 부름 (예: rank 2은 행렬)
- 각 차원의 크기를 나열한 것을 shape로 부름
- 아래와 같이 constant(·)과 variable(·) 함수로 텐서 생성

```
P = [ [1,3], [2,2], [4,6] ]  # 3-by-2 list
C = tf.constant(P)    # tensor constant
V = tf.Variable(P)    # tensor variable

print(C)  # Tensor("Const:0", shape=(3,2), dtype=int32)
print(V)  # <tensorflow.python.ops.variables.Variable obj ..
print(C.get_shape(), V.get_shape())  # (3,2) (3,2)
>> C.get_shape() # TensorShape([Dimension(3), Dimension(2)])
```

- P = [[1,3], [2,2], [4]]는 텐서 변환 불가

Shapes of Tensors (tensor_type.py)

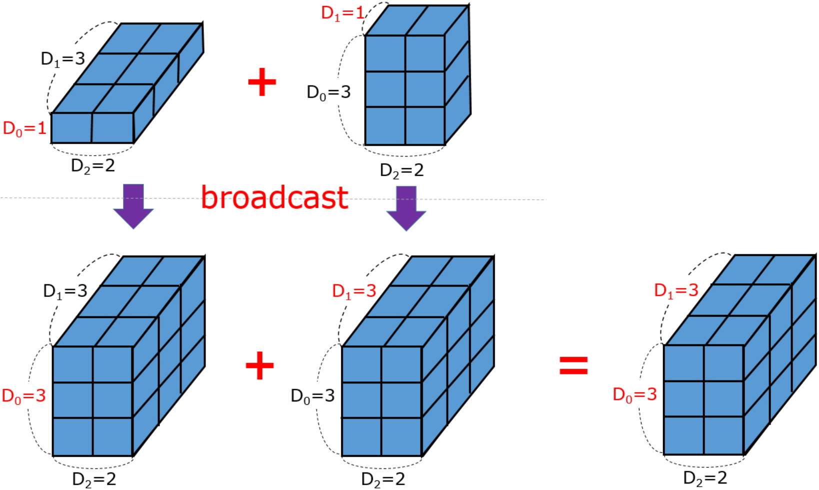
```
P = [ [1,3], [2,2], [4,6] ]  # 3x2 list
C1 = tf.constant(P)    # tensor constant
V1 = tf.Variable(P)    # tensor variable
print(C1.get_shape(), V1.get_shape())  # (3,2) (3,2)

C2 = tf.expand_dims(C1, 0)
V2 = tf.expand_dims(V2, 1)
print(C2.get_shape(), V2.get_shape())  # (1,3,2) (3,1,2)
```



- 크기가 1인 dimension은 broadcast로 자동으로 커질 수 있음

Shapes of Tensors: Broadcast

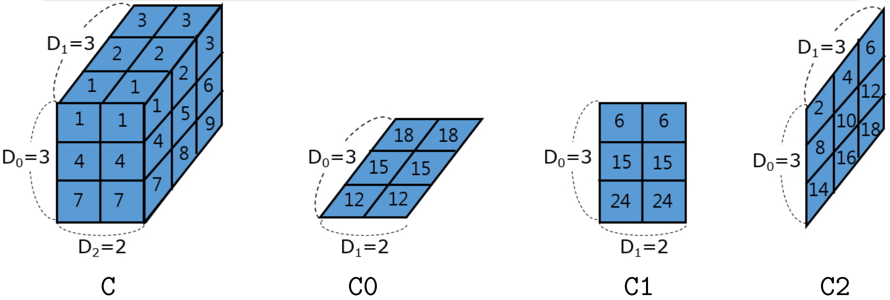
- 크기가 1인 dimension은 broadcast로 자동으로 커질 수 있음
- 차원의 크기가 늘어날때 데이터들이 그대로 복사되는 효과
- numpy의 broadcast와 같은 방식



Shapes of Tensors: Dimension Reduction (reduce.py, pp.83-84)

연산(sum, prod, min, max, mean)을 하면서 지정한 차원을 감소

```
C = tf.constant([ [[1,1], [2,2], [3,3]], \
                  [[4,4], [5,5], [6,6]], \
                  [[7,7], [8,8], [9,9]] ])
C0 = tf.reduce_sum(C, 0) # [[12,12], [15,15], [18,18]]
C1 = tf.reduce_sum(C, 1) # [[6,6], [15,15], [24,24]]
C2 = tf.reduce_sum(C, 2) # [[2,4,6], [8,10,12], [14,16,18]]
C3 = tf.reduce_sum(C)    # 90 (scalar)
C4 = tf.argmin(C2, 1)    # [0,0,0]
```



Creating Tensors with Initialization (Python API) (tensor_init.py)

Constant tensor를 생성/초기화

```
C1 = tf.zeros([3,4], tf.int32)  # [[0,0,0,0],[0,0,0,0],[0,0,0,0]
C2 = tf.zeros_like([[1,2,3], [4,5,6]])  # [[0,0,0], [0,0,0]]
C3 = tf.fill([2,3], 9)  # [[9,9,9], [9,9,9]]
C4 = tf.constant([[1,3],[2,2],[4,6]])  # [[1,3],[2,2],[4,6]]

C5 = tf.random_uniform([2,3], -1,1)
C6 = tf.random_normal([2,3], 5,2)
C7 = tf.random_shuffle(C4)  # shuffles C4 along dimension 0

sess = tf.Session()
print(sess.run(C7), type(C7))
```

Variable tensor를 constant tensor값으로 생성/초기화

```
V = tf.Variable(C4)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
print(sess.run(V), type(V))
```

Symbolic Variable: placeholder (placeholder.py)

```
import tensorflow as tf
import numpy as np

x = tf.placeholder(tf.float32, shape=(2,2))
y = tf.matmul(x, x)

sess = tf.Session()
print(sess.run(y)) # ERROR: will fail because x was not fed

rand_array = np.random.rand(2,2)
print(sess.run(y, feed_dict={x:rand_array}))
```

- Computation graph의 leaf에 들어갈 값을 미리 초기화하지 않고 sess.run(·)을 하는 시점에서 설정할 경우 유용
- sess.run(·)을 부를 때 feed_dict dictionary에서 초기값을 설정

Slicing: `slice` (slice.py)

numpy의 slicing과 같은 방식

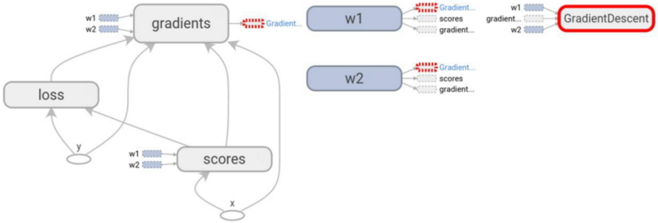
```
import tensorflow as tf

C = tf.constant([ [[1,1,1], [2,2,2]], \
                  [[3,3,3], [4,4,4]], \
                  [[5,5,5], [6,6,6]] ])

C1 = tf.slice(C, [1,0,0], [1,1,3]) # [[[3,3,3]]]
C2 = tf.slice(C, [1,0,0], [1,2,3]) # [[[3,3,3], [4,4,4]]]
C3 = tf.slice(C, [1,0,0], [2,1,3]) # [[[3,3,3]], [[5,5,5]]]

sess = tf.Session()
print(sess.run(C1), sess.run(C2), sess.run(C3))
```

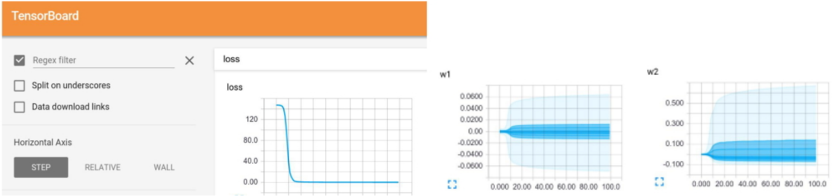
그의 tensor에 대한 유용한 operation/function들은 교과서 pp.42-43, p.45 참고

Outline

TensorBoard

- Computation graph와 최적화 과정을 시각화
- 약간의 annotation을 코드에 추가해주면 TensorBoard의 입력 data가 생성됨
- http://tensorflowstepbystep.tistory.com/5 참조





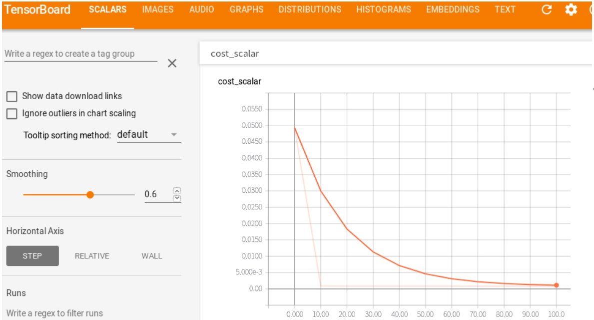
Annotation for TensorBoard (line_tensorboard.py)

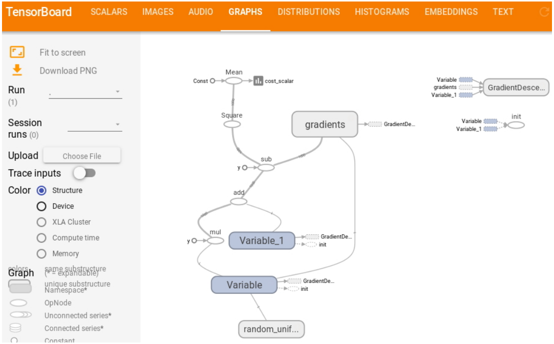
```
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
add = X+Y
mul = X*Y

# step 1: select node
add_hist = tf.summary.scalar("add_scalar", add)
mul_hist = tf.summary.scalar("mul_scalar", mul)

# step 2: collect summary
merged = tf.summary.merge_all()
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# step 3: generate writer
writer = tf.summary.FileWriter("./line_tboard", sess.graph)

for step in range(100):
    # step 4: add node
    summary = sess.run(merged, feed_dict={X:step*1.0, Y:2.0})
    writer.add_summary(summary, step)
```

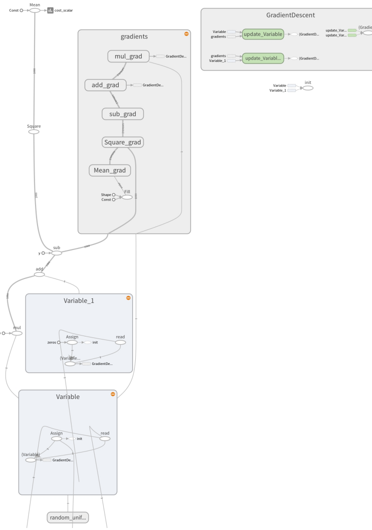
Annotation for TensorBoard (regression_tensorboard.py)

```
cost = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(cost)

cost_hist = tf.summary.scalar("cost_scalar", cost)
merged = tf.summary.merge_all()

   ...

writer = tf.summary.FileWriter("./regression_tboard", sess.graph)

for step in xrange(101):
    sess.run(train)  # minimize cost
    summary = sess.run(merged)
    writer.add_summary(summary, step)
```

1. 터미널에서 "python regression_tensorboard.py" 수행
2. "tensorboard --logdir=reg.._tboard" 수행한 상태에서
3. FireFox에서 주소 127.0.1.1:6006로 접속


TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
TEXT
Write a regex to create a tag group
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
cost_scalar
cost_scalar
0.0550
0.0500
0.0450
0.0400
0.0350
0.0300
0.0250
0.0200
0.0150
0.0100
5.000e-3
0.00
0.000
10.00
20.00
30.00
40.00
50.00
60.00
70.00
80.00
90.00
100.0


TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
TEXT
Fit to screen
Download PNG
Run (1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
Graph
OpNode
Unconnected series*
Connected series*
Constant
Const
Mean
cost_scalar
Square
gradients
GradientDe...
sub
y
add
mul
Variable_1
init
Variable
random_unif...
GradientDesce...
Variable
gradients
Variable_1


gradients
mul_grad
add_grad
sub_grad
Square_grad
Mean_grad
GradientDescent
update_Variable
Variable_1
Variable
random_unif...

Outline

k-Clustering

k-clustering problem (an unsupervised one, NP-hard)

- Input: $T \subseteq \mathbb{R}^2$ (i.e. set of points in the plane)
- Output: k points $p_1, p_2, \cdots, p_k \in \mathbb{R}^2$ that minimizes

$$\boxed{\max_{p \in T} \min_{1 \leq k \leq n} dist(p, p_k)}$$

  - each p belongs to cluster with center p_k with min $dist(p, p_k)$

- Output은 centroid라고 불리는 k개의 각 cluster의 중심점
- 각 input 점은 하나의 cluster에만 속함 (최단거리 centroid)
- k-clustering 문제는 NP-hard로 가장 널리 사용되는 휴리스틱은 k-means algorithm

k-means algorithm

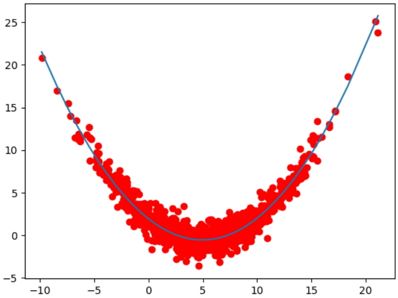
1. 입력으로 들어온 점들 중 임의의 k를 centroid로 초기화
2. 각 점들을 가장 가까운 centroids의 cluster에 할당
3. 각 cluster에 대해 새로운 centroid를 계산 (cluster의 점들의 평균값으로). 2번 스텝으로 가서 반복

TensorFlow Code for k-Means Algorithm (Ch. 3)

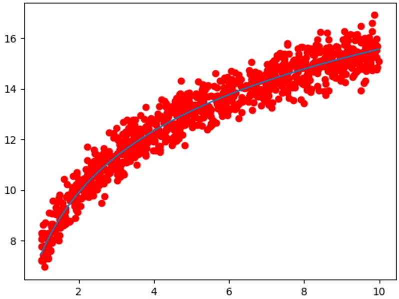
```
centroids = tf.Variable(tf.slice(tf.random_shuffle(vectors),
                                  [0,0], [num_clusters,-1]))
expanded_vectors = tf.expand_dims(vectors, 0)
expanded_centroids = tf.expand_dims(centroids, 1)

distances = tf.reduce_sum(
  tf.square(tf.sub(expanded_vectors, expanded_centroids)), 2)
assignments = tf.argmin(distances, 0)

means = tf.concat(0, [
  tf.reduce_mean(
      tf.gather(vectors,
                tf.reshape(tf.where(tf.equal(assignments, c)), [
               ),reduction_indices=[1])
  for c in xrange(num_clusters)])
update_centroids = tf.assign(centroids, means)
...
for step in xrange(num_steps):
   _, centroid_values, assignment_values = sess.run([update_cent
                                                      centroids,
```

Homework: Quadratic/Logarithmic Regression

Quadratic regression (regression_quadratic_hw.py)



Logarithmic regression (regression_log_hw.py)



- regression.py를 정확히 이해한 후 그대로 따라하면 됨
- Optimizer를 GradientDescentOptimizer(0.5)로 그래도 쓰면 수렴하지 않음을 확인할 수 있음
- 대신 AdamOptimizer(0.5)를 사용함을 시도
- np.log에 대응되는 TensorFlow 함수는 tf.log
    - tf.log도 다른 함수들처럼 임의의 차원에 대해 작동