**Lab 1 – TensorFlow Basics**

TensorFlow

* TensorFlow™ is an open source software library for numerical computation using data flow graphs.
* Python을 사용한다.

Data flow graph

* Nodes in the graph represent mathematical operations.
* Edges represent the multidimensional data arrays (tensors) communicated between them.

Hello World 출력

* hello = tf.constant("Hello, TensorFlow!")
  + hello라는 constant node를 하나 만든 것이다.
* sess = tf.Session()
  + TF session을 하나 만들었다.
* print(sess.run(hello))
  + 값을 출력한다.

Computational Graph

* 덧셈을 수행하는 연산이다.
  + node1 = tf.constant(3.0, tf.float32)

node2 = tf.constant(4.0) # also tf.float32 implicitly

node3 = tf.add(node1, node2)

print("node1:", node1, "node2:", node2)

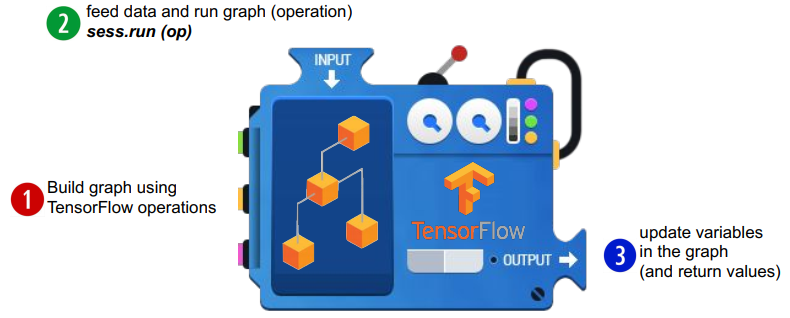
print("node3: ", node3)

sess = tf.Session()

print("sess.run(node1, node2): ", sess.run([node1, node2]))

print("sess.run(node3): ", sess.run(node3))

TensorFlow의 작동 원리



Placeholder

* node를 만드는 시점에서 값을 미리 지정하지 않고 변수처럼 나중에 지정하고 싶은 경우
* placeholder라는 기능을 사용한다.
  + a = tf.placeholder(tf.float32)

b = tf.placeholder(tf.float32)

adder\_node = a + b # + provides a shortcut for tf.add(a, b)

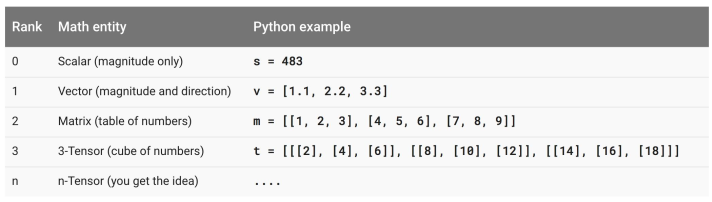
print(sess.run(adder\_node, feed\_dict={a: 3, b: 4.5}))

print(sess.run(adder\_node, feed\_dict={a: [1,3], b: [2, 4]}))

* 위의 feed\_dict가 값을 넘겨주는 기능을 한다.

Tensor

* Tensor를 사용해 연산을 진행한다.
  + 3 # a rank 0 tensor; this is a scalar with shape []
  + [1. ,2., 3.] # a rank 1 tensor; this is a vector with shape [3]
  + [[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
  + [[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
* Tensor rank



* Tensor shape



**Lab2 – Linear Regression**

Hypothesis & cost function

Building graph

* Variable
  + 기존의 프로그램에서 언급하는 변수와는 조금 다르다.
  + TF에서 사용하는 변수의 개념이라고 생각하자.
* x\_train = [1, 2, 3]

y\_train = [1, 2, 3]

W = tf.Variable(tf.random\_normal([1]), name="weight") # 1-dimensional array

b = tf.Variable(tf.random\_normal([1]), name="bias")

* + 위의 경우는 x와 y의 값을 직접 지정해 넘겨주는 방식이다.
* hypothesis = x\_train \* W + b

cost = tf.reduce\_mean(tf.square(hypothesis - y\_train))

train = tf.train.GradientDescentOptimizer(learning\_rate=0.01).minimize(cost) #optimizer

# Launch the graph in a session.

with tf.Session() as sess:

# Initializes global variables in the graph.

sess.run(tf.global\_variables\_initializer())

# Fit the line

for step in range(2001):

\_, cost\_val, W\_val, b\_val = sess.run([train, cost, W, b])

if step % 20 == 0:

print(step, cost\_val, W\_val, b\_val)

* + 전역 변수를 먼저 초기화한 뒤, 2000까지 training을 진행한 후 진행 과정을 일정 분기마다 출력한다.

Placeholders

* 우리가 앞서 배운 개념 중에는 placeholder라는 개념이 있다. 이를 이용해 x와 y의 값을 미리 지정하지 않고도 이용이 가능하다.
* 학습이 진행될 순간에 data를 주는 것이 가능하다는 장점이 있다.

**Lab 3 – Minimizing Cost**

Hypothesis and cost function

* Simplified hypothesis:

Matplotlib

* 그래프를 시각화해서 보여주는 python library이다.

Gradient

* Gradient를 직접 설정하는 것이 가능하다.
  + learning\_rate = 0.1

gradient = tf.reduce\_mean((W \* X - Y) \* X)

descent = W - learning\_rate \* gradient

update = W.assign(descent)

* 하지만 TF에서 제공하는 optimizer를 사용해도 상관 없다.
  + train = tf.train.GradientDescentOptimizer(learning\_rate=0.1).minimize(cost)
* 경우에 따라, TF에서 제공하는 gradient와 자신이 직접 만든 gradient를 비교하고 싶은 경우가 있다. 이 경우 TF에서 제공하는 gradient를 받을 수 있다.
  + cost = tf.reduce\_mean(tf.square(hypothesis - Y))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=0.01)

# Get gradients of automatically calculated value by TF

# using to check difference with manual gradient

gvs = optimizer.compute\_gradients(cost)

**Lab 4 - Multi-variable linear regression**

Hypothesis using matrix

* Variable의 수가 많을수록 y를 예측하는 힘이 강해진다(reduction power가 강해진다).

Variables

* 변수는 각각 하나씩 지정해도 가능하지만, 그 수가 많아지면 개별적으로 지정하는 것이 비효율적이고 코드도 복잡해진다.
* 이 경우 사용하는 것이 array이다.
* 또는, file(.csv 등)로 저장되어 있는 data를 load 해오는 것 역시 가능하다.

Queue runners

* File이 너무 크고 많아서 한 번에 메모리에 올려서 분석하기 힘든 경우 사용하는 TF의 기능이다.
* 여러 파일명을 읽어와서 queue에 쌓은 다음, reader로 연결해 값을 읽어온 뒤 example queue에 쌓는다.
* 과정은 다음의 3단계이다.
  + 읽어올 file 이름들을 지정한다.
  + File을 읽을 reader(text line, binary line 등)를 지정한다.
  + 데이터 파싱 방법을 지정한다. Csv 파일의 경우 comma로 값이 구분되며, 값이 없는 경우를 대비해 default value를 지정할 수도 있다.

**Lab 5 – Logistic regression classifier**

Hypothesis & cost function

* Logistic hypothesis
  + (sigmoid function)
* Cost function

Training data

* x: multi-variable data
* y: 0 or 1 data

**Lab 6 - Softmax Classifier**

Softmax function

* XW = y (score)
* score 형태로 나온 값을 normalization하는 함수이다.
* TF에서는 tf.nn.softmax 라는 함수로서 사용 가능하다.
* Source code
  + X = tf.placeholder("float", [None, 4])

Y = tf.placeholder("float", [None, 3])

nb\_classes = 3 # number of classes

W = tf.Variable(tf.random\_normal([4, nb\_classes]), name='weight')

b = tf.Variable(tf.random\_normal([nb\_classes]), name='bias')

# softmax = exp(logits) / reduce\_sum(exp(logits), dim)

hypothesis = **tf.nn.softmax**(tf.matmul(X, W) + b)

cost = tf.reduce\_mean(-tf.reduce\_sum(Y \* tf.log(hypothesis), axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=0.1).minimize(cost)

softmax\_cross\_entropy\_with\_logits

* 위에서 직접 cross entropy를 식으로서 작성해줄 수도 있지만, TF에서 이를 자동으로 구현하는 기능이 있다.
* 위에서 Y는 기본적으로 one hot 상태로 제공된다. 위의 코드를 좀 더 fancy하게 바꾸면 다음과 같다.
  + logits = tf.matmul(X, W) + b

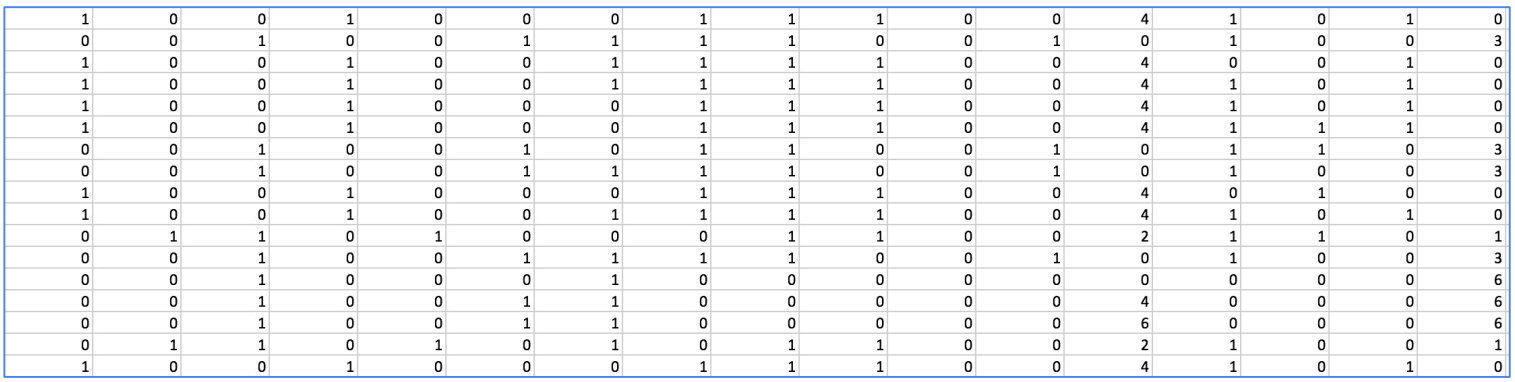
hypothesis = tf.nn.softmax(logits)

# Cross entropy cost/loss

cost = tf.reduce\_mean(**tf.nn.softmax\_cross\_entropy\_with\_logits\_v2**(logits = logits, labels = tf.stop\_gradient([Y\_one\_hot])))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=0.1).minimize(cost)

tf.one\_hot & reshape



* 데이터셋이 위와 같다. 여기서 class는 가장 오른쪽 열이다.
* 현재 Y는 단순히 csv 파일로 읽은 후에 [[0],[3],[0],[0], …]와 같은 꼴이다. 우리는 이를 최종적으로 [[1,0,0,0,0,0,0], [0,0,0,1,0,0,0], …]와 같은 형태로 만들고 싶다.
  + nb\_classes = 7

Y = tf.placeholder(tf.int32, [None, 1]) # 0~6, shape = (?, 1)

Y\_one\_hot = tf.one\_hot(Y, nb\_classes) # one hot, shape = (?, 1, 7)

Y\_one\_hot = tf.reshape(Y\_one\_hot, [-1, nb\_classes]) # shape = (?, 7)

* 처음 one\_hot을 적용하게 되면 모양이 shape = (?, 1, 7)과 같이 3차원이 나오게 된다. 이는 input의 rank가 N이면 output으로 rank가 N+1인 것을 반환하기 때문이다. 따라서 재조정이 필요하다.
* reshape를 적용하게 되면 우리가 원하는 모양대로 바뀌게 된다. 여기서 첫 번째의 -1은 everything이다.