

TensorFlow

안남혁

딥러닝 라이브러리

Caffe
(UC Berkeley)



Caffe2
(Facebook)

Torch
(NYU)



PyTorch
(Facebook)

Theano
(U Montreal)



TensorFlow
(Google)

*** Torch 빼고 전부 파이썬 지원!!**

PyTorch vs. TensorFlow

PyTorch:

- (+) 빠름
- (+) 파이썬스러운 코드
- (+) 다이내믹 그래프
- (-) 베타 버전.. (v0.12)
- (-) 아직 추가될 모듈이 많음

-> 연구용

TensorFlow:

- (+) 다양한 디바이스에 포팅 가능
- (+) 시각화 툴 (TensorBoard)
- (+) 많은 사용자
- (-) 정적 그래프
- (-) 어려운 디버깅
- (+++) 구글!!

-> 실제 프로젝트, 프로덕트용

왜 딥러닝 라이브러리를 써야 하나요?

- 딥러닝은 수학 계산, 특히 미분 계산이 많음
- 딥러닝 라이브러리의 특징
 - 수치 계산이 쉬움
 - **자동 미분 계산**
 - GPU에서 연산 가능 (numpy는 CPU만 가능)

Tensor

- 수학, 물리학에서 사용되는 용어
- 간단하게 행렬, 벡터를 일반화한 꼴이라 볼 수 있음
(물론 엄밀한 정의는 아니지만..)
- 0차 텐서: 스칼라, 1차 텐서: 벡터, 2차 텐서: 행렬 ...
- 그래서, numpy의 **ndarray**와 비슷함

numpy 예제

```
>>> import numpy as np
```

```
>>> a = np.zeros((2, 2)); b = np.ones((2, 2))
```

```
>>> np.sum(b, axis=1)  
array([ 2.,  2.])
```

```
>>> a.shape  
(2, 2)
```

```
>>> np.reshape(a, (1, 4))  
array([[ 0.,  0.,  0.,  0.]])
```

TensorFlow 예제

```
>>> import tensorflow as tf
>>> sess = tf.Session()

>>> a = tf.zeros((2, 2)); b = tf.ones((2, 2))

>>> sess.run(tf.reduce_sum(b, axis=1))
[ 2.,  2.]

>>> a.get_shape()
(2, 2)

>>> sess.run(tf.reshape(a, (1, 4)))
[[ 0.,  0.,  0.,  0.]]
```

sess.run()

```
>>> sess = tf.Session()
>>> a = np.zeros((2, 2)); ta = tf.zeros((2, 2))

>>> print(a)
[[ 0.  0.]
 [ 0.  0.]]

>>> print(ta)
Tensor("zeros:0", shape=(2, 2), dtype=float32)

>>> print(sess.run(ta))
[[ 0.  0.]
 [ 0.  0.]]
```


sess.run()

```
>>> sess = tf.Session()
>>> a = np.zeros((2, 2)); ta = tf.zeros((2, 2))
```

```
>>> print(a)
[[ 0.  0.]
 [ 0.  0.]
```

TensorFlow는 먼저 **계산 그래프**를 정의하는 것에서
부터 출발함. 이 때 계산 그래프는 실행 (evaluate)
되기 전까지 값을 가지고 있지 않음.

```
>>> print(ta)
Tensor("zeros:0", shape=(2, 2), dtype=float32)
```

```
>>> print(sess.run(ta))
[[ 0.  0.]
 [ 0.  0.]
```

계산 그래프

(Computation graph)

- TensorFlow 프로그램은 **계산 그래프**를 만들고, 세션을 통해 계산 그래프의 명령어를 **실행**하는 과정을 거침
 - 정적그래프 방식 (이해하기 어렵고 사용하기 불편하지만 구현이 효율적)

```
>>> a = tf.constant(5.0)
>>> b = tf.constant(6.0)
>>> c = a * b
```

```
>>> sess = tf.Session()
>>> print(sess.run(c))
30.0
```

계산 그래프

(Computation graph)

- TensorFlow 프로그램은 **계산 그래프**를 만들고, 세션을 통해 계산 그래프의 명령어를 **실행**하는 과정을 거침
 - 정적그래프 방식 (이해하기 어렵고 사용하기 불편하지만 구현이 효율적)

```
>>> a = tf.constant(5.0)
>>> b = tf.constant(6.0)
>>> c = a * b
```

← **계산 그래프**를 정의하는 단계로,
내부 값들을 볼 수 없음

```
>>> sess = tf.Session()
>>> print(sess.run(c))
30.0
```

계산 그래프

(Computation graph)

- TensorFlow 프로그램은 **계산 그래프**를 만들고, 세션을 통해 계산 그래프의 명령어를 **실행**하는 과정을 거침
 - 정적그래프 방식 (이해하기 어렵고 사용하기 불편하지만 구현이 효율적)

```
>>> a = tf.constant(5.0)
>>> b = tf.constant(6.0)
>>> c = a * b
```

← **계산 그래프**를 정의하는 단계로,
내부 값들을 볼 수 없음

```
>>> sess = tf.Session()
>>> print(sess.run(c))
30.0
```

← 사전 정의된 계산 그래프를 실행하는 단계로,
그래프를 실행하기 전 **tf.Session**을 통해
세션을 만들어야 함

TensorFlow 세션

- 세션은 명령어와 텐서 오브젝트들이 실행될 환경에 대한 정보를 내부에 갖고 있음
- 아래는 세션을 만드는 예시로, 두 코드 모두 같은 역할을 함

```
>>> sess = tf.Session()  
>>> print(sess.run(c))
```

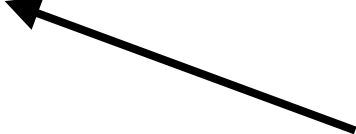
```
>>> with tf.Session() as sess:  
>>>     print(sess.run(c))  
>>>     print(c.eval())
```

TensorFlow 세션

- 세션은 명령어와 텐서 오브젝트들이 실행될 환경에 대한 정보를 내부에 갖고 있음
- 아래는 세션을 만드는 예시로, 두 코드 모두 같은 역할을 함

```
>>> sess = tf.Session()  
>>> print(sess.run(c))
```

```
>>> with tf.Session() as sess:  
>>>     print(sess.run(c))  
>>>     print(c.eval())
```



`sess.run(c)`를 쓰기 쉽게 만들어 둔 편리 함수로,
`tf.Session` 범위와 같이 쓰이곤 함

TensorFlow 변수

- 지금까지 텐서는 모두 상수 타입의 텐서를 사용
- 하지만 모델을 학습할 때는 값을 변경해야 하기 때문에 변수 타입의 텐서가 필요

```
>>> w = tf.Variable(tf.zeros((2, 2)), name="weight")
>>> with tf.Session() as sess:
>>>     print(sess.run(w))
```

TensorFlow 변수

- 지금까지 텐서는 모두 **상수** 타입의 텐서를 사용
- 하지만 모델을 학습할 때는 값을 변경해야 하기 때문에 **변수** 타입의 텐서가 필요

```
>>> w = tf.Variable(tf.zeros((2, 2)), name="weight")
>>> with tf.Session() as sess:
>>>     print(sess.run(w))
```

Traceback (most recent call last):

.....

tensorflow.python.framework.errors_impl.FailedPreconditionError:
Attempting to use uninitialized value weight

변수는 초기화가 필요!

```
>>> w = tf.Variable(tf.random_normal([5, 2], stddev=0.1),  
                    name="weight")
```

```
>>> with tf.Session() as sess:  
>>>     sess.run(tf.global_variables_initializer())  
>>>     print(sess.run(w))
```

```
[[-0.10020355 -0.01114563]  
 [ 0.04050281 -0.15980773]  
 [-0.00628474 -0.02608337]  
 [ 0.16397022  0.02898547]  
 [ 0.04264377  0.04281621]]
```

변수는 초기화가 필요!

tf.Variable는 임의의 값이나 상수로부터 생성 가능



```
>>> w = tf.Variable(tf.random_normal([5, 2], stddev=0.1),  
                    name="weight")
```

```
>>> with tf.Session() as sess:  
>>>     sess.run(tf.global_variables_initializer())  
>>>     print(sess.run(w))
```

```
[[-0.10020355 -0.01114563]  
 [ 0.04050281 -0.15980773]  
 [-0.00628474 -0.02608337]  
 [ 0.16397022  0.02898547]  
 [ 0.04264377  0.04281621]]
```

변수는 초기화가 필요!

tf.Variable는 임의의 값이나 상수로부터 생성 가능

```
>>> w = tf.Variable(tf.random_normal([5, 2], stddev=0.1),  
                    name="weight")
```

```
>>> with tf.Session() as sess:  
>>>     sess.run(tf.global_variables_initializer())  
>>>     print(sess.run(w))
```

```
[[-0.10020355 -0.01114563]  
 [ 0.04050281 -0.15980773]  
 [-0.00628474 -0.02608337]  
 [ 0.16397022  0.02898547]  
 [ 0.04264377  0.04281621]]
```

tf.Variable는 초기화 단계가 필요!!!

변수 업데이트 예제

```
>>> state = tf.Variable(0, name="counter")
>>> new_value = tf.add(state, tf.constant(1))
>>> update = tf.assign(state, new_value)

>>> with tf.Session() as sess:
>>>     sess.run(tf.global_variables_initializer())
>>>     print(sess.run(state))
>>>     for _ in range(3):
>>>         sess.run(update)
>>>         print(sess.run(state))
0
1
2
3
```

변수 가져오기

```
>>> x1 = tf.constant(1)
>>> x2 = tf.constant(2)
>>> x3 = tf.constant(3)
>>> temp = tf.add(x2, x3)
>>> mul = tf.mul(x1, temp)


>>> with tf.Session() as sess:
>>>     result1, result2 = sess.run([mul, temp])
>>>     print(result1, result2)
5 5
```

변수 가져오기

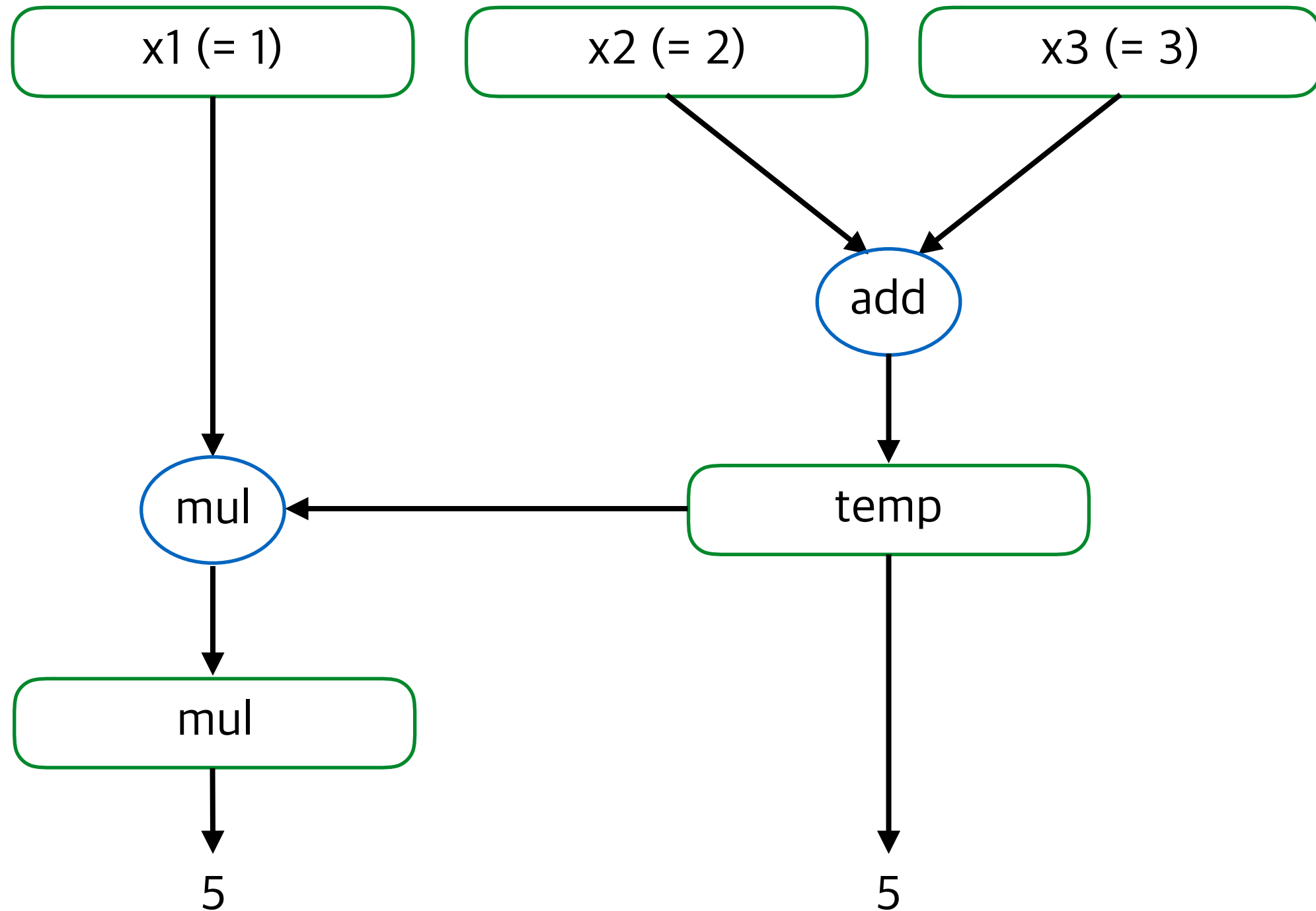
```
>>> x1 = tf.constant(1)
>>> x2 = tf.constant(2)
>>> x3 = tf.constant(3)
>>> temp = tf.add(x2, x3)
>>> mul = tf.mul(x1, temp)
```

sess.run(var)를 호출해서 var 에 있는 값을 가져올 수 있고, sess.run([var1, ..,])를 통해 여러 변수의 값을 한꺼번에 가져올 수 있음

```
>>> with tf.Session() as sess:
>>>     result1, result2 = sess.run([mul, temp])
>>>     print(result1, result2)
5 5
```



변수 가져오기



TensorFlow Placeholder

- 지금까지는 텐서 내부 값을 초기에 하나하나 지정해주었음
 - 만약 외부 데이터 (입력 데이터 등)를 텐서에 주입하려면??
 - 가장 쉬운 방법은 `tf.placeholder` 와 `feed_dict`를 사용하는 것!

```
>>> a = tf.placeholder(tf.int16)
>>> b = tf.placeholder(tf.int16)
```

```
>>> add = tf.add(a, b)
>>> mul = tf.mul(a, b)
```

```
>>> with tf.Session() as sess:
>>>     print(sess.run(add, feed_dict={a: 2, b: 3}))
>>>     print(sess.run(mul, feed_dict={a: 2, b: 3}))
```

5
6

TensorFlow Placeholder

- 지금까지는 텐서 내부 값을 초기에 하나하나 지정해주었음
 - 만약 외부 데이터 (입력 데이터 등)를 텐서에 주입하려면??
 - 가장 쉬운 방법은 `tf.placeholder` 와 `feed_dict`를 사용하는 것!

```
>>> a = tf.placeholder(tf.int16)
>>> b = tf.placeholder(tf.int16)
```

 데이터를 저장하기 위한 **placeholder** 선언

```
>>> add = tf.add(a, b)
>>> mul = tf.mul(a, b)
```

```
>>> with tf.Session() as sess:
>>>     print(sess.run(add, feed_dict={a: 2, b: 3}))
>>>     print(sess.run(mul, feed_dict={a: 2, b: 3}))
```

5
6

TensorFlow Placeholder

- 지금까지는 텐서 내부 값을 초기에 하나하나 지정해주었음
 - 만약 외부 데이터 (입력 데이터 등)를 텐서에 주입하려면??
 - 가장 쉬운 방법은 `tf.placeholder` 와 `feed_dict`를 사용하는 것!

```
>>> a = tf.placeholder(tf.int16)
>>> b = tf.placeholder(tf.int16)
```

← 데이터를 저장하기 위한 **placeholder** 선언

```
>>> add = tf.add(a, b)
>>> mul = tf.mul(a, b)
```

```
>>> with tf.Session() as sess:
>>>     print(sess.run(add, feed_dict={a: 2, b: 3}))
>>>     print(sess.run(mul, feed_dict={a: 2, b: 3}))
```

5
6

← A **feed_dict** 는 `tf.placeholder` 에 데이터를 넣어주기 위한 파이썬 딕셔너리

TensorFlow Placeholder 예제

tf.constant를 사용하는 예제

```
matrix1 = tf.constant([[3., 3.]])  
matrix2 = tf.constant([[2.], [2.]])  
product = tf.matmul(matrix1, matrix2)
```

```
with tf.Session() as sess:  
    result = sess.run(product)  
    print(result)
```

placeholder를 사용하는 예제

```
import numpy as np
```

```
matrix1 = tf.placeholder(tf.float32, [1, 2])  
matrix2 = tf.placeholder(tf.float32, [2, 1])  
product = tf.matmul(matrix1, matrix2)
```

```
with tf.Session() as sess:  
    mv1 = np.array([[3., 3.]])  
    mv2 = np.array([[2.], [2.]])  
    result = sess.run(product, feed_dict={matrix1: mv1, matrix2: mv2})  
    print(result)
```

Placeholder와 변수의 차이

- placeholder는 내용물을 담는 **그릇**
 - placeholder 정의는 말 그대로 그릇을 만드는 것
 - feed_dict를 통해 그릇에 내용물을 채움
- 네트워크를 학습할 때 placeholder가 필요함
 - TensorFlow는 **정적 그래프**이기 때문에 미리 입력값을 받을 그릇을 생성
 - for 문을 돌면서 매 스텝마다 그릇에 실제 입력값을 넣어줌 (feed_dict)

주의사항

- TensorFlow는 기본적으로 GPU 메모리를 모두 할당
 - 세션을 만들 때 사용 가능한 모든 GPU 메모리를 사용
 - 서버에서 돌릴 경우 한 사람이 모든 자원을 독점할 가능성이 있음
- 세션을 생성할 때 필요한 만큼 메모리를 점유하는 옵션 제공

GPU 필요량에 따라 메모리를 동적으로 할당하는 옵션

```
sess_config = tf.ConfigProto(  
    gpu_options=tf.GPUOptions(allow_growth=True))
```

```
with tf.Session(config=sess_config) as sess:  
    sess.run(..)
```

퍼셉트론으로 AND, OR 풀기

```
and_x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
and_y = np.array([0, 0, 0, 1])
or_x  = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
or_y  = np.array([0, 1, 1, 1])
```

```
learning_rate = 0.1
```

```
inputs = tf.placeholder(tf.float32, [None, 2])
label   = tf.placeholder(tf.int64, [None])
weight  = tf.Variable(tf.random_normal([2, 2]))
bias     = tf.Variable(tf.zeros([2]))
```

```
logit = tf.matmul(inputs, weight) + bias
pred_op = tf.nn.softmax(logit)
```

```
loss_op = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels=label, logits=logit))
opt = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss_op)
```

```
with tf.Session(config=sess_config) as sess:
    sess.run(tf.global_variables_initializer())
```

```
    for step in range(500):
```

```
        _, loss = sess.run([opt, loss_op], feed_dict={inputs:and_x, label:and_y})
```

```
        if (step+1)%100 == 0:
```

```
            pred = sess.run(pred_op, feed_dict={inputs:and_x, label:and_y})
```

```
            print("step", step+1, "loss:", loss, "gt:", and_y, "predict:", np.argmax(pred, axis=1))
```

```
step 100 loss: 0.381541 gt: [0 0 0 1] predict: [0 0 0 1]
```

```
step 200 loss: 0.266401 gt: [0 0 0 1] predict: [0 0 0 1]
```

```
step 300 loss: 0.207806 gt: [0 0 0 1] predict: [0 0 0 1]
```

```
step 400 loss: 0.170944 gt: [0 0 0 1] predict: [0 0 0 1]
```

```
step 500 loss: 0.145226 gt: [0 0 0 1] predict: [0 0 0 1]
```

퍼셉트론 구성

```
# 입력값을 받기 위한 placeholder
```

```
inputs = tf.placeholder(tf.float32, [None, 2])  
label  = tf.placeholder(tf.int64, [None])
```

```
# weight / bias 변수
```

```
weight = tf.Variable(tf.random_normal([2, 2]))  
bias    = tf.Variable(tf.zeros([2]))
```

```
# 네트워크 그래프 구성 ( $Wx + b$ )
```


```
logit = tf.matmul(inputs, weight) + bias  
pred_op = tf.nn.softmax(logit)
```

Loss

```
loss_op = tf.reduce_mean(  
    tf.nn.sparse_softmax_cross_entropy_with_logits(  
        labels=label, logits=logit))
```


Loss


loss를 계산 후 평균 (배치 단위로 실행하기 때문)
reduce_sum의 경우 배치가 큰 경우 NaN 에러 발생 가능



```
loss_op = tf.reduce_mean(  
    tf.nn.sparse_softmax_cross_entropy_with_logits(  
        labels=label, logits=logit))
```

Loss

loss를 계산 후 평균 (배치 단위로 실행하기 때문)
reduce_sum의 경우 배치가 큰 경우 NaN 에러 발생 가능



```
loss_op = tf.reduce_mean(  
    tf.nn.sparse_softmax_cross_entropy_with_logits(  
        labels=label, logits=logit))
```

softmax_cross_entropy_with_logits:

- **Softmax를 취하기 전** 값 (logits)을 사용하여 loss 계산
- 이 때 logits, labels은 [batch, class_num] 이며 tf.float32 타입
- 예: 클래스가 3개인 경우 logits=[0.3, 0.9, 0.1] labels=[0.0, **1.0**, 0.0]

sparse_softmax_cross_entropy_with_logits:

- 위 함수의 단점: 클래스가 많은 경우 label을 **one-hot으로 저장하는 것은 비효율적**
- 이 함수는 logits은 동일하지만, labels은 [batch] 크기며, tf.int32 혹은 int64 타입
- 예: 클래스가 3개인 경우 logits=[0.3, 0.9, 0.1] labels=[**2**]

Optimizer

```
opt = tf.train.GradientDescentOptimizer(  
    learning_rate).minimize(loss_op)
```

Optimizer

```
opt = tf.train.GradientDescentOptimizer(  
    learning_rate).minimize(loss_op)
```

- `tf.train.####Optimizer` 를 통해 Optimizer 생성
- `tf.train.####Optimizer.minimize(loss_op)`
로 `loss_op`를 최소화 하는 back propagation 그래프 생성

퍼셉트론으로 AND, OR 풀기

```
and_x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
and_y = np.array([0, 0, 0, 1])
or_x  = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
or_y  = np.array([0, 1, 1, 1])

learning_rate = 0.1

inputs = tf.placeholder(tf.float32, [None, 2])
label  = tf.placeholder(tf.int64, [None])
weight = tf.Variable(tf.random_normal([2, 2]))
bias    = tf.Variable(tf.zeros([2]))

logit = tf.matmul(inputs, weight) + bias
pred_op = tf.nn.softmax(logit)

loss_op = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels=label, logits=logit))
opt = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss_op)
```

```
with tf.Session(config=sess_config) as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(500):
        _, loss = sess.run([opt, loss_op], feed_dict={inputs:and_x, label:and_y})
        if (step+1)%100 == 0:
            pred = sess.run(pred_op, feed_dict={inputs:and_x, label:and_y})
            print("step", step+1, "loss:", loss, "gt:", and_y, "predict:", np.argmax(pred, axis=1))
```

```
step 100 loss: 0.381541 gt: [0 0 0 1] predict: [0 0 0 1]
step 200 loss: 0.266401 gt: [0 0 0 1] predict: [0 0 0 1]
step 300 loss: 0.207806 gt: [0 0 0 1] predict: [0 0 0 1]
step 400 loss: 0.170944 gt: [0 0 0 1] predict: [0 0 0 1]
step 500 loss: 0.145226 gt: [0 0 0 1] predict: [0 0 0 1]
```

변수 스코프 (scope)

- 네트워크가 복잡해지면 변수(파라미터)가 매우 많아짐
 - 변수를 이름과 스코프로 관리하자 -> `tf.variable_scope()`
 - `tf.variable_scope()` 를 사용하면 변수간 이름 충돌을 방지 가능
 - `tf.get_variable("name")`: 현재 스코프 내에서 "name" 과 이름이 일치하는 변수를 가져오거나 없으면 생성하는 함수

tf.variable_scope()

- 변수 스코프는 변수 이름 앞에 접두사로 추가되는 변수 name-spacing 타입

tf.variable_scope()

- 변수 스코프는 변수 이름 앞에 접두사로 추가되는 변수 name-spacing 타입

```
var1 = tf.Variable([1], name="var")
with tf.variable_scope("foo"):
    with tf.variable_scope("bar"):
        var2 = tf.Variable([1], name="var")
        var3 = tf.Variable([1], name="var")
```

```
print("var1: {}".format(var1.name))
print("var2: {}".format(var2.name))
print("var3: {}".format(var3.name))
```

```
var1: var:0
var2: foo/bar/var:0
var3: foo/bar/var_1:0
```


tf.get_variable()

- tf.Variable 의 문제점:
 - 변수의 **재사용**이 불가능함 (RNN이나 Recursive NN을 구현할 때 필수)
 - 또한 파라미터 초기화를 지정해 줄 방법이 없음 (he, xavier 초기화)

•

tf.get_variable()

- tf.Variable 의 문제점:
 - 변수의 **재사용**이 불가능함 (RNN이나 Recursive NN을 구현할 때 필수)
 - 또한 파라미터 초기화를 지정해 줄 방법이 없음 (he, xavier 초기화)

```
var1 = tf.Variable([1], name="var")
with tf.variable_scope("foo"):
    with tf.variable_scope("bar") as scp:
        var2 = tf.Variable([1], name="var")
        scp.reuse_variables() # allow reuse variables
        var3 = tf.Variable([1], name="var")
```

```
print("var1: {}".format(var1.name))
print("var2: {}".format(var2.name))
print("var3: {}".format(var3.name))
```

```
var1: var:0
var2: foo/bar/var:0
var3: foo/bar/var_1:0
```

tf.Variable

- tf.Variable 의 문제점:
 - 변수의 **재사용**이 불가능함 (RNN이나 Recursive NN을 구현할 때 필수)
 - 또한 파라미터 초기화를 지정해 줄 방법이 없음 (he, xavier 초기화)

tf.Variable

- tf.Variable 의 문제점:
 - 변수의 **재사용**이 불가능함 (RNN이나 Recursive NN을 구현할 때 필수)
 - 또한 파라미터 초기화를 지정해 줄 방법이 없음 (he, xavier 초기화)

```
var1 = tf.Variable([1], name="var")
with tf.variable_scope("foo"):
    with tf.variable_scope("bar") as scp:
        var2 = tf.Variable([1], name="var")
        scp.reuse_variables() # allow reuse variables
        var3 = tf.Variable([1], name="var")
```

```
print("var1: {}".format(var1.name))
print("var2: {}".format(var2.name))
print("var3: {}".format(var3.name))
```

```
var1: var:0
var2: foo/bar/var:0
var3: foo/bar/var_1:0
```

tf.Variable

- `tf.Variable` 의 문제점:
 - 변수의 **재사용**이 불가능함 (RNN이나 Recursive NN을 구현할 때 필수)
 - 또한 파라미터 초기화를 지정해 줄 방법이 없음 (he, xavier 초기화)

```
var1 = tf.Variable([1], name="var")
with tf.variable_scope("foo"):
    with tf.variable_scope("bar") as scp:
        var2 = tf.Variable([1], name="var")
        scp.reuse_variables() # allow reuse variables
        var3 = tf.Variable([1], name="var")
```

```
print("var1: {}".format(var1.name))
print("var2: {}".format(var2.name))
print("var3: {}".format(var3.name))
```

```
var1: var:0
```

```
var2: foo/bar/var:0
var3: foo/bar/var_1:0
```

← var3 가 foo/bar/var:0 를 참조하기를 원하지만
`tf.Variable` 는 불가능함

tf.get_variable()

- tf.get_variable() 은 잘 동작

```
var1 = tf.get_variable("var", [1])
with tf.variable_scope("foo"):
    with tf.variable_scope("bar") as scp:
        var2 = tf.get_variable("var", [1])
        scp.reuse_variables() # allow reuse variables
        var3 = tf.get_variable("var", [1])
```

```
print("var1: {}".format(var1.name))
print("var2: {}".format(var2.name))
print("var3: {}".format(var3.name))
```

```
var1: var:0
var2: foo/bar/var:0
var3: foo/bar/var:0
```

tf.get_variable()

- tf.get_variable() 은 잘 동작

```
var1 = tf.get_variable("var", [1])
with tf.variable_scope("foo"):
    with tf.variable_scope("bar") as scp:
        var2 = tf.get_variable("var", [1])
        scp.reuse_variables() # allow reuse variables
        var3 = tf.get_variable("var", [1])
```

```
print("var1: {}".format(var1.name))
print("var2: {}".format(var2.name))
print("var3: {}".format(var3.name))
```

```
var1: var:0
var2: foo/bar/var:0
var3: foo/bar/var:0
```

tf.get_variable()

- Reuse의 설정 여부에 따라 다르게 동작함
- Case 1: reuse == False
 - 전달 받은 이름을 가지는 새로운 변수를 생성하고 반환
 - 전달 받은 이름을 가진 변수가 존재한다면 ValueError 에러
- Case 2: reuse == True
 - 전달 받은 이름을 가지는 변수 탐색
 - 없으면 ValueError 에러

변수 공유

```
with tf.variable_scope("foo"):
    with tf.variable_scope("bar") as scp:
        var1 = tf.get_variable("var", [1])
        scp.reuse_variables()
        var2 = tf.get_variable("var", [1])

    with tf.variable_scope("bar", reuse=True):
        var3 = tf.get_variable("var", [1])

print("var1: {}".format(var1.name))
print("var2: {}".format(var2.name))
print("var3: {}".format(var3.name))

var1: foo/bar/var:0
var2: foo/bar/var:0
var3: foo/bar/var:0
```

변수 공유

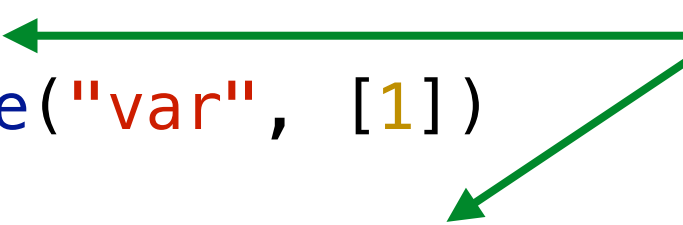
```
with tf.variable_scope("foo"):
    with tf.variable_scope("bar") as scp:
        var1 = tf.get_variable("var", [1])
        scp.reuse_variables()
        var2 = tf.get_variable("var", [1])

    with tf.variable_scope("bar", reuse=True):
        var3 = tf.get_variable("var", [1])

print("var1: {}".format(var1.name))
print("var2: {}".format(var2.name))
print("var3: {}".format(var3.name))
```

reuse를 True로
설정하는 방법

var1: foo/bar/var:0
var2: foo/bar/var:0
var3: foo/bar/var:0



요약

- **항상!!** `tf.Variable` 대신 `tf.get_variable` 를 사용하세요
 - `tf.Variable` 는 변수(파라미터) 공유를 지원하지 않음
 - `tf.Variable` 는 초기화나 regularizer를 지원하지 않음
- **스코핑을 열심히 합시다**
 - 모델을 시각화하거나 출력할 때 정말 중요
 - TensorFlow 내부에 접두사/접미사로 변수를 가져오는 함수가 존재
`tf.names.get_variables_by_name("my_var", "my_scope")`
 - 디버깅하거나 사전학습 된 모델을 가져올 때 편합니다

요약

- **항상!!** `tf.Variable` 대신 `tf.get_variable`
 - `tf.Variable` 는 변수(파라미터) 공유
 - `tf.Variable` 는 초기화나 regularizer

- **스코핑을 열심히 합시다**

- 모델을 시각화하거나 출력할 때 정말 중요
- TensorFlow 내부에 접두사/접미사로 변수를 가져오는 함수가 존재
`tf.names.get_variables_by_name("my_var", "my_scope")`
- 디버깅하거나 사전학습 된 모델을 가져올 때 편합니다

```
vgg_16/conv1/conv1_1 (?, 224, 224, 64)
vgg_16/conv1/conv1_2 (?, 224, 224, 64)
vgg_16/pool1 (?, 112, 112, 64)
vgg_16/conv2/conv2_1 (?, 112, 112, 128)
vgg_16/conv2/conv2_2 (?, 112, 112, 128)
vgg_16/pool2 (?, 56, 56, 128)
vgg_16/conv3/conv3_1 (?, 56, 56, 256)
vgg_16/conv3/conv3_2 (?, 56, 56, 256)
vgg_16/conv3/conv3_3 (?, 56, 56, 256)
vgg_16/pool3 (?, 28, 28, 256)
vgg_16/conv4/conv4_1 (?, 28, 28, 512)
vgg_16/conv4/conv4_2 (?, 28, 28, 512)
vgg_16/conv4/conv4_3 (?, 28, 28, 512)
vgg_16/pool4 (?, 14, 14, 512)
vgg_16/conv5/conv5_1 (?, 14, 14, 512)
vgg_16/conv5/conv5_2 (?, 14, 14, 512)
vgg_16/conv5/conv5_3 (?, 14, 14, 512)
vgg_16/pool5 (?, 7, 7, 512)
vgg_16/fc6 (?, 1, 1, 4096)
vgg_16/fc7 (?, 1, 1, 4096)
vgg_16/fc8 (?, 1000)
```

주의할 점

- `tf.get_variable` 은 이름을 가진 변수가 있는지 먼저 탐색
 - Jupyter 노트북과 사용할 경우 약간 불편함
 - 네트워크를 생성하는 셀을 두 번 이상 실행 하면 에러 발생

ValueError: Variable vgg_19/conv1/conv1_1/weights already exists, disallowed. Did you mean to set reuse=True in VarScope? Originally defined at:

```
File "/home/nmhkahn/.venv3/lib/python3.5/site-packages/tensorflow/contrib/framework/python/ops/variables.py", line 217, in variable
    use_resource=use_resource)
File "/home/nmhkahn/.venv3/lib/python3.5/site-packages/tensorflow/contrib/framework/python/ops/arg_scope.py", line 181, in func_with_args
    return func(*args, **current_args)
File "/home/nmhkahn/.venv3/lib/python3.5/site-packages/tensorflow/contrib/framework/python/ops/variables.py", line 262, in model_variable
    use_resource=use_resource)
```

- 해결책: Jupyter 노트북을 다시 꺼다가 켜세요..

TensorFlow API

- 활성 함수

- `tf.sigmoid(x)`
- `tf.tanh(x)`
- `tf.nn.relu(x)`

- weight 초기화

```
normal = tf.truncated_normal_initializer()  
xavier = tf.contrib.layers.xavier_initializer()  
he      = tf.contrib.layers.variance_scaling_initializer()
```

```
v1 = tf.get_variable("V1", [3, 3], initializer=normal)  
v2 = tf.get_variable("V2", [3, 3], initializer=xavier)  
v3 = tf.get_variable("V3", [3, 3], initializer=he)
```

Dropout API

- 학습/테스트 수행 방법이 다르므로 이를 명시해야 함

- 방법 1:

```
keep_prob = tf.placeholder(tf.float32, None)
out = tf.nn.dropout(X, keep_prob=keep_prob)
sess.run(out, feed_dict={keep_prob: 0.5}) # 학습 시
sess.run(out, feed_dict={keep_prob: 1.0}) # 테스트 시
```

- 방법 2:

```
is_training = tf.placeholder(tf.bool)
out = tf.layers.dropout(X, rate=0.5, training=is_training)
sess.run(out, feed_dict={is_training: True}) # 학습 시
sess.run(out, feed_dict={is_training: False}) # 테스트 시
```

Dropout API

- 학습/테스트 수행 방법이 다르므로 이를 명시해야 함

- 방법 1:

keep_prob: 뉴런을 살릴 확률

```
keep_prob = tf.placeholder(tf.float32, None)
out = tf.nn.dropout(X, keep_prob=keep_prob)
sess.run(out, feed_dict={keep_prob: 0.5}) # 학습 시
sess.run(out, feed_dict={keep_prob: 1.0}) # 테스트 시
```

- 방법 2:


```
is_training = tf.placeholder(tf.bool)
out = tf.layers.dropout(X, rate=0.5, training=is_training)
sess.run(out, feed_dict={is_training: True}) # 학습 시
sess.run(out, feed_dict={is_training: False}) # 테스트 시
```


Dropout API

- 학습/테스트 수행 방법이 다르므로 이를 명시해야 함


- 방법 1:

`keep_prob = tf.placeholder(tf.float32, None)`
`out = tf.nn.dropout(X, keep_prob=keep_prob)`
`sess.run(out, feed_dict={keep_prob: 0.5})` # 학습 시
`sess.run(out, feed_dict={keep_prob: 1.0})` # 테스트 시



- 방법 2:

`is_training = tf.placeholder(tf.bool)`
`out = tf.layers.dropout(X, rate=0.5, training=is_training)`
`sess.run(out, feed_dict={is_training: True})` # 학습 시
`sess.run(out, feed_dict={is_training: False})` # 테스트 시



배치 정규화 API

- Dropout과 마찬가지로 학습/테스트 명시

```
is_training = tf.placeholder(tf.bool)
bn = tf.contrib.layers.batch_norm(X,
    is_training=is_training,
    decay=0.9,
    updates_collections=None)

sess.run(bn, feed_dict={is_training:True})
sess.run(bn, feed_dict={is_training:False})
```

배치 정규화 API

- Dropout과 마찬가지로 학습/테스트 명시

```
is_training = tf.placeholder(tf.bool)
bn = tf.contrib.layers.batch_norm(X,
    is_training=is_training,
    decay=0.9,
    updates_collections=None)
```

이동평균의 업데이트를 알아서 하도록
(더 빠르지만 명시적으로
업데이트를 해야하는 방법도 존재)

```
sess.run(bn, feed_dict={is_training:True})
sess.run(bn, feed_dict={is_training:False})
```

배치 정규화 API

- Dropout과 마찬가지로 학습/테스트 명시

이동평균을 구할 때 사용하는 하이퍼파라미터
기본값은 0.999인데, 0.9가 더 좋은 성능을 보인다고 합니다

```
is_training = tf.placeholder(tf.bool)
bn = tf.contrib.layers.batch_norm(X,
    is_training=is_training,
    decay=0.9,
    updates_collections=None)
```

이동평균의 업데이트를 알아서 하도록
(더 빠르지만 명시적으로
업데이트를 해야하는 방법도 존재)

```
sess.run(bn, feed_dict={is_training:True})
sess.run(bn, feed_dict={is_training:False})
```

컨볼루션 / 풀링 API

- `tf.nn.conv2d(input, filter, strides, padding, name=None)`
 1. `input`: [batch, height, width, channels] 크기의 **4D 텐서**
 2. `filter`: [filter_height, filter_width, in_channels, out_channels] 크기의 **4D 텐서**
 3. `stride`: 컨볼루션 stride 크기. [1, stride, stride, 1] 와 같이 사용
(배치와 채널별로 stride를 적용하지 않기 때문)
 4. `padding`: 패딩시 사용할 알고리즘 (“SAME” = 크기 똑같이 “VALID” = 패딩 x)
- `tf.nn.nn_pool(value, ksize, strides, padding, name=None)`
 1. `value`:
 2. `ksize`: 컨볼루션의 filter와 비슷 [1, ksize, ksize, 1] 와 같이 사용
 3. `stride`:
 4. `padding`:

Wrapper

- 매 컨볼루션/풀링/FC 마다 변수를 만들고 각 함수를 사용하는 것은 매우 번거로운 작업
- 복붙 코드 (**boilerplate**) 양산
- 컨볼루션/풀링/FC 레이어를 함수로 만들자!
- (취향 차이인데 저는 주로 이렇게 작업합니다)

Conv

```
from tensorflow.contrib.layers import variance_scaling_initializer
he_init = variance_scaling_initializer()

def conv(bottom,
          num_filter, ksize=3, stride=1, padding="SAME",
          scope=None):
    bottom_shape = bottom.get_shape().as_list()[3]

    with tf.variable_scope(scope or "conv"):
        W = tf.get_variable("W",
                            [ksize, ksize, bottom_shape, num_filter],
                            initializer=he_init)
        b = tf.get_variable("b", [num_filter],
                            initializer=tf.constant_initializer(0))

        x = tf.nn.conv2d(bottom, W,
                          strides=[1, stride, stride, 1],
                          padding=padding)
        x = tf.nn.relu(tf.nn.bias_add(x, b))

    return x
```

Conv

```
from tensorflow.contrib.layers import variance_scaling_initializer  
he_init = variance_scaling_initializer()
```

```
def conv(bottom,  
          num_filter, ksize=3, stride=1, padding="SAME",  
          scope=None):
```

```
    bottom_shape = bottom.get_shape().as_list()[3]
```

```
    with tf.variable_scope(scope or "conv"):
```

```
        W = tf.get_variable("W",  
                             [ksize, ksize, bottom_shape, num_filter],  
                             initializer=he_init)
```

```
        b = tf.get_variable("b", [num_filter],  
                             initializer=tf.constant_initializer(0))
```

```
        x = tf.nn.conv2d(bottom, W,  
                          strides=[1, stride, stride, 1],  
                          padding=padding)
```

```
        x = tf.nn.relu(tf.nn.bias_add(x, b))
```

```
    return x
```

이전 레이어의 shape을 계산

max pool

```
def maxpool(bottom,  
            ksize=2, stride=2, padding="SAME",  
            scope=None):  
  
    with tf.variable_scope(scope or "maxpool"):  
        pool = tf.nn.max_pool(bottom, ksize=[1, ksize, ksize, 1],  
                               strides=[1, stride, stride, 1],  
                               padding=padding)  
  
    return pool
```

FC

```
def fc(bottom, num_dims, scope=None):

    bottom_shape = bottom.get_shape().as_list()
    if len(bottom_shape) > 2:
        bottom = tf.reshape(bottom,
                              [-1, reduce(lambda x, y: x*y, bottom_shape[1:])])
        bottom_shape = bottom.get_shape().as_list()

    with tf.variable_scope(scope or "fc"):
        W = tf.get_variable("W", [bottom_shape[1], num_dims],
                            initializer=he_init)
        b = tf.get_variable("b", [num_dims],
                            initializer=tf.constant_initializer(0))

        out = tf.nn.bias_add(tf.matmul(bottom, W), b)
    return out


def fc_relu(bottom, num_dims, scope=None):

    with tf.variable_scope(scope or "fc"):
        out = fc(bottom, num_dims, scope="fc")
        relu = tf.nn.relu(out)

    return relu
```

FC

```
def fc(bottom, num_dims, scope=None):
```

```
    bottom_shape = bottom.get_shape().as_list()
```

```
    if len(bottom_shape) > 2:
```

```
        bottom = tf.reshape(bottom,
```

```
                             [-1, reduce(lambda x, y: x*y, bottom_shape[1:])])
```

```
        bottom_shape = bottom.get_shape().as_list()
```

```
    with tf.variable_scope(scope or "fc"):
```

```
        W = tf.get_variable("W", [bottom_shape[1], num_dims],
```

```
                             initializer=he_init)
```

```
        b = tf.get_variable("b", [num_dims],
```

```
                             initializer=tf.constant_initializer(0))
```

```
        out = tf.nn.bias_add(tf.matmul(bottom, W), b)
```

```
    return out
```

```
def fc_relu(bottom, num_dims, scope=None):
```

```
    with tf.variable_scope(scope or "fc"):
```

```
        out = fc(bottom, num_dims, scope="fc")
```

```
        relu = tf.nn.relu(out)
```

```
    return relu
```

이전 레이어의 shape가 2차원 이상이면 flatten

- 컨볼루션은 (N, H, W, C) shape를
- FC 레이어는 (N, C) shape를 입력으로 받기 때문

네트워크 조합

```
keep_prob = tf.placeholder(tf.float32, None)

def conv_net(x, keep_prob):
    x = tf.reshape(x, shape=[-1, 28, 28, 1])

    conv1 = conv(x, 32, 5, scope="conv_1")
    conv1 = maxpool(conv1, scope="maxpool_1")
    conv2 = conv(conv1, 64, 5, scope="conv_2")
    conv2 = maxpool(conv2, scope="maxpool_2")

    fc1 = fc_relu(conv2, 1024, scope="fc_1")
    fc1 = tf.nn.dropout(fc1, keep_prob)

    out = fc(fc1, 10, scope="out")
    return out
```

네트워크 조합

tf.layers.dropout을 사용해도 됩니다

```
keep_prob = tf.placeholder(tf.float32, None)
```

```
def conv_net(x, keep_prob):  
    x = tf.reshape(x, shape=[-1, 28, 28, 1])  
  
    conv1 = conv(x, 32, 5, scope="conv_1")  
    conv1 = maxpool(conv1, scope="maxpool_1")  
    conv2 = conv(conv1, 64, 5, scope="conv_2")  
    conv2 = maxpool(conv2, scope="maxpool_2")  
  
    fc1 = fc_relu(conv2, 1024, scope="fc_1")  
    fc1 = tf.nn.dropout(fc1, keep_prob)  
  
    out = fc(fc1, 10, scope="out")  
    return out
```

그 외에는

- 공식문서
- https://www.tensorflow.org/api_docs/

tf.contrib

- `tf.contrib` 는 contribute된 코드들이 들어있는 모듈
 - 완벽히 정식 모듈은 아니지만, 거의 준 정식수준의 코드들
 - 사용하기 편리한 인터페이스 혹은 유틸리티들이 많이 있음
 - RNN, seq2seq 혹은 레이어 wrapper 함수도 존재
 - `tf.contrib.slim`: 편리한 함수 및 유틸리티들을 묶은 모듈

tf.contrib.slim

- TensorFlow의 경량화 및 편의성을 위한 라이브러리
 - tf.contrib 에 속한 컴포넌트들을 tf.contrib.slim 라는 네임스페이스로 묶어놓음
 - 다른 하イレ벨 라이브러리와 달리 TensorFlow 네이티브와 쉽게 연동 가능
- 왜 tf.slim이 편리한가요?
 - 네트워크 모델을 쌓고 이를 학습하는 과정이 매우 간단함
 - 모델을 정의할 때 복붙하기 쉬운 코드를 방지 (별도의 wrapper 함수 존재)
 - 사전 학습된 모델에 대한 코드와 학습파일 제공
 - 편리한 유틸리티 제공 (initializer, regularizers, losses ...)

tf.contrib.slim

What are the various components of TF-Slim?

TF-Slim is composed of several parts which were design to exist independently. These include the following main pieces (explained in detail below).

- **arg_scope**: provides a new scope named `arg_scope` that allows a user to define default arguments for specific operations within that scope.
- **data**: contains TF-slim's **dataset** definition, **data providers**, **parallel_reader**, and **decoding** utilities.
- **evaluation**: contains routines for evaluating models.
- **layers**: contains high level layers for building models using tensorflow.
- **learning**: contains routines for training models.
- **losses**: contains commonly used loss functions.
- **metrics**: contains popular evaluation metrics.
- **nets**: contains popular network definitions such as **VGG** and **AlexNet** models.
- **queues**: provides a context manager for easily and safely starting and closing QueueRunners.
- **regularizers**: contains weight regularizers.
- **variables**: provides convenience wrappers for variable creation and manipulation.

Layers

Layer	TF-Slim
BiasAdd	slim.bias_add
BatchNorm	slim.batch_norm
Conv2d	slim.conv2d
Conv2dInPlane	slim.conv2d_in_plane
Conv2dTranspose (Deconv)	slim.conv2d_transpose
FullyConnected	slim.fully_connected
AvgPool2D	slim.avg_pool2d
Dropout	slim.dropout
Flatten	slim.flatten
MaxPool2D	slim.max_pool2d
OneHotEncoding	slim.one_hot_encoding
SeparableConv2	slim.separable_conv2d
UnitNorm	slim.unit_norm

Layers

```
input = ...
with tf.name_scope('conv1_1') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 64, 128],
                                             dtype=tf.float32, stddev=1e-1),
                        name='weights')
    conv = tf.nn.conv2d(input, kernel, [1, 1, 1, 1],
                        padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[128],
                                         dtype=tf.float32),
                        trainable=True, name='biases')
    bias = tf.nn.bias_add(conv, biases)
    conv1 = tf.nn.relu(bias, name=scope)
```

Layers

```
input = ...
with tf.name_scope('conv1_1') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 64, 128],
                                             dtype=tf.float32, stddev=1e-1),
                        name='weights')
    conv = tf.nn.conv2d(input, kernel, [1, 1, 1, 1],
                        padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[128],
                                       dtype=tf.float32),
                        trainable=True, name='biases')
    bias = tf.nn.bias_add(conv, biases)
    conv1 = tf.nn.relu(bias, name=scope)
```

TF-Slim:

```
input = ...
net = slim.conv2d(input, 128, [3, 3],
                  padding='SAME', scope='conv1_1')
```

Layers

1. *slim*을 통해 네트워크를 정의하는 방법

```
net = ...  
net = slim.conv2d(net, 256, [3, 3], scope='conv3_1')  
net = slim.conv2d(net, 256, [3, 3], scope='conv3_2')  
net = slim.conv2d(net, 256, [3, 3], scope='conv3_3')  
net = slim.max_pool2d(net, [2, 2], scope='pool3')
```

2. *repeat* 연산자 사용하기 (1번과 같은 결과):

```
net = ...  
net = slim.repeat(net, 3, slim.conv2d, 256, [3, 3],  
                  scope='conv3')  
net = slim.max_pool(net, [2, 2], scope='pool3')
```

3. *FC* 레이어:

```
x = slim.fully_connected(x, 32, scope='fc/fc_1')  
x = slim.fully_connected(x, 64, scope='fc/fc_2')  
x = slim.fully_connected(x, 128, scope='fc/fc_3')
```

4. *stack* 연산자를 통해 3번과 같은 결과:

```
slim.stack(x, slim.fully_connected, [32, 64, 128], scope='fc')
```

Initializer / Regularizer

- Initializer:
 - `tf.truncated_normal_initializer`
 - `slim.xavier_initializer`
 - `slim.variance_scaling_initializer`
- Regularizer:
 - `slim.l1_regularizer`
 - `slim.l2_regularizer`

```
net = slim.conv2d(inputs, 64, [11, 11], 4, padding='SAME',  
weights_initializer=slim.xavier_initializer(),  
weights_regularizer=slim.l2_regularizer(0.0005),  
scope='conv1')
```

argscope

- 계속 반복되는 복붙코드 (boilerplate) 방지

[illegible]

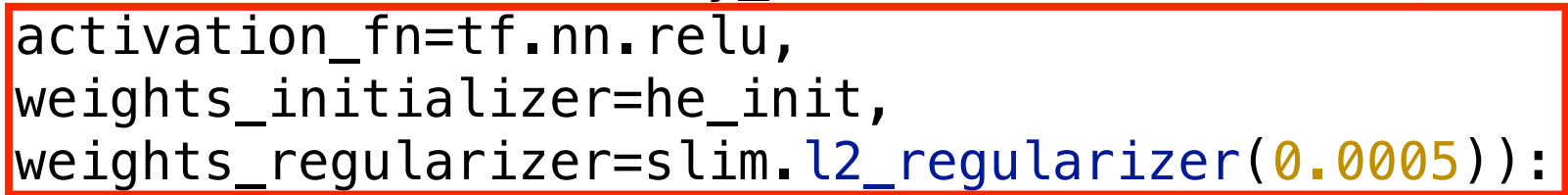
argscope

- 계속 반복되는 복붙코드 (boilerplate) 방지

```
he_init = slim.variance_scaling_initializer()
xavier_init = slim.xavier_initializer()
```

인자들이 함수에 주입됨 [...]

```
with slim.argscope([slim.conv2d, slim.fully_connected],
                    activation_fn=tf.nn.relu,
                    weights_initializer=he_init,
                    weights_regularizer=slim.l2_regularizer(0.0005)):
    with slim.argscope([slim.conv2d], stride=1, padding='SAME'):
        net = slim.conv2d(inputs, 64, [11, 11], 4, scope='conv1')
        net = slim.conv2d(net, 256, [5, 5],
                           weights_initializer=xavier_init,
                           scope='conv2')
        net = slim.fully_connected(net, 1000,
                                    activation_fn=None, scope='fc')
```



argscope

- 계속 반복되는 복붙코드 (boilerplate) 방지

```
he_init = slim.variance_scaling_initializer()
xavier_init = slim.xavier_initializer()
```

인자들이 함수에 주입됨 [...]

```
with slim.argscope([slim.conv2d, slim.fully_connected],
                    activation_fn=tf.nn.relu,
                    weights_initializer=he_init,
                    weights_regularizer=slim.l2_regularizer(0.0005)):
    with slim.argscope([slim.conv2d], stride=1, padding='SAME'):
        net = slim.conv2d(inputs, 64, [11, 11], 4, scope='conv1')
        net = slim.conv2d(net, 256, [5, 5],
                           weights_initializer=xavier_init,
                           scope='conv2')
        net = slim.fully_connected(net, 1000,
                                   activation_fn=None, scope='fc')
```

함수를 호출할 때 직접 넣어주면 인자가 덮어쓰기됨

Losses

- 여러 개의 loss 혹은 regularizer 텀을 위한 loss 모듈

전체 *loss* 정의하기

```
loss1 = slim.losses.softmax_cross_entropy(pred1, label1)
```

```
loss2 = slim.losses.mean_squared_error(pred2, label2)
```

아래 두줄은 같은 역할

```
total_loss = loss1 + loss2
```

```
slim.losses.get_total_loss(add_regularization_losses=False)
```

regularizer 텀 추가하기

```
reg_loss =
```

```
tf.add_n(slim.losses.get_regularization_losses())
```

```
total_loss = loss1 + loss2 + reg_loss
```

아니면 아래처럼

```
total_loss = slim.losses.get_total_loss()
```

모델 저장

```
saver = tf.train.Saver(max_to_keep=30)
sess = tf.Session()

dirname = os.path.join(ckpt_dir, name)
if not os.path.exists(dirname):
    os.makedirs(dirname)

for step in range(1000):
    if (step+1) % 100 == 0:
        saver.save(sess, dirname, global_step=step+1)
```

모델 불러오기

```
def load_latest_checkpoint(sess, ckpt_dir, exclude=None):  
    path = tf.train.latest_checkpoint(ckpt_dir)  
    if path is None:  
        raise AssertionError("No ckpt exists")  
  
    print("Load {} save file".format(path))  
    _load(sess, path, exclude)  
  
def load_from_path(sess, ckpt_path, exclude=None):  
    _load(sess, ckpt_path, exclude)  
  
def _load(sess, ckpt_path, exclude):  
    init_fn = slim.assign_from_checkpoint_fn(ckpt_path,  
        slim.get_variables_to_restore(exclude=exclude),  
        ignore_missing_vars=True)  
    init_fn(sess)
```

모델 불러오기

```
def load_latest_checkpoint(sess, ckpt_dir, exclude=None):  
    path = tf.train.latest_checkpoint(ckpt_dir)  
    if path is None:  
        raise AssertionError("No ckpt exists")  
  
    print("Load {} save file".format(path))  
    _load(sess, path, exclude)  
  
def load_from_path(sess, ckpt_path, exclude=None):  
    _load(sess, ckpt_path, exclude)  # 필요없는 레이어를 제외하고 불러오기  
  
def _load(sess, ckpt_path, exclude):  
    init_fn = slim.assign_from_checkpoint_fn(ckpt_path,  
        slim.get_variables_to_restore(exclude=exclude),  
        ignore_missing_vars=True)  
    init_fn(sess)
```

모델 불러오기

```
def load_latest_checkpoint(sess, ckpt_dir, exclude=None):  
    path = tf.train.latest_checkpoint(ckpt_dir)  
    if path is None:  
        raise AssertionError("No ckpt exists")  
  
    print("Load {} save file".format(path))  
    _load(sess, path, exclude)  
  
def load_from_path(sess, ckpt_path, exclude=None):  
    _load(sess, ckpt_path, exclude)  # 필요없는 레이어를 제외하고 불러오기  
  
def _load(sess, ckpt_path, exclude):  
    init_fn = slim.assign_from_checkpoint_fn(ckpt_path,  
        slim.get_variables_to_restore(exclude=exclude),  
        ignore_missing_vars=True)  
    init_fn(sess)
```

False일 경우:

체크포인트 파일에만 존재하는 파라미터 발견시 에러
fine-tuning시 에러를 피하기 위해 True로 설정해야함

사전 학습된 모델 불러오기

```
X = tf.placeholder(tf.float32, [None, 224, 224, 3], name="X")
y = tf.placeholder(tf.int32, [None, 8], name="y")
is_training = tf.placeholder(tf.bool, name="is_training")
with slim.arg_scope(vgg.vgg_arg_scope()):
    net, end_pts = vgg.vgg_16(X, is_training=is_training,
                              num_classes=10)

with tf.variable_scope("losses"):
    cls_loss = slim.losses.softmax_cross_entropy(net, y)
    reg_loss = tf.add_n(slim.losses.get_regularization_losses())
    loss_op = cls_loss + reg_loss

with tf.variable_scope("opt"):
    opt = tf.train.AdamOptimizer(0.001).minimize(loss_op)

self.load_from_path(ckpt_path=VGG_PATH, exclude=["vgg_16/fc8"])
...
```

사전 학습된 모델 불러오기

```
X = tf.placeholder(tf.float32, [None, 224, 224, 3], name="X")
y = tf.placeholder(tf.int32, [None, 8], name="y")
is_training = tf.placeholder(tf.bool, name="is_training")
with slim.arg_scope(vgg.vgg_arg_scope()):
    net, end_pts = vgg.vgg_16(X, is_training=is_training,
                              num_classes=10)

with tf.variable_scope("losses"):
    cls_loss = slim.losses.softmax_cross_entropy(net, y)
    reg_loss = tf.add_n(slim.losses.get_regularization_losses())
    loss_op = cls_loss + reg_loss

with tf.variable_scope("opt"):
    opt = tf.train.AdamOptimizer(0.001).minimize(loss_op)

self.load_from_path(ckpt_path=VGG_PATH, exclude=["vgg_16/fc8"])
...
```

fc8 레이어는 처음부터 학습하기 위해 (fine-tuning)

