TensorFlow Basic

Youngjae Yu

Aug 2, 2017

https://github.com/yj-yu/tensorflow-basic

https://yj-yu.github.io/tensorflow-basic

SEOUL NATIONAL UNIV.
VISION & LEARNING

About

- TensorFlow Basic - Op, Graph, Session, Feed 등
- Logistic Regression using TensorFlow
- `tf.flags`, Tensorboard 등 Minor tips
- Variable Saving, Restoring

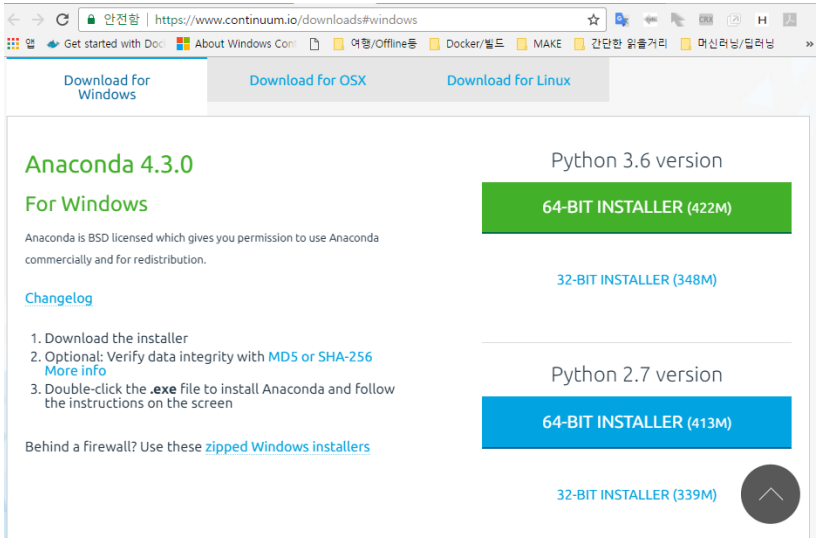
TensorFlow Basic

Install anaconda

Install instruction for windows OS
https://www.tensorflow.org/install/install_windows

Anaconda에 있는 배포판에 numpy 등 기본 라이브러리를 기본적으로 포함

아래 링크에서 Python 3.x 버전으로 설치를 해주세요.
https://www.continuum.io/downloads#windows



Install anaconda

Windows 키를 누른뒤에 anaconda prompt 입력하면 console이 뜹니다.

conda 라는 명령어로 여러 개의 가상환경을 만들 수 있습니다.

```
conda create --name tf python=3.6
```

```
#tf라는 환경이 만들어졌는지 확인
conda info --envs
```

Install anaconda

자 이제 기본 실습환경 세팅을 위해 tf라는 가상 환경으로 들어갑니다.

```
activate tf
#만약 비활성화하고 싶다면 deactivate tf를 치세요.

#기본 라이브러리 확인
conda list
```

pip가 보이시죠? 이제 pip를 통해 tensorflow 및 실습 환경을 위한 라이브러리를 추가할 것입니다. 다음 명령어들을 입력하여 자동으로 tensorflow 최신 배포판을 설치합니다.

```
pip install ipython
pip install jupyter
pip install tensorflow
```

Install configuration

```
git clone https://github.com/yj-yu/tensorflow-basic.git
cd tensorflow-basic
ls
```

code(https://github.com/yj-yu/tensorflow-basic)

```
./code
├── train.py
├── train_quiz1.py
├── train_quiz2.py
└── eval.py
```

- `train.py` : basic regression model code
- `train_quiz1.py` : quiz1 정답을 포함
- `train_quiz2.py` : quiz2 정답을 포함
- `eval.py` : quiz3 정답을 포함

Tensor

데이터 저장의 기본 단위

```
import tensorflow as tf
a = tf.constant(1.0, dtype=tf.float32) # 1.0 의 값을 갖는 1차원 Tensor 생성
b = tf.constant(1.0, shape=[3,4]) # 1.0 의 값을 갖는 3x4 2차원 Tensor 생성
c = tf.constant(1.0, shape=[3,4,5]) # 1.0 의 값을 갖는 3x4x5 3차원 Tensor 생성
d = tf.random_normal(shape=[3,4,5]) # Gaussian Distribution 에서 3x4x5 Tensor를 Sampling

print (c)
```

```
<tf.Tensor 'Const_24:0' shape=(3, 4, 5) dtype=float32>
```

TensorFlow Basic

TensorFlow Programming의 개념

1. `tf.Placeholder` 또는 Input Tensor 를 정의하여 Input **Node**를 구성한다
2. Input Node에서 부터 Output Node까지 이어지는 관계를 정의하여 **Graph**를 그린다
3. **Session**을 이용하여 Input Node(`tf.Placeholder`)에 값을 주입(feeding) 하고, **Graph**를 **Run** 시킨다

TensorFlow Basic

1과 2를 더하여 3을 출력하는 프로그램을 작성

Tensor들로 Input Node를 구성한다

```
import tensorflow as tf
a = tf.constant(1) # 1의 값을 갖는 Tensor a 생성
b = tf.constant(2) # 2의 값을 갖는 Tensor b 생성
```

TensorFlow Basic

1과 2를 더하여 3을 출력하는 프로그램을 작성

Output Node까지 이어지는 **Graph**를 그린다

```
import tensorflow as tf
a = tf.constant(1) # 1의 값을 갖는 Tensor a 생성
b = tf.constant(2) # 2의 값을 갖는 Tensor b 생성

c = tf.add(a,b) # a + b의 값을 갖는 Tensor c 생성
```

TensorFlow Basic

1과 2를 더하여 3을 출력하는 프로그램을 작성

Session을 이용하여 **Graph**를 **Run** 시킨다

```
import tensorflow as tf
a = tf.constant(1) # 1의 값을 갖는 Tensor a 생성
b = tf.constant(2) # 2의 값을 갖는 Tensor b 생성

c = tf.add(a,b) # a + b의 값을 갖는 Tensor c 생성

sess = tf.Session() # Session 생성

# Session을 이용하여 구하고자 하는 Tensor c를 run
print (sess.run(c)) # 3
```

Tip. native operation op `+,-,*,/` 는 TensorFlow Op 처럼 사용가능

```
c = tf.add(a,b) <-> c = a + b
c = tf.subtract(a,b) <-> c = a - b
c = tf.multiply(a,b) <-> c = a * b
c = tf.div(a,b) <-> c = a / b
```

Exploring in Tensor: Tensor name

```
import tensorflow as tf
a = tf.constant(1)
b = tf.constant(2)
c = tf.add(a,b)
sess = tf.Session()

print (a, b, c, sess)
print (sess.run(c)) # 3
```

Tensor("Const:0", shape=(), dtype=int32 , Tensor("Const_1:0", shape=(), dtype=int32), Tensor("Add:0", shape=(), dtype=int32)

3

모든 텐서는 op **name**으로 구분 및 접근되어서, 이후 원하는 텐서를 가져오는 경우나, 저장(Save)/복원(Restore) 또는 재사용(reuse) 할 때에도 name으로 접근하기 때문에 텐서 name 다루는 것에 익숙해지는 것이 좋습니다

Placeholder: Session runtime에 동적으로 Tensor의 값을 주입하기

Placeholder: 선언 당시에 값은 비어있고, 형태(shape)와 타입(dtype)만 정의되어 있어 Session runtime에 지정한 값으로 텐서를 채울 수 있음

Feed: Placeholder에 원하는 값을 주입하는 것

```
a = tf.placeholder(dtype=tf.float32, shape=[]) # 1차원 실수형 Placeholder 생성
b = tf.placeholder(dtype=tf.float32, shape=[]) # 1차원 실수형 Placeholder 생성
c = a + b
with tf.Session() as sess:
  feed = {a:1, b:2} # python dictionary
  print (sess.run(c, feed_dict=feed)) # 3

  feed = {a:2, b:4.5}
  print (sess.run(c, feed_dict=feed)) # 6.5
```

Quiz 0.

1. 3x4 행렬에 대한 Placeholder `a` 와 4x6 행렬에 대한 Placeholder `b` 를 선언한다.
2. 행렬 a와 b를 곱하여 3x6 행렬 c로 이어지는 그래프를 그린다.
3. `numpy.random.randn` 함수로 `a, b`에 대한 랜덤 feed를 만들고, `Session`을 이용하여 랜덤값으로 채운 `a, b` 에 대한 `c`의 값을 출력한다.

Tip.

```
import numpy as np
a = np.random.randn(2,3)
print (a)
```

Variable: 학습하고자 하는 모델의 Parameter

Variable과 Constant/Placeholder의 차이점: 텐서의 값이 변할 수 있느냐 없느냐의 여부

Parameter `W, b` 를 `1.0` 으로 **초기화** 한 후 linear model의 출력 구하기

```
W = tf.Variable(1.0, dtype=tf.float32)
b = tf.Variable(1.0, dtype=tf.float32)
x = tf.placeholder(dtype=tf.float32, shape=[])

linear_model_output = W * x + b

# Important!!
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
  sess.run(init_op)

  feed = {x:5.0}
  sess.run(linear_model_output, feed_dict=feed) # 6
```

만약 `sess.run` 하는 op의 그래프에 변수(`tf.Variable`)이 하나라도 포함되어 있다면, 반드시 해당 변수를 초기화 `tf.global_variables_initializer()` 를 먼저

Variable: 학습하고자 하는 모델의 Parameter

Parameter `W, b` 를 **랜덤** 으로 **초기화** 한 후 linear model의 출력 구하기

```
W = tf.Variable(tf.random_normal(shape=[]), dtype=tf.float32)
b = tf.Variable(tf.random_normal(shape=[]), dtype=tf.float32)
x = tf.placeholder(dtype=tf.float32, shape=[])

linear_model_output = W * x + b

# Important!!
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
  sess.run(init_op)

  feed = {x:5.0}
  sess.run(linear_model_output, feed_dict=feed) # 6
```

MNIST using Logistic Regression

Code(https://https://github.com/yj-yu/tensorflow-basic)

MNIST

Image Classsification Dataset

0 ~ 9까지의 손글씨 이미지를 알맞은 label로 분류하는 Task

Example. MNIST Using Logistic Regression

1. 모델의 입력 및 출력 정의
2. 모델 구성하기(Logistic Regression model)
3. Training

모델의 입력 및 출력 정의

Input: 28*28 이미지 = 784차원 벡터 `model_input = [0, 255, 214, ...]`

각각에 해당하는 정답 `labels = [0.0, 1.0, 0.0, 0.0, ...]`

Output: 이미지가 각 클래스에 속할 확률 예측값을 나타내는 10차원 벡터
`predictions = [0.12, 0.311, ...]`

하고싶은 것은?

모델의 예측값이 정답 데이터(Label 또는 Ground-truth)와 최대한 비슷해지도록 모델 Parameter를 학습시키고 싶다 <-> `label` 과 `predictions` 의 **오차를 최소화 하고 싶다**

모델의 입력 및 출력 정의

데이터 준비

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./data", one_hot=True)
```

```
for _ in range(10000):
  batch_images, batch_labels = mnist.train.next_batch(100)
  batch_images_val, batch_labels_val = mnist.validation.next_batch(100)
  print (batch_image.shape) # [100, 784]
  print (batch_labels.shape) # [100, 10]
```

모델 구성하기

모델의 입력을 Placeholder로 구성

Batch 단위로 학습할 것이기 때문에 `None`을 이용하여 임의의 batch size를 핸들링 할 수 있도록 합니다

```
# defien model input: image and ground-truth label
model_inputs = tf.placeholder(dtype=tf.float32, shape=[None, 784])
labels = tf.placeholder(dtype=tf.float32, shape=[None, 10])
```

Logistic Regression Model의 Parameter 정의

```
# define parameters for Logistic Regression model
w = tf.Variable(tf.random_normal(shape=[784, 10]))
b = tf.Variable(tf.random_normal(shape=[10]))
```

모델 구성하기

그래프 그리기

```
logits = tf.matmul(model_inputs, w) + b
predictions = tf.nn.softmax(logits)

# define cross entropy loss term
loss = tf.losses.softmax_cross_entropy(
         onehot_labels=labels,
         logits=predictions)
```

모델 구성하기

Optimizer 정의 -> 모델이 loss(predictions 와 labels 사이의 차이)를 최소화 하는 방향으로 파라미터 업데이트를 했으면 좋겠다

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train_op = optimizer.minimize(loss)
```

Training

Session을 이용하여

`Variable`들을 초기화시켜준 후에

각 iteration마다 이미지와 라벨 데이터를 batch단위로 가져오고

가져온 데이터를 이용하여 feed를 구성

`train_op`(가져온 데이터에 대한 loss를 최소화 하도록 파라미터 업데이트를 하는 Op)을 실행

```
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for step in range(10000):
    batch_images, batch_labels = mnist.train.next_batch(100)
    feed = {model_inputs: batch_images, labels: batch_labels}
    _, loss_val = sess.run([train_op, loss], feed_dict=feed)
    print ("step {}| loss : {}".format(step, loss_val))
```

Minor Tips - `tensorflow.flags`

TensorFlow에서 FLAGS를 통한 argparsing 기능도 제공하고 있습니다.
HyperParamter(batch size, learning rate, max_step 등) 세팅에 유용!

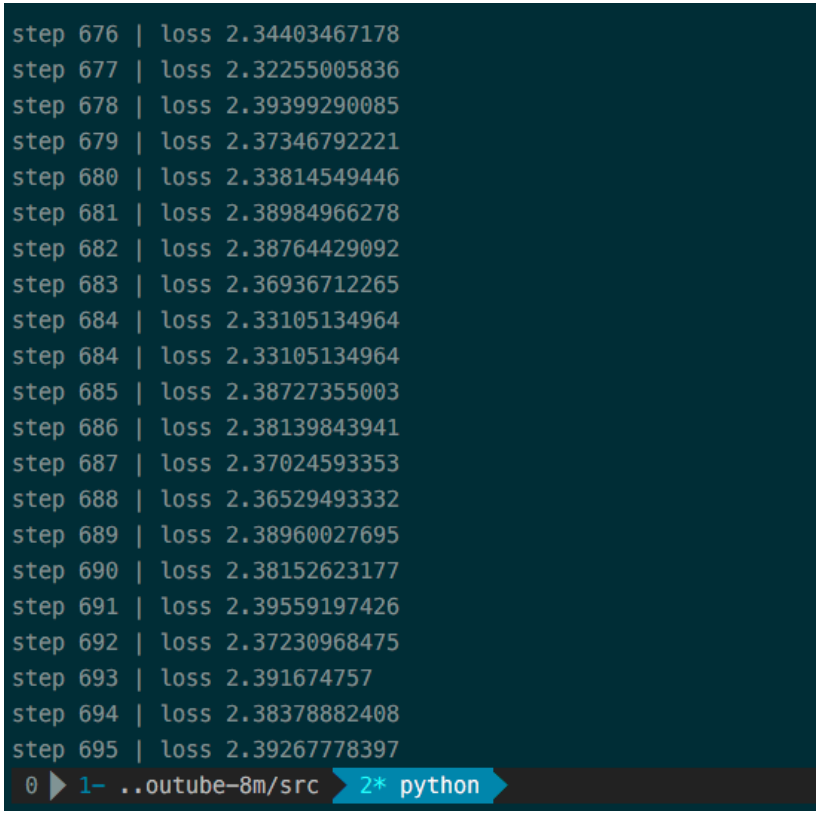
```
from tensorflow import flags
FLAGS = flags.FLAGS

flags.DEFINE_integer("batch_size", 128, "number of batch size. default 128.")
flags.DEFINE_float("learning_rate", 0.01, "initial learning rate.")
flags.DEFINE_integer("max_steps", 10000, "max steps to train.")
```

```
# train.py
batch_size = FLAGS.batch_size
learning_rate = FLAGS.learning_rate
max_step = FLAGS.max_steps
```

```
$ python train.py --batch_size=256 --learning_rate=0.001 --
max_steps=100000
```

Result

```
$ python train.py --batch_size=128 --learning_rate=0.01 --max_steps=10000
```



Minor Tips - Tensorboard

학습 진행 상황을 Visualize 하고 싶다면? -> Tensoroboard 사용

Minor Tips - Tensorboard

scalar summary와 histogram summary

loss, learning rate 등 scalar 값을 가지는 텐서들은 scalar summary로, parameter 등 n차원 텐서들은 histogram summary로 선언한다

```
tf.summary.scalar("loss", loss)
tf.summary.histogram("W", w)
tf.summary.histogram("b", b)
```

merge_all() 로 summary 모으기

```
merge_op = tf.summary.merge_all()
```

Minor Tips - Tensorboard

이 후, `tf.summary.FileWriter` 객체를 선언하고, Session으로 `merge_op`을 실행하여 Summary를 얻고, `FileWriter`에 추가

```
summary_writer = tf.summary.FileWriter("./logs", sess.graph)
for step in range(10000):
  # some training code...
  sess.run(train_op, feed=...)
  if step % 10 == 0:
    # session으로 merge_op을 실행시켜 summary를 얻고
    summary = sess.run(merge_op, feed_dict=feed)

    # summary_writer 에 얻은 summary값을 추가
    summary_writer.add_summary(summary, step)
```

Minor Tips - Tensorboard

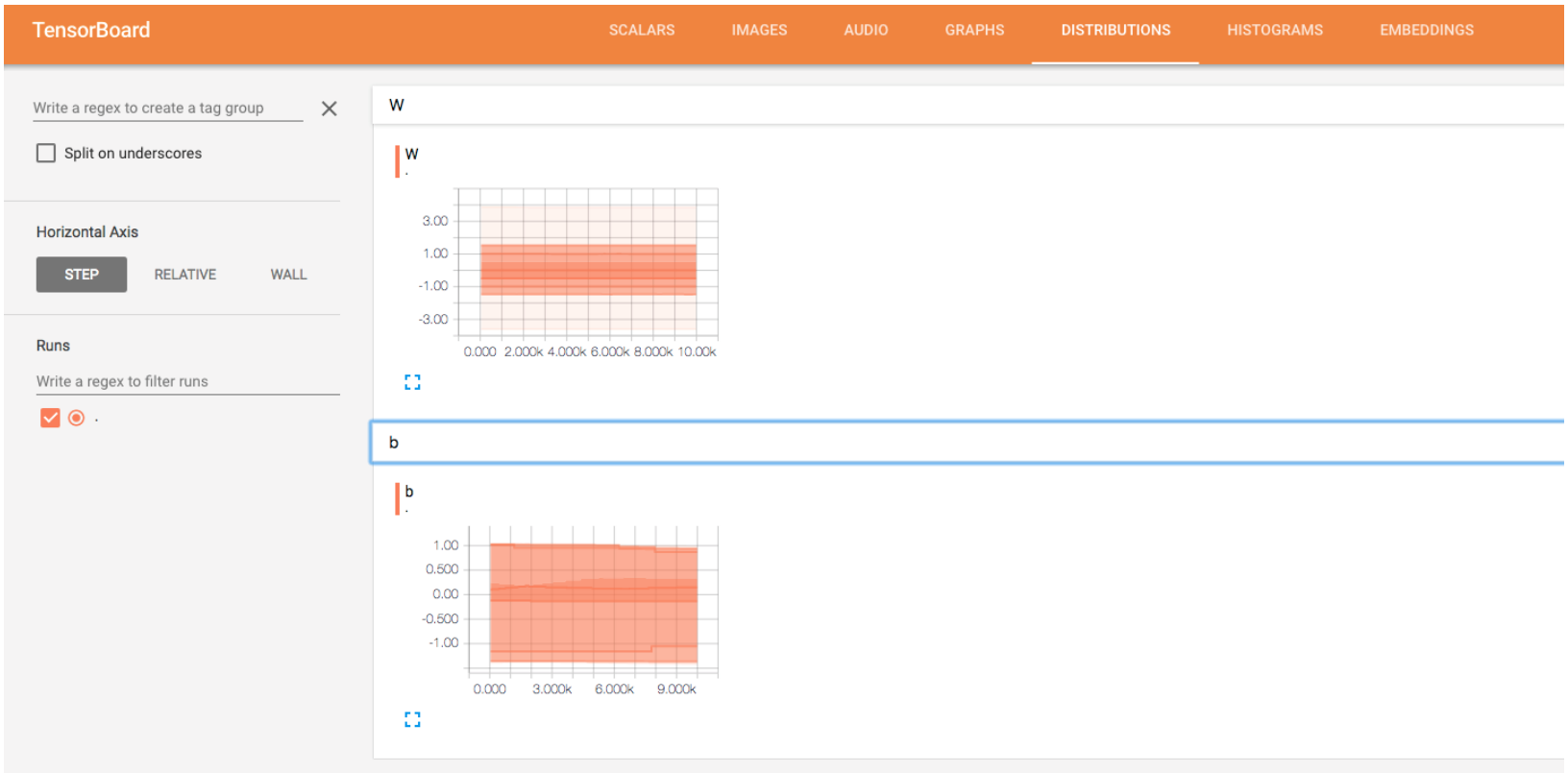
`$ tensorboard --logdir="./logs" --port=9000` 입력

& `localhost:9000` 접속

scalar summary

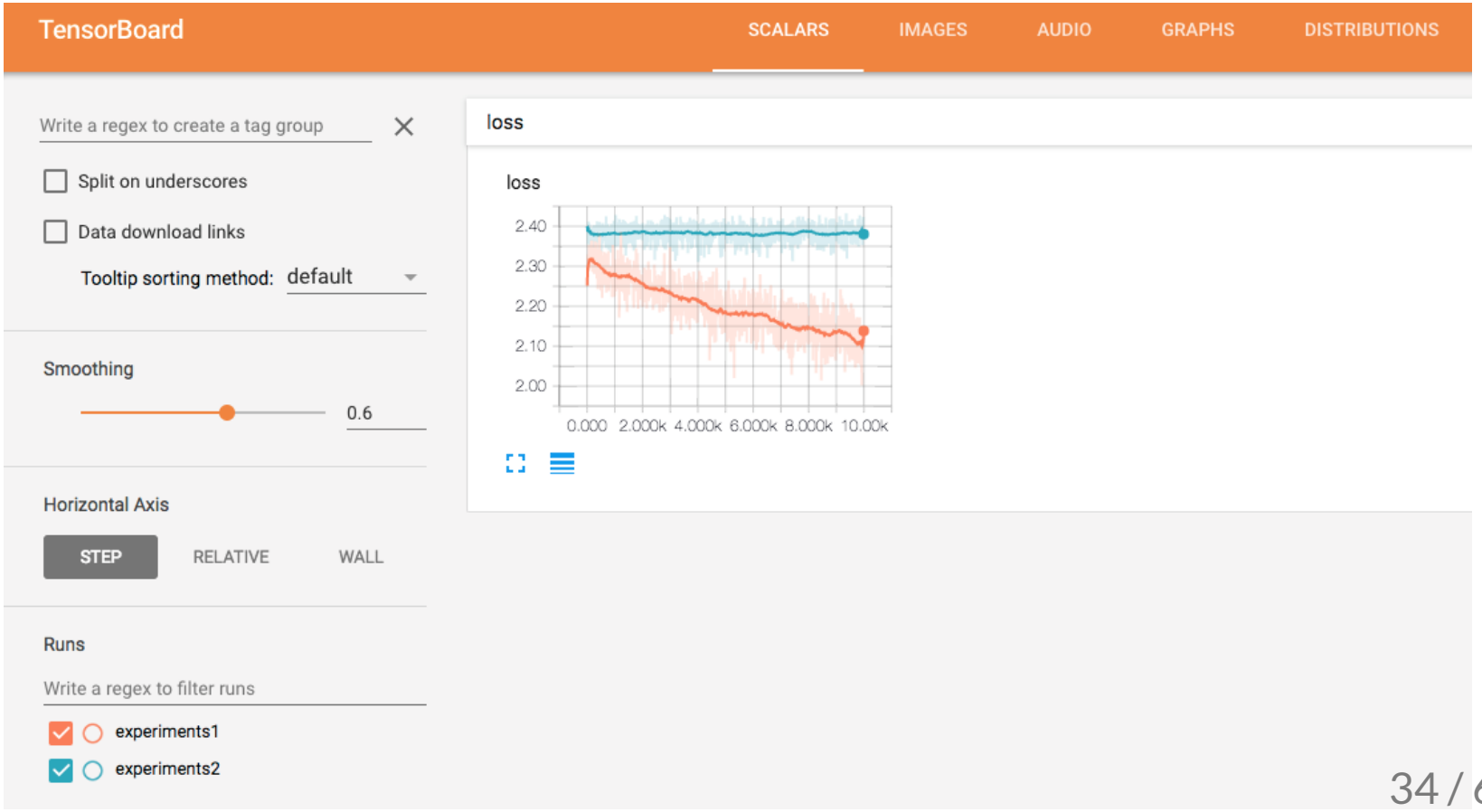
Minor Tips - Tensorboard

histogram summary



Minor Tips - Tensorboard

summary 폴더 여러 개를 두고 서로 다른 실험 결과를 실시간으로 비교할 수도 있습니다 (여러 실험 결과값을 비교해볼 때 편리)

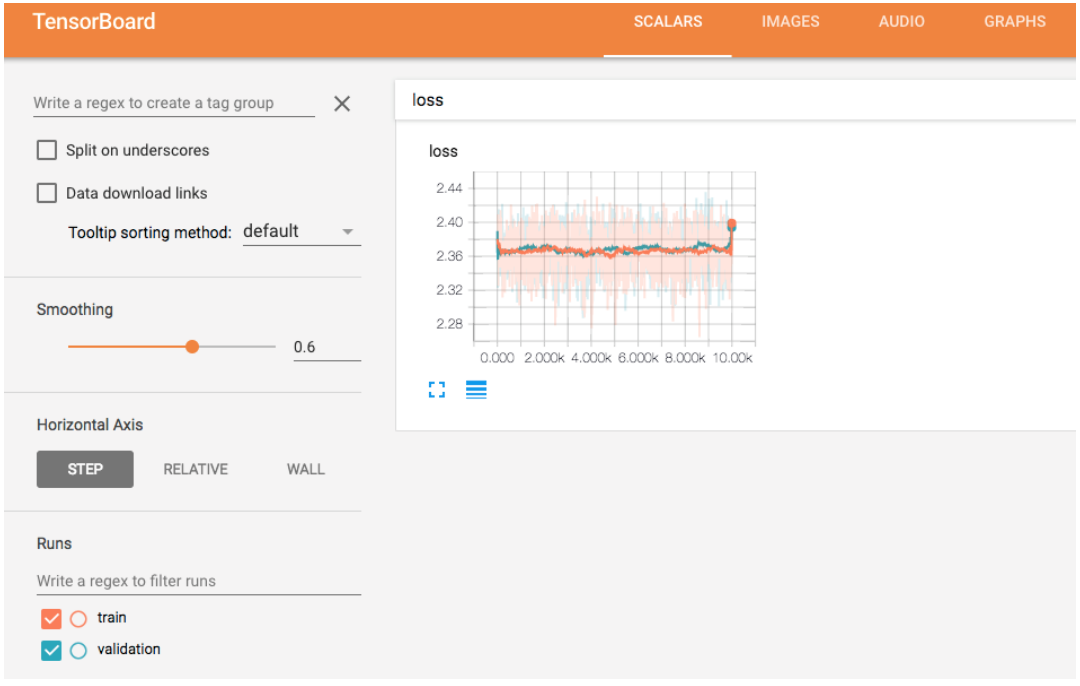


Quiz 1.

`tf.argmax` `tf.equal` `tf.cast` `tf.reduce_mean` 을 사용하여 Accuracy Tensor를 정의하고, 이를 Tensorboard에 나타내기

Quiz 2.

모델을 트레이닝 할 때, `tf.summary.FileWriter` 를 train, validation 용으로 각각 1개씩 만들어서 Tensorboard로 Training/Validation performance 를 함께 모니터링 할 수 있도록 해보기



Variable Saving & Restoring

Variable Saving, Restoring

학습은 어찌저찌 잘 했는데...

우리의 목적은 모델 학습 그 자체가 아님!

학습한 모델을 이용하여 새로운 입력 X 에 대하여 그에 알맞은 출력을 내는 것이 원래 목표였습니다.

그렇다면, Training Phase에서 모델이 학습한 Parameter들의 값을 디스크에 저장해놓고, 나중에 불러올 수 있어야겠다.

`tf.train.Saver` 모듈을 통해서 이와 같은 기능을 수행할 수 있습니다.

Variable name

시작하기 전에... 처음에 배웠던 Tensor name에 대해서 자세히 알아야 합니다.

모든 텐서는 선언하는 시점에 이름이 자동으로 부여되며, 중복되지 않습니다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10]))
b = tf.Variable(tf.zeros(shape=[10]))
print (a.name)
print (b.name)
```

Variable:0

Variable_1:0

Variable name

`name` 을 통해서 이름을 명시적으로 지정할 수도 있지만, 같은 이름으로 지정된 경우 중복을 피하기위해 자동으로 인덱스가 붙습니다.

```
c = tf.Variable(tf.ones(shape=[10]), name="my_variable")
d = tf.Variable(tf.zeros(shape=[]), name="my_variable")

print (c.name)
print (d.name)
```

my_variable:0

my_variable_1:0

Variable name

`name` 을 통해서 이름을 명시적으로 지정할 수도 있지만, 같은 이름으로 지정된 경우 중복을 피하기위해 자동으로 인덱스가 붙습니다.

```
c = tf.Variable(tf.ones(shape=[10]), name="my_variable")
d = tf.Variable(tf.zeros(shape=[]), name="my_variable")

print (c.name)
print (d.name)
```

한줄 요약: 모든 텐서에는 중복되지 않게 이름이 부여된다.

Variable Saving, Restoring

그럼 이제, `tf.train.Saver` 객체를 이용해 변수 저장을 해봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  # some training code...
  save_path = saver.save(sess, "./logs/model.ckpt")
```

변수 a와 b를 선언합니다. 이름을 따로 지정해주지 않았으므로 `Variable_0:0` 과 같이 자동으로 지정됩니다.

Variable Saving, Restoring

그럼 이제, `tf.train.Saver` 객체를 이용해 변수 저장을 해봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  # some training code...
  save_path = saver.save(sess, "./logs/model.ckpt")
```

Saver 객체를 생성합니다. Saver 객체 안에 아무런 파라미터가 없다면, 기본값으로 Saver 객체는 `{key="Variable name", value=Variable Tensor}` 쌍의 dictionary를 내부적으로 가지게 됩니다.

즉, 이 경우에 Saver 객체가 가지고 있는 dictionary는 `{"Variable_0:0":a, "Variable_1:0":b}` 가 됩니다.

Variable Saving, Restoring

그럼 이제, `tf.train.Saver` 객체를 이용해 변수 저장을 해봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  # some training code...
  save_path = saver.save(sess, "./logs/model.ckpt")
```

initializer 를 실행시키면 Variable `a` `b`에 값이 할당됩니다.

Variable Saving, Restoring

그럼 이제, `tf.train.Saver` 객체를 이용해 변수 저장을 해봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  # some training code...
  save_path = saver.save(sess, "./logs/model.ckpt")
```

현재 Saver 객체가 가지고 있는 dictionary 정보를 디스크의 `"./logs/model.ckpt"` 이름으로 저장(save)합니다. 저장된 파일을 checkpoint 라고 부릅니다.

Variable Saving, Restoring

그럼 이제, `tf.train.Saver` 객체를 이용해 변수 저장을 해봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  # some training code...
  save_path = saver.save(sess, "./logs/model.ckpt")
```

다음과 같이 저장되어 있는 것을 확인할 수 있습니다.

```
./
├── train.py
└── logs
    ├──checkpoint
    ├──model.ckpt.data-00000-of-00001
    ├──model.ckpt.index
    └──model.ckpt.meta
```

Variable Saving, Restoring

그럼 이제, `tf.train.Saver` 객체를 이용해 변수 저장을 해봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  # some training code...
  save_path = saver.save(sess, "./logs/model.ckpt", global_step=1000)
```

`global_step` 인자를 통해서 현재 트레이닝 i번째 스텝의 파라미터 값을 가지고 있는 체크포인트임을 명시할 수 있습니다.

```
./
├── train.py
└── logs
    ├──checkpoint
    ├──model.ckpt-1000.data-00000-of-00001
    ├──model.ckpt-1000.index
    └──model.ckpt-1000.meta
```

Variable Saving, Restoring

checkpoint를 저장했으니, 저장한 checkpoint를 불러와 기록되어있는 파라미터 값으로 변수 값을 채워봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  # some training code...
  saver.restore(sess, "./logs/model.ckpt-1000")
  # sess.run(tf.global_variables_initializer())
```

변수 `a, b`를 생성하고 Saver 객체를 생성합니다.

Saver 객체가 인자 없이 선언되었으니, 생성된 모든 변수들에 대한 dictionary를 가지고 있습니다: `{"Variable_0:0":a, "Variable_1:0":b}`

Variable Saving, Restoring

checkpoint를 저장했으니, 저장한 checkpoint를 불러와 기록되어있는 파라미터 값으로 변수 값을 채워봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  # some training code...
  saver.restore(sess, "./logs/model.ckpt-1000")
  # sess.run(tf.global_variables_initializer())
```

checkpoint 파일의 이름을 인자로 넣어 저장된 파라미터 값을 불러옵니다.

이 시점에서, saver 객체가 가지고 있는 dictionary 의 key값을 checkpoint파일에서 찾고, 매칭되는 checkpoint 파일의 key값이 존재한다면, 해당 value 텐서의 값을 saver 객체가 가지고 있는 dictionary의 value 에 할당합니다.

Variable Saving, Restoring

checkpoint를 저장했으니, 저장한 checkpoint를 불러와 기록되어있는 파라미터 값으로 변수 값을 채워봅시다.

```
import tensorflow as tf
a = tf.Variable(tf.random_normal(shape=[10])) #a.name="Variable_0:0"
b = tf.Variable(tf.random_normal(shape=[5])) # b.name="Variable_1:0"
saver = tf.train.Saver()
with tf.Session() as sess:
  # some training code...
  saver.restore(sess, "./logs/model.ckpt-1000")
  # sess.run(tf.global_variables_initializer())
```

variable initializer를 restoring 이후에 run 하지 않는다는 사실에 주의해야 합니다.

만약 restoring 이후에 initializer run을 하게 되면, 불러온 파라미터 값이 전부 지워지고 원래 변수의 initializer로 초기화됩니다.

Quiz 3.

1. MNIST에 모델을 트레이닝하고, checkpoint파일을 저장합니다.
2. `eval.py` 파일을 만들고, 그래프를 그린 후 저장한 checkpoint 파일을 restore 합니다.
3. 전체 Validation data에 대해서 불러온 파라미터 값을 가지는 모델을 Fully Evaluation하는(전체 Validation data 대한 Accuracy) 코드를 작성해 봅시다.

Tip. Validation data는 5000개 Image/Label pair이고, `batch_size=100` 으로 50 iteration을 돌려서 Accuracy를 평균내면 됩니다.

Deep Neural Network using TensorFlow

시작하기 전에...

데이터 준비

```
mnist = input_data.read_data_sets( # data loading...)
```

그래프 그리기

```
x = tf.placeholder(dtype=tf.float32, shape=[None, 784]
# ...
logits = tf.matmul() # ...
# ...
predictions = # ...
```

- 모델 부분만 빼면 `part1/train.py` 코드와 대부분 중복된다
- 모델 코드와 트레이닝 코드를 분리하면 각 컴포넌트를 수정하기 매우 편리해짐
- 코드를 `models.py` 와 `train.py` 로 분리해보자!

Code structure

```
./code-part2
├── train.py
└── models.py
```

- `train.py` : 모델 코드를 제외하고 Loss 계산, Optimizer 정의 및 학습 코드를 포함한다
- `models.py` : class 형태의 모델 코드를 포함한다.

DNN

Input 텐서들을 입력으로 받아, predictions 를 출력으로 하는 구조의 모델

```
# models.py
class DNN(object):
  def create_model(self, model_inputs):
    # model architectures here!
    # ...

    return predictions
```

DNN

필요한 파라미터 선언

```
# models.py
def create_model(model_inputs):
  initializer = tf.random_normal
  w1 = tf.Variable(initializer(shape=[784, 128]))
  b1 = tf.Variable(initializer(shape=[128]))

  w2 = tf.Variable(initializer(shape=[128, 10]))
  b2 = tf.Variable(initializer(shape=[10]))
```

DNN

그래프 그리기

```
# models.py
h1 = tf.nn.relu(tf.matmul(model_inputs, w1) + b1) # 1st hidden layer

logits = tf.matmul(h1,  w2) + b2
predictions = tf.nn.softmax(logits)

return predictions
```

`models.py` 안에 있는 모델 class가 input tensor를 argument를 받아 그에 대한 output(`predictions`)를 리턴하도록 합니다

DNN

Trainer - data reader, 모델 불러오기, train_op 정의, Session run 등

```
# train.py
mnist = input_data.read_data_sets("./data", one_hot=True)

# define model input: image and ground-truth label
model_inputs = tf.placeholder(dtype=tf.float32, shape=[None, 784])
labels = tf.placeholder(dtype=tf.float32, shape=[None, 10])
```

DNN

모델 불러오기 `getattr` 함수 사용

```
# train.py
import models
models = getattr(models, "DNN", None)
predictions = models.create_model(model_inputs)
```

모델이 여러개인 경우에, 다음과 같이 `tf.flags` 모듈을 이용하여 argparse로 사용할 모델을 선택하면 편리합니다. (대신 코드의 일관성을 위해 반드시 모든 모델의 입출력 포맷이 같아야 함)

```
# train.py
import models
models = getattr(models, flags.model, None)
predictions = models.create_model(model_inputs)
```

```
$ python train.py --model=DNN
```

```
$ python train.py --model=LogisticRegression
```

DNN

loss & train op 정의

```
# train.py
# define cross entropy loss term
loss = tf.losses.softmax_cross_entropy(
    onehot_labels=labels,
    logits=predictions)

# train.py 안에서 정의되는 텐서들에 대하여 summary 생성
tf.summary.scalar("loss", loss)
merge_op = tf.summary.merge_all()

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.0001)
train_op = optimizer.minimize(loss)
```

DNN

Session으로 Training 실행

```
with tf.Session() as sess:
  summary_writer_train = tf.summary.FileWriter("./logs/train", sess.graph)

  sess.run(tf.global_variables_initializer())
  for step in range(10000):
    batch_images, batch_labels = mnist.train.next_batch(100)
    feed = {model_inputs: batch_images, labels: batch_labels}
    _, loss_val = sess.run([train_op, loss], feed_dict=feed)
    print ("step {} | loss {}".format(step, loss_val))

    if step % 10 == 0:
      summary_train = sess.run(merge_op, feed_dict=feed)
      summary_writer_train.add_summary(summary_train, step)
```

DNN

Hmm...

DNN

더 잘할 수 없을까?

모델의 구조를 이것저것 바꿔봅시다

- Hidden Layer 의 개수
- 각 Hidden Layer 의 차원(Dimension)
- Learning rate
- Optimizer 종류 (`tf.train.GradientDescentOptimizer`, `tf.train.AdamOptimizer`, ...)
- Batch size

등등...

Do it!

Thank You!

Special Thanks to: Seil Na, Jongwook Choi, Byungchang Kim

Slideshow created using remark.