딥러닝 연습 - Tensorflow 기초 1

Lecture Notes on 인공지능입문, Spring 2018

Made by JeHwan Ryu

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National Univertisy

http://bi.snu.ac.kr

TensorFlow

TensorFlow란?

- Open source software library for **numerical computation** using **data flow graphs**
- Originally developed by Google Brain Team to conduct machine learning and deep learning research
- General enough to be applicable in a wide variety of other domain as well

알고리즘을 표현하는 부분과 알고리즘 실행하는 부분을 분리

Data Flow Graph **Session**

TensorFlow

TensorFlow의 장점

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

- Automatic gradient 계산 가능
- CPU, GPU, Multi-GPU 환경을 모두 지원
- 높은 범용성 : 윈도우, 맥, 리눅스, 안드로이드, iOS, 라즈베리 파이 등에서 실행 가능
- 넓은 사용자 커뮤니티

Environment setting

TensorFlow

- Installation: https://www.tensorflow.org/install/install_windows

To install TensorFlow, start a terminal. Then issue the appropriate `pip3 install` command in that terminal. To install the CPU-only version of TensorFlow, enter the following command:

```
C:\> pip3 install --upgrade tensorflow
```

To install the GPU version of TensorFlow, enter the following command:

```
C:\> pip3 install --upgrade tensorflow-gpu
```

- Verification

Enter the following short program inside the python interactive shell:

```
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
```

If the system outputs the following, then you are ready to begin writing TensorFlow programs:

```
Hello, TensorFlow!
```

TensorFlow

TensorFlow에서 중요한 세 가지

```
import tensorflow as tf
```

Tensor

Multi-dimensional numerical array

Data Flow Graph

구성하는 시스템을 표현 (알고리즘 구성)

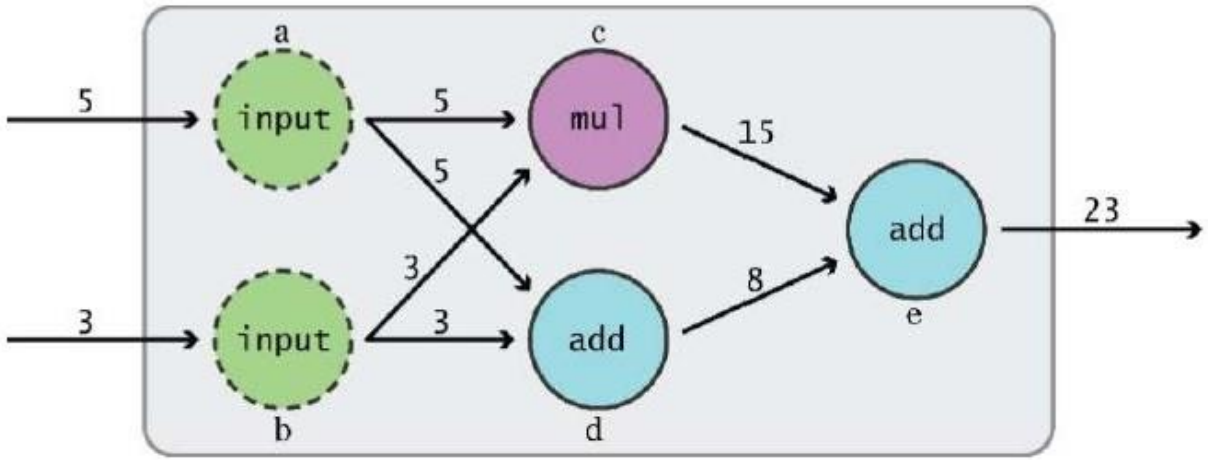
Session

Graph(알고리즘)의 실행

TensorFlow

Data flow graph

- **Computational graph**: series of TensorFlow **operations** arranged into a graph of nodes
- TensorFlow separates definition of computations from their execution



- 알고리즘의 구성
  1. Build a computational graph
  2. Run the computational graph

- 오퍼레이션(Operation)
  - 그래프 상의 노드. 계산을 수행하고 결과를 하나 이상의 텐서로 반환
- 텐서(Tensor)
  - 행렬의 일반화된 개념. 텐서플로우 내부의 모든 데이터는 텐서 형태로 표현.
- 세션(Session)
  - 오퍼레이션의 실행 환경을 캡슐화한 것.

- 자료형
  - 상수형(Constant) : 변하지 않는 값을 표현하는 데이터형.
  - 플레이스 홀더(Placeholder) : 가변적인 입력을 받는 값을 표현하는 데이터형.
  - 변수(Variable) : 변할 수 있는 값을 표현하는 데이터형.

The computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

print 결과:

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

The computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

print 결과:

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

Session을 열고 run 해야 tensor의 값을 획득 가능!

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

print 결과:

```
[3.0, 4.0]
```

The computational graph

수학 연산(operation)은 tensor를 반환!

```
node3 = tf.add(node1, node2)
print("node3:", node3)
print("sess.run(node3):", sess.run(node3))
```

print 결과:

```
node3: Tensor("Add:0", shape=(), dtype=float32)
sess.run(node3): 7.0
```

Reference: CS20SI – lecture 1. Overview of Tensorflow (http://web.stanford.edu/class/cs20si/syllabus.html)

TensorFlow

Session

```
import tensorflow as tf

a = tf.add(3, 5)

sess = tf.Session()

print sess.run(a)        >> 8

sess.close()
```

sess.close()

Graph를 실행하기 위하여 sess = tf.Session() 으로 session을 열면 sess.close() 으로 세션을 닫는 과정이 필요!

```
import tensorflow as tf

a = tf.add(3, 5)

sess = tf.Session()

with tf.Session() as sess:

      print sess.run(a)

sess.close()
```

with tf.Session as sess:

sess = tf.Session()을 with tf.Session as sess: 의 코드 블럭으로 대체 가능

Reference: CS20SI – lecture 1. Overview of Tensorflow (http://web.stanford.edu/class/cs20si/syllabus.html)

Constant

- tf.constant(value, dtype=None, shape=None, name='Const', verify_shape=False)
- value : 상수의 값,

dtype : 상수의 데이터 타입(실수, 정수 등),
shape : 상수의 형태([2, 3, 4]의 경우 2x3x4의 크기를 갖는 3차원 텐서,
name : 해당 tensor의 name(코딩환경이 아닌 텐서플로우 그래프 내의 이름)

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

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

print 결과:

```
[3.0, 4.0]
```

TensorFlow

Placeholder

나중에 입력 받을 데이터를 고려하여 모델을 만들 때**: placeholder !**

Placeholders

A TF program often has 2 phases:

1. Assemble a graph
2. Use a session to execute operations in the graph.

⇒ Can assemble the graph first without knowing the values needed for computation

Analogy:
Can define the function f(x, y) = x*2 + y without knowing value of x or y. x, y are placeholders for the actual values.

Reference: CS20SI – lecture 2. Operations (http://web.stanford.edu/class/cs20si/syllabus.html)

Placeholder

- tf.placeholder(dtype, shape, name)
- A placeholder is a promise to provide an actual value later.

```
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, {a: 3, b: 4.5}))
print(sess.run(adder_node, {a: [1, 3], b: [2, 4]}))
```

Feeding

- 앞서 정의한 placeholder a와 b에 실제 데이터 (actual data) 입력
- Key와 value로 구성된 dictionary 형태의 자료형 사용

print 결과:

```
7.5
[ 3.  7.]
```

(TensorFlow는 자체적으로 효과적인 벡터 연산을 지원)

Variable

- 어떤 알고리즘을 학습시키기 위해서는 학습의 반복에 따라 값이 변하는 변수(**variable**)이 필요
- Variable tensor는 session에서 실행(run)하기 전 초기화(initialization)이 필요

```
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W * x + b
```

Initialization

- Initialization에 대한 선언! 아직 해당 값 혹은 method로 초기화 된 값이 들어간 것은 아님!

```
init = tf.global_variables_initializer()
sess.run(init)
```

Variable initialization

- 해당 session에 있는 모든 variable tensor를 각각의 방법으로 initialization
- 여기서 값이 들어감! (미리 session을 열어놨다고 가정)

```
print(sess.run(linear_model, {x: [1, 2, 3, 4]}))
```

print 결과:

```
[ 0.          0.30000001  0.60000002  0.90000004]
```

TensorFlow

Variable

Variable의 값을 확인하고 싶을 때:

Eval() a variable

```
# W is a random 700 x 100 variable object
W = tf.Variable(tf.truncated_normal([700, 10]))
with tf.Session() as sess:
      sess.run(W.initializer)
      print W

>> Tensor("Variable/read:0", shape=(700, 10), dtype=float32)
```

Reference: CS20SI – lecture 2. Operations (http://web.stanford.edu/class/cs20si/syllabus.html)

TensorFlow

Variable

Variable의 값을 확인하고 싶을 때: ***variable*.eval()**

Eval() a variable

```
# W is a random 700 x 100 variable object
W = tf.Variable(tf.truncated_normal([700, 10]))
with tf.Session() as sess:
      sess.run(W.initializer)
      print W.eval()
```

```
>> [[-0.76781619 -0.67020458  1.15333688 ..., -0.98434633 -1.25692499
  -0.90904623]
 [-0.36763489 -0.65037876 -1.52936983 ...,  0.19320194 -0.38379928
   0.44387451]
 [ 0.12510735 -0.82649058  0.4321366  ..., -0.3816964   0.70466036
   1.33211911]
 ...,
 [ 0.9203397  -0.99590844  0.76853162 ..., -0.74290705  0.37568584
   0.64072722]
 [-0.12753558  0.52571583  1.03265858 ...,  0.59978199 -0.91293705
  -0.02646019]
 [ 0.19076447 -0.62968266 -1.97970271 ..., -1.48389161  0.68170643
   1.46369624]]
```

Reference: CS20SI – lecture 2. Operations (http://web.stanford.edu/class/cs20si/syllabus.html)

TensorFlow

Assign

초기화한 **variable**의 값을 바꾸고 싶을때**!**

▣ tf.assign()

```
fixW = tf.assign(W, [-1.])
fixb = tf.assign(b, [1.])
sess.run([fixW, fixb])
print(sess.run(loss, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]}))
```

▲ print 결과:

```
0.0
```

tf.Variable.assign()

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
    sess.run(W.initializer)
    print W.eval() # >> 10

--------

W = tf.Variable(10)
assign_op = W.assign(100)
with tf.Session() as sess:
    sess.run(assign_op)
    print W.eval() # >> 100
```

- **tf.assign()** 도 일종의 **operation**
- **Session** 안에서 **run** 하지 않으면 실행되지 않음**!**

- **tf.assign()** 한 **variable**은 **initialization**을 할 필요가 없음 **(tf.assign** 자체가 **initialization** 기능 수행**)**

Variable의 값 획득**!**

Reference: CS20SI – lecture 2. Operations (http://web.stanford.edu/class/cs20si/syllabus.html)

Loss function의 정의

- Model의 값이 우리가 원하는 값 y와 얼마나 가까운지 measure
- Example: sum of squared error

```
y = tf.placeholder(tf.float32)
squared_deltas = tf.square(linear_model - y)
loss = tf.reduce_sum(squared_deltas)
print(sess.run(loss, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]}))
```

→ Loss function의 정의

- print 결과:

```
23.66
```

TensorFlow

Training

▲ **Optimizer**의 정의

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

Optimizer의 정의

Training operation의 정의

```
sess.run(init) # reset values to incorrect defaults.
for i in range(1000):
  sess.run(train, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]})
print(sess.run([W, b]))
```

Training 실행**!**

Training에 필요한 **actual data feeding**

▲ print 결과: (학습으로 구한 W와 b의 결과)

```
[array([-0.9999969], dtype=float32), array([ 0.99999082],
 dtype=float32)]
```

TensorFlow

Multiple sessions

여러 개의 **session**을 동시에 운용 가능하며, 각 **session** 안에서 **variable**은 독립적으로 관리

Each session maintains its own copy of variable

```
W = tf.Variable(10)

sess1 = tf.Session()
sess2 = tf.Session()

sess1.run(W.initializer)
sess2.run(W.initializer)

print sess1.run(W.assign_add(10)) # >> 20
print sess2.run(W.assign_sub(2)) # >> 8

print sess1.run(W.assign_add(100)) # >> 120
print sess2.run(W.assign_sub(50)) # >> -42

sess1.close()
sess2.close()
```

기존 variable 값에서 10을 더한 값을 assign

기존 variable 값에서 2를 뺀 값을 assign

Reference: CS20SI – lecture 2. Operations (http://web.stanford.edu/class/cs20si/syllabus.html)

TensorFlow

TensorFlow implementation for linear regression model

1. **Data preparation**
2. **Create variables for weight and bias**
3. **Create place holder**
4. **Construct a model to predict Y**
5. **Define loss function and optimizer**
6. **Initialize variables**
7. **Train the model**
8. **Output the value of w and b**

```python
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W * x + b
y = tf.placeholder(tf.float32)

# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

# training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
  sess.run(train, {x: x_train, y: y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x: x_train, y: y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```

TensorFlow

Problem1

In [12]:
```
import tensorflow as tf
a = tf.constant(1, name="c1")
b = tf.constant(2, name="c2")
c = a + b
# 채워야 하는 곳
```

```
detailed information of tensor a (not the value) :
Tensor("c1_5:0", shape=(), dtype=int32)
value of tensor c (not the detailed information) :
3
```

- 주석 부분을 채워 아래와 같이 출력
- Tensor(….) 부분은 내부 값이 위와 달라도 상관없음

TensorFlow

Problem2

```
In [17]: import tensorflow as tf
         # 채워야 하는 곳
         c = a + b
         print("Result of add : ")
         print(sess.run(c, feed_dict={b:[-5, 0.5, 7]}))

         Result of add :
         [-2.5  3.   9.5]
```

click to expand output; double click to hide output

- 주석 부분을 채워 아래와 같이 출력
- 주어진 코드 부분은 수정하지 않는것을 권장

- X_train : [5.7, 10.6, 22.4, 24.7]
- Y_train : [2.545, 6.71, 16.74, 18.695]
  일 때 Y = W*X + b 를 가장 잘 근사하는 W, b를 tensorflow를 이용하여 찾으시오
  (slide 19 참고)

Reference

- 모두의 딥러닝 by 김성훈
  : http://hunkim.github.io/ml/

- TensorFlow 공식 홈페이지
  : http://tensorflow.org/

- Tensorflow 한글 정리자료
  : https://tensorflowkorea.gitbooks.io/tensorflow-kr/content/