Python 수업 3회차

Tensorflow 설치 및 기본형 배우기

1. 지난시간

Tensorflow 설치

1. Tensorflow를 설치해보자

- 다음 링크를 참고함
  - 함
- 1. Anaconda 설치하기
  - https://www.continuum.io/downloads
- 2. Anaconda Prompt를 이용해 tensorflow 설치
  - pip install tensorflow
  - 혹은 conda install tensorflow
- 3. jupyter notebook 실행
- 4. 이후 텐서플로우를 사용하고 싶을 때마다 conda 환경을 활성화하고 실행하면 됩니다.

Anaconda 란?

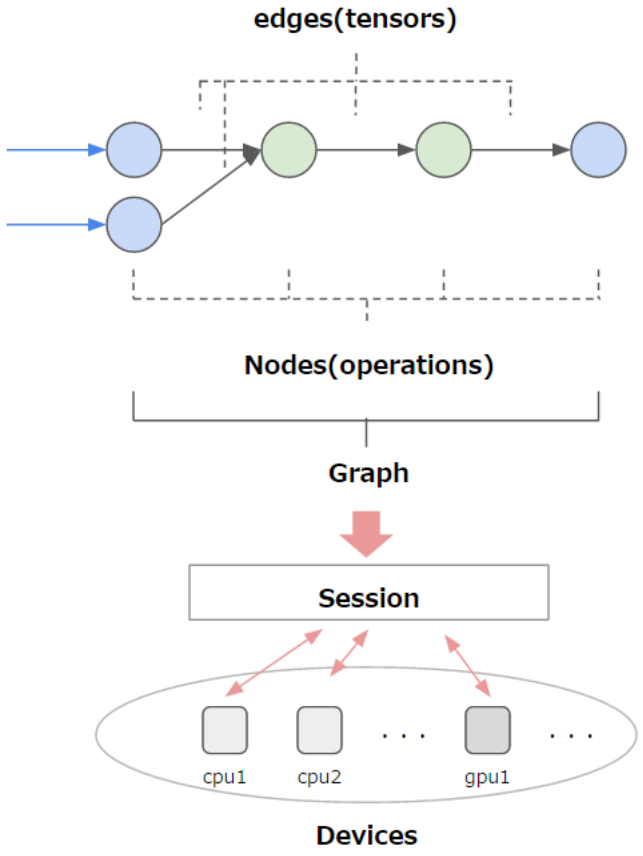
- Anaconda 는 여러 수학, 과학 패키지를 기본적으로 포함하고 있는 파이썬 배포판입니다. Anaconda 는 "conda" 로 불리는 패키지 매니저를 사용하여 Virtualenv 와 유사한 환경 시스템을 제공합니다.
- Virtualenv 처럼 conda 환경은 각기 다른 파이썬 프로젝트에서 필요한 패키지들의 버전이 충돌되지 않도록 다른 공간에서 운영합니다. 텐서플로우를 Anaconda 환경으로 설치하면 기존 파이썬 패키지들을 덮어쓰지 않게 됩니다.

오늘의 목표 – 아래의 코드 이해하기.

```
state = tf.Variable(0, name="counter")
one = tf.constant(1)
new_value = tf.add(state, one)
update = tf.assign(state, new_value)
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    print(sess.run(state))
    for _ in range(3):
        sess.run(update)
        print(sess.run(state))
```

기본 사용법과 프로그램 단계

기본 사용법 - 동작이해



- TensorFlow는 graph로 연산을 나타내는 프로그래밍 시스템입니다.
- 노드는 *작업(op)*이라고 부릅니다.
  - 작업(op)은 0개 혹은 그 이상의 Tensor를 가질 수 있고 연산도 수행하며 0개 혹은 그 이상의 Tensor를 만들어 내기도 합니다.
- Tensorflow에서 Tensor는 정형화된 다차원 배열(a typed multi-dimensional array)입니다.
  - Ex) 이미지는 4차원 배열([batch, 가로, 세로, channels(RGB)])로 나타낼 수 있습니다.
- 연산을 하려면 graph가 Session 상에 실행되어야 합니다.
- Session은 graph의 작업(op)(노드)을 CPU나 GPU같은 Device에 배정하고 실행을 위한 메서드들을 제공합니다.
  - 이런 메서드들은 작업(op)을 실행해서 tensor를 만들어 냅니다.
    - 파이썬에서 tensor는 numpy ndarray 형식.

기본 사용법 - 동작이해

- TensorFlow는 graph로 연산을 나타내는 프로그래밍 시스템입니다.
  - graph는 점(노드)과 선(엣지)으로 이뤄진 수학적인 구조를 의미합니다.
- 노드는 *작업(op)*이라고 부릅니다.
  - 작업(operation 혹은 op)은 tensor(데이터)로 연산해 tensor(데이터)를 만들어냅니다.
- 엣지(데이터)는 tensor로 표현합니다.
  - Tensor는 numpy ndarray 형식
- graph는 Session내에서 실행됩니다.

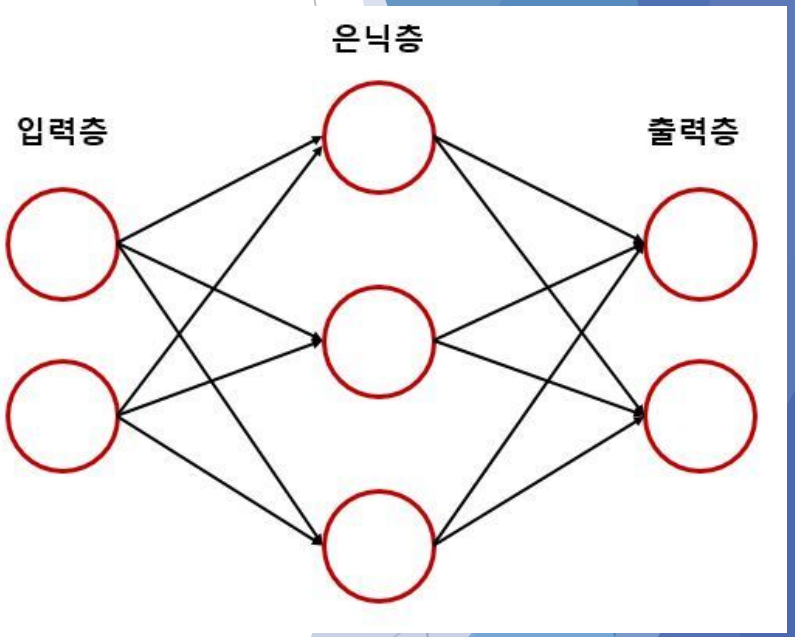
프로그램의 단계

- 프로그램은 2단계로 나뉨
- 1. 구성 단계(construction phase)
  - 보통 graph를 조립함
- 2. 실행 단계(execution phase)
  - session을 이용해 graph의 작업(op)을 실행시키는 단계

  - Ex) 뉴럴 네트워크를 표현하고 학습하는 프로그램
    - 구성 단계 : graph를 만들기
    - 실행 단계 : graph의 훈련용 작업들(set of training ops)을 반복해서 실행

구성 단계

- 상수(constant)같이 아무 입력값이 필요없는 작업(편향 등)을 정의하는 것에서부터 시작
  - 이 op을 연산이 필요한 다른 op들에게 입력값으로 제공
- 작업 생성 함수(op constructor)
  - 만들어진 작업(op)들의 결과값을 반환합니다. 이 결과값은 다른 작업(op) 생성시 함수의 입력값으로 이용할 수 있다.
- Ex) 작업 -> 결과값(새 입력값) -> 작업 -> 결과값(새 입력값) -> ...
- 작업 생성 함수로 노드를 추가할 수 있는 *default graph*라는 것이 있습니다. default graph는 다양하게 이용하기 좋습니다.



1x2 행렬, 2X1행렬 만드는 상수(constant) op를 만들어 봅시다.

- data = tf.constant(<행렬 or 스칼라값>) 이용

```
import tensorflow as tf
matrix1 = tf.constant([[3., 3.]])
matrix2 = tf.constant([[2.],[2.]])
product = tf.matmul(matrix1, matrix2)
# 이 op의 결과값인 'product'는 행렬곱(Matrix multiply)의 결과를 의미합니다.
 # 이 op들은 default graph에 노드로 들어갈 것입니다.
```

실행단계 추가

 # session에서 (default) graph 실행

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

sess = tf.Session()

result = sess.run(product)

print(result)

sess.close()

작업의 결과물은 numpy `ndarray` 오브젝트인 result' 값

Session을 효율적으로 써보자.

- 시스템 자원을 더 쉽게 관리하려면 with 구문을 쓰면 됨.
- 각 Session에 컨텍스트 매니저가 있어서 'with' 구문 블락의 끝에서 자동으로 'close()'가 호출됨.

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

변수(Variable)

변수란

- 데이터를 담는 그릇
- 이때 데이터란? -> tensor
- 즉, 모델을 학습 시킬 때, 매개 변수(parameter) 업데이트와 유지를 위해 변수(Variables)를 사용
- 반드시 명시적으로 초기화해야 함
- 디스크에 저장 가능
  - 학습 중 혹은 학습 후에도.
- 나중에 복원도 가능
  - 저장된 값들을 모델 실행이나 분석을 위해.
- 다음을 이용
  - tf.Variable 클래스. tf.train.Saver 클래스.

변수 생성 – Variable 클래스

- 생성자의 초기값으로 Tensor 를 전달받음
- variable 오퍼레이션은 그 변수의 값을 가지고 있습니다.
- tf.assign 오퍼레이션을 이용해서 변수의 초기값을 설정할 수 있습니다.

- TensorFlow는 다양한 초기화 명령어(op)를 제공.
  - 상수(constants) or 임의(random)의 값 등
- 다음과 같이 생성함
  - data = tf.Variable(<initial-value>, name=<optional-name>)

백문이 불여일견 – 변수 생성

```
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35), name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")

#가중치와 편향 데이터.
#참고 - data = tf.Variable(<initial-value>, name=<optional-name>)
```

상수값 텐서

TensorFlow는 상수를 생성할 수 있는 몇가지 연산을 제공합니다.

`tf.zeros(shape, dtype=tf.float32, name=None)`

모든 원소의 값이 0인 텐서를 생성합니다.

이 연산은 모든 원소의 값이 0이고, `shape` shape을 가진 `dtype` 타입의 텐서를 반환합니다.

예시:

```
tf.zeros([3, 4], int32) ==> [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

인자:

- **`shape`** : 정수 리스트 또는 `int32` 타입의 1-D(1-Dimension) `Tensor` .
- **`dtype`** : 반환되는 `Tensor` 의 원소 타입.
- **`name`** : 연산의 명칭 (선택사항).

반환값:

모든 원소의 값이 0인 `Tensor` .

- biases = tf.Variable(tf.zeros([200]), name="biases")
- 200 X 1의 행렬, 모든 값 0, name은 biases

`tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)`

정규분포로부터의 난수값을 반환합니다.

인자:

- `shape` : 정수값의 1-D 텐서 또는 파이썬 배열. 반환값 텐서의 shape입니다.
- `mean` : 0-D 텐서 또는 `dtype` 타입의 파이썬 값. 정규분포의 평균값.
- `stddev` : 0-D 텐서 또는 `dtype` 타입의 파이썬 값. 정규분포의 표준 편차.
- `dtype` : 반환값의 타입.
- `seed` : 파이썬 정수. 분포의 난수 시드값을 생성하는데에 사용됩니다. 동작 방식은 `set_random_seed` 를 보십시오.
- `name` : 연산의 명칭 (선택사항).

반환값:

정규 난수값들로 채워진 shape으로 정해진 텐서.

- weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35), name="weights")
- 난수, 784 X 200 행렬의 shape, 표준편차 0.35, 평균0, 반환값타입 float32, name은 weights

변수 초기화

- 변수 초기화
  - 모델의 다른 연산을 실행하기 전에 반드시 명시적으로 실행해야 함
- 가장 쉬운 방법
  - 모든 변수를 초기화 하는 연산을 모델 사용 전에 실행하는 것.
- 변수 초기화를 위한 작업(op)을 추가
  - tf.global_variables_initializer()를 사용
  - 모델을 모두 만들고 세션에 올린 후 이 작업을 실행할 수 있음.

백문이 불여일견 - 변수초기화

```
# 두 변수를 생성
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35), name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")
...
# 변수 초기화 오퍼레이션을 초기화
init_op = tf.global_variables_initializer()
# 나중에 모델을 실행할때
with tf.Session() as sess:
    # 초기화 오퍼레이션을 실행
    sess.run(init_op)
    ...
    # 모델 사용
    ...
```

다른 변수값을 참조하여 초기화 하기

- **initialized_value()** 속성을 사용

랜덤 값으로 새로운 변수 초기화

weights와 같은 값으로 다른 변수 초기화

weights의 2배 값으로 다른 변수 초기화

```
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35), name="weights")
w2 = tf.Variable(weights.initialized_value(), name="w2")
w_twice = tf.Variable(weights.initialized_value() * 2.0, name="w_twice")
```

tf.global_variables_initializer()연산으로 모든 변수를 초기화 해야 하는 경우에는 종속적으로 바뀌므로 고려해야 함.

변수 Saver(저장)

- tf.train.Saver() 로 Saver 를 생성하여 모든 변수를 관리

```
# 몇개의 변수를 생성
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
# 변수 초기화를 위한 오퍼레이션
init_op = tf.global_variables_initializer()
# 모든 변수의 저장과 복구를 위한 오퍼레이션 추가
saver = tf.train.Saver()
# 모델 실행, 변수 초기화, 몇 가지의 작업 실행, 디스크에 변수 저장
with tf.Session() as sess:
    sess.run(init_op)
    # 모델을 사용하여 작업
    ...
    # 디스크에 변수를 저장
    save_path = saver.save(sess, "/tmp/model.ckpt")
    print("Model saved in file: %s" % save_path)
```

선택적 변수 저장 및 복구

- 만약 tf.train.Saver()에 아무런 인자도 전달하지 않는다면, saver는 그래프 안의 모든 변수를 다룹니다.
- tf.train.Saver() 오브젝트에 파이썬 dictionary 타입의 데이터를 전달해서 저장할 이름과 변수를 쉽게 지정할 수 있습니다.
  - dictionary의 키는 사용할 이름이며, 값(value)은 관리할 변수이다.

```
# 몇개의 변수 생성
v2 = tf.Variable(..., name="v2")
# "my_v2"라는 이름을 이용하여 'v2'를 저장하고 복구 하는 오퍼레이션 추가
saver = tf.train.Saver({"my_v2": v2})
```

최종 – 백문이 불여일견

```
state = tf.Variable(0, name="counter")
one = tf.constant(1)
new_value = tf.add(state, one)
update = tf.assign(state, new_value)
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    print(sess.run(state))
    for _ in range(3):
        sess.run(update)
        print(sess.run(state))
```