텐서플로우를 활용한 머신러닝

인공지능과 머신러닝 텐서플로우란?

기본개념

- 텐서플로우 특징
  - 데이터플로우 그래프(Data flow graph) 방식 사용
  - 수치 연산을 하는 오픈 소스 소프트웨어 라이브러리
  - 수학계산과 데이터 흐름을 노드(node)와 엣지(Edge)를 사용하여 방향 그래프(Directed Graph)로 표현
  - 노드(node)는 수학적 계산, 데이터 입/출력, 데이터 읽기/저장 등을 나타냄
  - 엣지(edge)는 노드 사이를 이동하는 다차원 배열(텐서, tensor)을 나타냄

❏ **텐서플로우 특징**
- ❏ **데이터플로우 그래프를 통한 풍부한 표현력**
- ❏ **코드 수정 없이 CPU/GPU 모드로 동작**
- ❏ **아이디어 테스트에서 서비스 단계까지 이용 가능**
- ❏ **계산 구조와 목표 함수만 정의하면 자동으로 미분 계산을 처리**
- ❏ **Python/C++을 지원하며, SWIG를 통해 다양한 언어 지원 가능**

기본개념

- 기본용어
  - 텐서(tensor) : 과학과 공학 등 다양한 분야에서 쓰이던 개념
    - 수학과 물리학에서 선형 관계를 나타내는 기하적 대상
    - 두 개 이상의 독립적인 방향을 동시 표현할 때 사용
    - 데이터를 표현하는 방식
    - Tensorflow 내부적으로 모든 데이터는 텐서를 통해 표현됨
    - 그래프 내의 오퍼레이션 간에는 텐서만 전달됨
  - 오퍼레이션(Operation) : 그래프상의 노드는 오퍼레이션으로 불림
    - 하나 이상의 텐서를 받을 수 있음
    - 계산을 수행하고 결과를 하나 이상의 텐서로 반환할 수 있음
  - 세션(Session) : 오퍼레이션의 실행 환경을 캡슐화
    - 그래프를 실행하기 위해서는 세션 객체가 필요
  - 변수(Variables) : 그래프의 실행 시 파라미터를 저장하고 갱신
    - 메모리상에서 텐서를 저장하는 버퍼 역할

텐서자료형

- 랭크(Rank)
  - 텐서의 차원은 랭크로 표현됨
  - Tensor rank(order, degree, n-dimensions)는 텐서의 차원 수

    t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

  - 텐서 t는 랭크가 2인 텐서 행렬
  - 크기만을 갖는 scalar는 랭크가 0, 크기와 방향을 갖는 벡터는 랭크가 1이고, 행렬이 랭크가 2

Rank	Math entity	Example
0	Scalar : 크기만 존재	s = 134
1	Vector : 크기, 방향 존재	v = [1.1, 2.2, 3.3, 4.4]
2	Matrix : 숫자 테이블	m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
3	3-Tensor : 3차원 숫자 큐브	t = [[[1], [3], [5]], [7], [9], [11], [13], [15]]]
n	n-Tensor	...

텐서자료형

- 셰이프(Shape)
  - 텐서는 셰이프라는 각 차원의 크기 값을 튜플의 형태로 표현할 수 있음

Shape	Dimension number	Rank	Example
[]	0-D	0	0-D tensor. A scalar
[D0]	1-D	1	1-D tensor with shape [5].
[D0, D1]	2-D	2	2-D tensor with shape [3, 4].
[D0, D1, D2]	3-D	3	3-D tensor with shape [1, 4, 3].
[D0, D1, … , Dn-1]	n-D	n	Tensor with shape [D0, D1, … , Dn-1]

텐서자료형

- 데이터 타입(Data types)
  - 텐서는 차원 외 데이터 타입을 가짐

Data type	Python type	Description	Data type	Python type	Description
DT_FLOAT	tf.float32	32비트 부동소수	DT_STRING	tf.string	가변 길이 바이트 배열, Tensor의 각 원소는 바이트 배열
DT_DOUBLE	tf.float64	64비트 부동소수	DT_BOOL	tf.bool	불리언
DT_INT8	tf.int8	8비트 부호 있는 정수	DT_COMPLEX64	tf.complex64	2개의 32비트 부동소수로 만든 복소수 : 실수부 + 허수부
DT_INT16	tf.int16	16비트 부호 있는 정수	DT_COMPLEX128	tf.complex128	2개의 64비트 부동소수로 만든 복소수 : 실수부 + 허수부
DT_INT32	tf.int32	32비트 부호 있는 정수	DT_QINT8	tf.qint8	8비트 부호 있는 정수로 quantized Ops에서 사용
DT_INT64	tf.int64	64비트 부호 있는 정수	DT_QINT32	tf.qint32	32비트 부호 있는 정수로 quantized Ops에서 사용
DT_UINT8	tf.uint8	8비트 부호 없는 정수	DT_QUINT8	tf.quint8	8비트 부호 없는 정수로 quantized Ops에서 사용

상수형(Constant)

- 상수형(Constant)
  - 말 그대로 상수를 저장하는 데이터형
  - tf.constant()를 사용하여 표현

상수형 선언 : tf.constant(value, dtype=None, shape=None, name='Const', verity_shape=False)

반환값 : 상수형 텐서

- value : 상수 값
- dtype : 상수의 데이터 형(tf.float32와 같이 실수, 정수 등의 데이터 타입 정의
- shape : 행렬의 차원 정의(shape=[3,3]인 경우, 3X3 행렬 저장)
- name : 상수의 이름 정의

예시)

```
a = tf.constant([10], dtype = tf.float32)
b = tf.constant(100)
```

상수형(Constant)

- 상수형(Constant)
  - x값을 그대로 출력하면 type이 int32, shape가 값이 없는 real number(scalar)로 구성이 된 Tensor를 출력
  - 모든 데이터는 Tensor로 생성이 되어 이후에 발생하는 모든 연산에서 수행이 됨
  - session 객체를 사용하여 실행
  - session 안에서만 실제적인 연산 및 로직이 수행됨

```
import tensorflow as tf

x = tf.constant(3)
print(x)

출력 > Tensor(“Const:0”, shape=(), dtype=int32)
```

```
import tensorflow as tf

x = tf.constant(3)
sess = tf.Session()
result = sess.run(x)
print(x)

출력 > 3
```

변수(Variable)

- 변수(Variable)
  - 변수는 매개변수(parameter) 업데이트와 유지를 위해 사용
  - tf.Variable()를 사용하여 표현
  - 메모리에 텐서를 저장하는 버퍼 역할
  - 변수는 반드시 명시적으로 초기화를 해야함
  - 학습 중 혹은 학습 후 디스크에 저장
  - 저장된 값들을 필요에 따라 다시 복원할 수 있음

변수(Variable)

- 변수(Variable)
  - 변수를 생성할 때 Variable() 생성자의 초기값으로 Tensor를 전달 받음
  - Tensorflow는 상수 또는 임의값으로 초기화하는 다양한 명령어 제공
  - 모든 명령어는 Tensor들의 형태를 지정하여야하며, 자동적으로 변수의 형태가 됨

변수 선언 : tf.Variable(<initial-value>, name = <optional-name>

예시)

```
Var_1 = tf.Variable(3, name = ‘var_1’)
Var_2 = tf.Variable(10)
```

변수(Variable)

- 변수(Variable)
  - 예상했던 결과와 다르게 에러코드 발생
  - 변수는 텐서가 아닌 하나의 객체가 됨
  - 선언한 각 변수들은 변수 클래스의 인스턴스가 생성
  - Tensorflow에서 변수형은 그래프를 실행하기 전에 초기화 작업을 진행해야 함
  - 초기화 작업을 하지 않으면 변수를 사용할 수 없으며, 초기화 전 변수에 값이 지정되어 있지 않음

```
import tensorflow as tf
var_1 = tf.Variable(3)
var_2 = tf.Variable(10)
result_sum = var_1 + var_2
sess = tf.Session()
print(sess.run(result_sum))

출력 > 에러코드 발생
```

```
import tensorflow as tf
var_1 = tf.Variable(3)
var_2 = tf.Variable(10)
result_sum = var_1 + var_2
init = tf.global_variables_initializer()    # 초기화 함수 추가
sess = tf.Session()
sess.run(init)                              # 초기화된 결과를 세센에 전달
print(sess.run(result_sum))

출력 > 5
```

플레이스홀더(Placeholder)

- 플레이스홀더(Placeholder)
  - 플레이스홀더는 선언과 동시에 초기화하는 것이 아니라, 선언 후 그 다음 값을 전달하기 때문에 반드시 실행 시 데이터가 제공되어야 함
  - 학습용 데이터를 담는 영역이라고 생각하면 됨

```
플레이스홀더 선언 : tf.placeholder(dtype, shape=None, name=None)

예시)
p_holder1 = tf.placeholder(dtype=tf.float32)
p_holder2 = tf.placeholder(dtype=tf.float32)
```

  - dtype은 데이터 타입을 의미하며 반드시 선언해야함
  - shape는 입력 데이터의 형태
  - name는 해당 오퍼레이션 이름을 지정

플레이스홀더(Placeholder)

- 플레이스홀더(Placeholder)
  - 기본 예제

```
import tensorflow as tf

var_1 = 15
var_2 = 8

p_holder1 = tf.placeholder(dtype=tf.float32)
p_holder2 = tf.placeholder(dtype=tf.float32)
p_holder_sum = p_holder1 + p_holder2

sess = tf.Session()
result = sess.run(p_holder_sum, feed_dict = {p_holder1: var_1, p_holder2: var_2})
print(result)

출력 > 23.0
```

- 플레이스홀더 두 개 선언
  - p_holder1, p_holder2
- 학습용 데이터
  - var_1, var_2
- 학습용 데이터를 계산할 그래프
  - p_holer_sum
- 학습용 데이터를 넣기 위해 feed_dict 변수 사용
- 세션을 이용하여 그래프 실행
- 최종결과가 result에 반환되어 출력

플레이스홀더(Placeholder)

- 플레이스홀더(Placeholder)
  - 기본 예제(배열 사용)

```
import tensorflow as tf

A = [1, 3, 5, 7, 9]
B = [2, 4, 6, 8, 10]

ph_A = tf.placeholder(dtype=tf.float32)
ph_B = tf.placeholder(dtype=tf.float32)
result_sum = ph_A + ph_B

sess = tf.Session()
result = sess.run(result_sum, feed_dict = {ph_A:A, ph_B:B})
print(result)

출력 > [3, 7, 11, 15, 19]
```

- 배열 형태로 학습용 데이터 선언
- 플레이스홀더 변수 선언
  - ph_A, ph_B
- 학습용 데이터 선언
  - result_sum
- feed_dict 변수 사용
- 세션을 이용하여 그래프 실행
- 최종결과가 result로 반환되어 출력