Deep Learning 기초 실습: Tensorflow & Keras

Tensorflow 기초

1. Tensorflow 소개

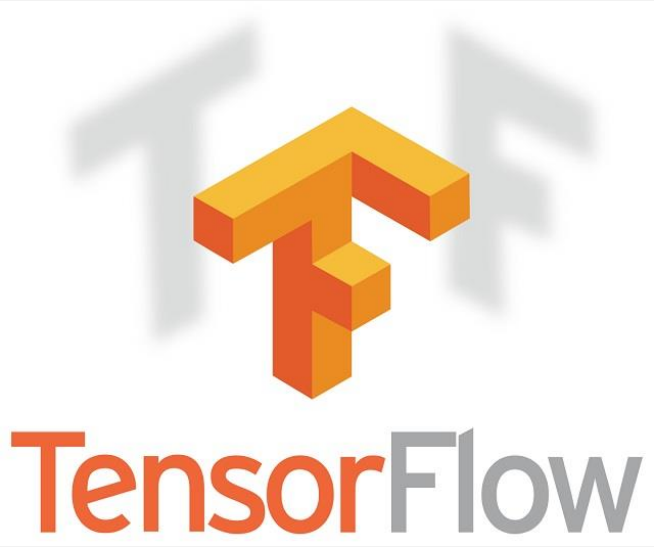
목차

1. Tensorflow 소개
2. Tensorflow 구성 및 동작
3. 예제 : Linear Regression
4. 예제 : MLP
5. TensorBoard

1. Tensorflow 소개

- Tensorflow는?
  - ✓구글에서 개발하여 공개한 딥러닝/머신러닝을 위한 오픈소스 라이브러리

- 지원언어
  - ✓Python, C++, JAVA, Go (Python 환경에 최적화가 되어 있음)



1. Tensorflow 소개

- 유사 라이브러리
  - ✓토치(Torch) : 페이스북에서 주도적으로 개발한 Lua 언어 기반의 라이브러리
  - ✓파이토치(PyTorch) : 토치의 파이썬 버전
  - ✓CNTK(CogNitive ToolKit) : MS에서 공개

1. Tensorflow 소개

- Tensorflow의 장점
  - ✓전세계적으로 활발한 커뮤니티
  - ✓TensorBoard를 이용한 편리한 시각화
  - ✓단일 데스크톱, 대량의 서버 클러스터, 모바일 등의 광범위한 이식성
  - ✓Keras, TF-Slim, Sonnet 등 다양한 추상화 라이브러리와 혼용해서 사용 가능
  - ✓구글, 딥마인드, 우버, 스냅챗 등 글로벌 기업들이 활발히 사용 중

1. Tensorflow 소개

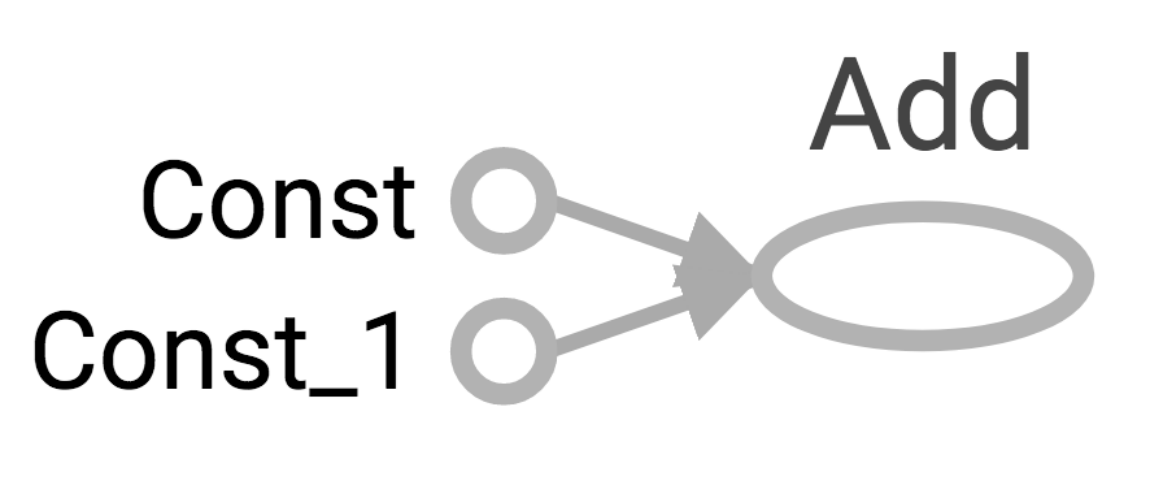
- Tensorflow 응용 분야
  - ✓컴퓨터 비전 : 이미지 분류, 물체 검출 등 (CNN)
  - ✓자연어처리
  - ✓음성인식
  - ✓게임
  - ✓생성모델 (GAN)

2. Tensorflow 구성 및 동작

Tensorflow 구성 및 동작

- Tensorflow 동작
  - ✓Tensor를 흘려보내면서(flow) 데이터를 처리

- Tensorflow 기본구조
  - ✓Graph : Node와 Edge의 조합
  - ✓Node : Operator, Variable, Constant
  - ✓Edge : 노드간의 연결



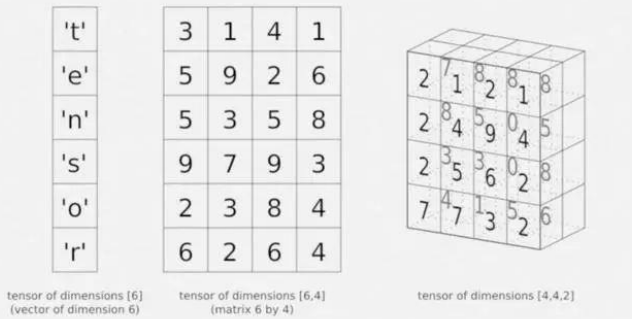
Tensorflow 구성 및 동작

- Tensor란?
  - ✓임의의 차원을 갖는 배열

- Tensor Rank
  - ✓Tensor의 차원

텐서(Tensor)의 Rank

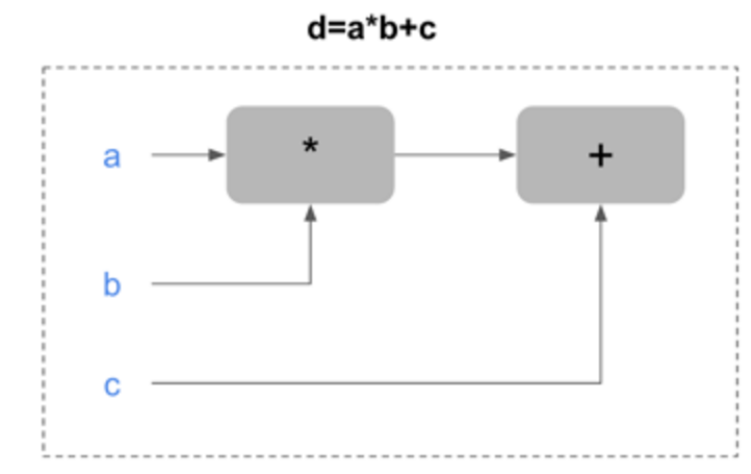
Rank	Math Entity
0	Scalar(magnitude only)
1	Vector(magnitude and direction)
2	Matrix(table of numbers)
3	3-**Tensor**(cube of numbers)



Tensorflow 구성 및 동작

- Tensorflow 실행 과정
  - ✓그래프 생성
  - ✓그래프 실행

- 그래프 구현 예시




```
# 그래프 생성
a = tf.constant(3.0)
b = tf.constant(4.0)
c = tf.constant(5.0)
d = a * b + c
print (d)

# 그래프 실행
sess = tf.Session()
result = sess.run(d)
print (result)
```

출력 : Tensor("add:0", shape=(), dtype=float32)

출력 : 17.0

Tensorflow 구성 및 동작

- 데이터 자료구조
  - ① Constant (상수형)
    - ✓ 변하지 않는 값
  - ② Placeholder
    - ✓ 데이터를 담는 그릇
    - ✓ 학습용 데이터를 위한 타입 (input feeding)
  - ③ Variable (변수형)
    - ✓ 학습을 통해 구해야 하는 값
    - ✓ Weight 등의 파라미터를 위한 타입

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_UINT16	tf.uint16	16 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8 bits signed integer used in quantized Ops.
DT_QINT32	tf.qint32	32 bits signed integer used in quantized Ops.
DT_QUINT8	tf.quint8	8 bits unsigned integer used in quantized Ops.

Tensorflow 구성 및 동작

- Constant (상수형)
  - ✓API 및 파라미터

```
tf.constant(
    value,
    dtype = None,
    shape = None,
    name = 'Const'
    verify_shape=False
)
```

파라미터	역할
value	상수값이며 직접 지정하고나 shape 형태로 채울 값을 지정
dtype	데이터 타입 (예 : tf.float32, tf.int32, tf.bool)
shape	상수 데이터의 형태
name	텐서의 이름 (Optional)
verify_shape	shape를 체크

  - ✓API 사용 예시

```
# Constant 1-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3, 4, 5, 6, 7]) => [1 2 3 4 5 6 7]

# Constant 2-D tensor populated with scalar value -1.
tensor = tf.constant(-1.0, shape=[2, 3]) => [[-1. -1. -1.]
                                             [-1. -1. -1.]]
```

Tensorflow 구성 및 동작

- Placeholder (플레이스 홀더)
  - ✓API 및 파라미터

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

파라미터	역할
dtype	Feed할 데이터 타입
shape	Feed할 데이터의 형태 (None이면 임의의 차원)
name	텐서의 이름 (Optional)

  - ✓API 사용 예시

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
```

Tensorflow 구성 및 동작

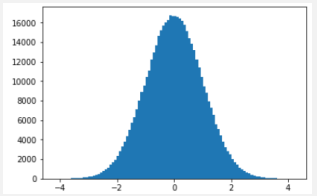
- Variable (변수형)
  - ✓API 및 파라미터

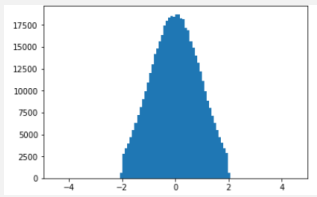
```
tf.Variable(
    initial_value = None,
    trainable = True,
    name = None,
)
```

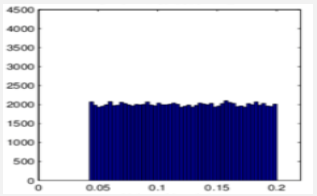
파라미터	역할
initial_value	변수의 초기값이며 변수의 shape를 포함한 상태로 지정
trainable	트레이닝 가능 여부 (False면 파라미터가 업데이트 안됨)
name	텐서의 이름 (Optional)

  - ✓API 사용 예시

```
W = tf.Variable(tf.random_normal(shape=[1], name='w'))
b = tf.Variable(tf.random_normal(shape=[1], name='b'))
```





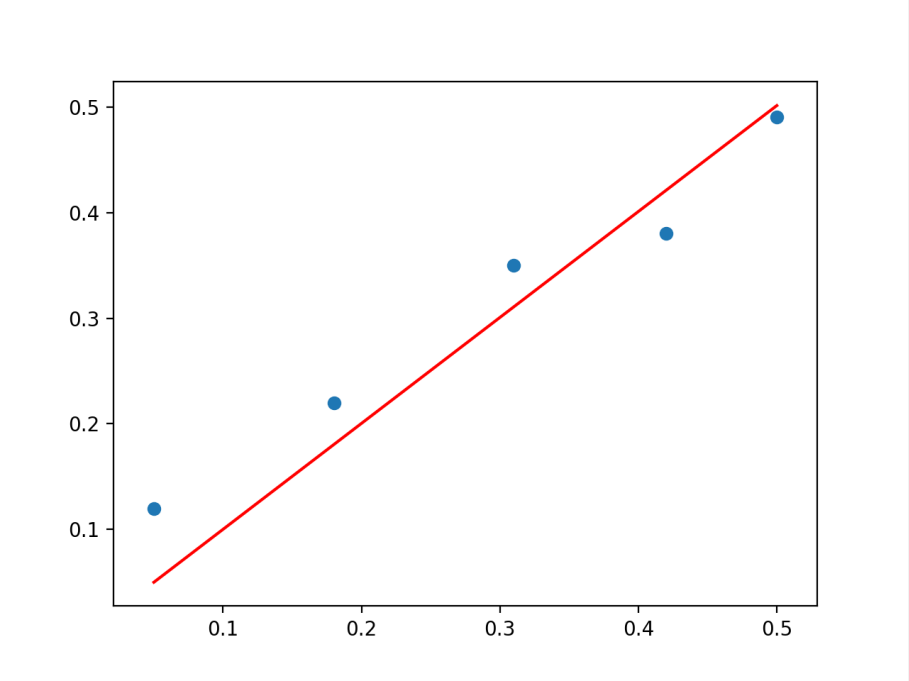


  - ✓초기화 종류

종류	설명
tf.random_normal	가우시안 분포(정규 분포)에서 임의의 값을 추출
tf.truncated_normal	Truncated normal 분표(끝 부분이 잘린 정규분포 모양)에서 임의의 값 추출
tf.random_uniform	균등 분포에서 임의의 값을 추출
tf.constant	특정한 상수값으로 지정한 행렬로 채움
tf.zeros	모두 0으로 채움
tf.ones	모두 1로 채움

3. 예제: Linear Regression

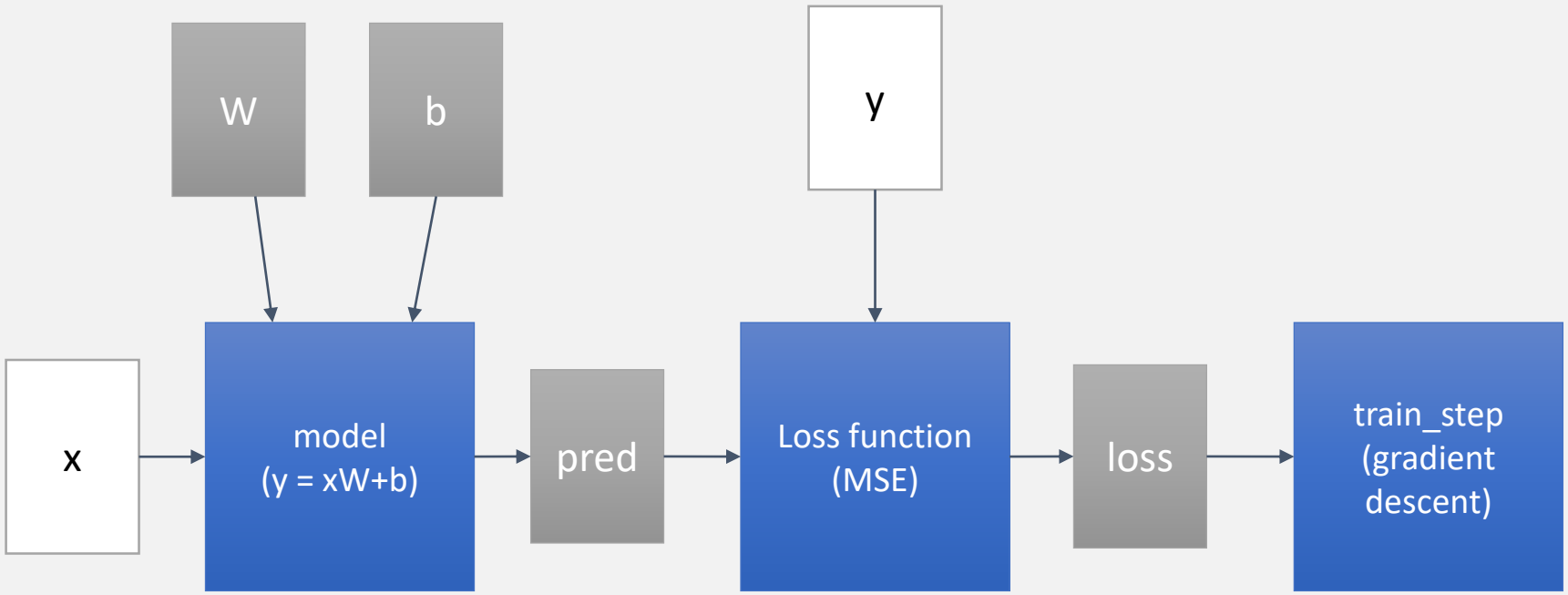
예제: Linear Regression



- Linear Regression (선형 회귀)란?
  - ✓ 데이터들의 1차 함수(직선) 상관관계를 모델링
- 학습 목표 및 방법
  - ✓ 데이터들을 통해 y = xW+b의 W와 b를 찾는 것이 목표
  - ✓ Optimizer : Gradient Descent
  - ✓ Loss Function : MSE (Mean Squared Error)

예제: Linear Regression

- 데이터 연산의 흐름

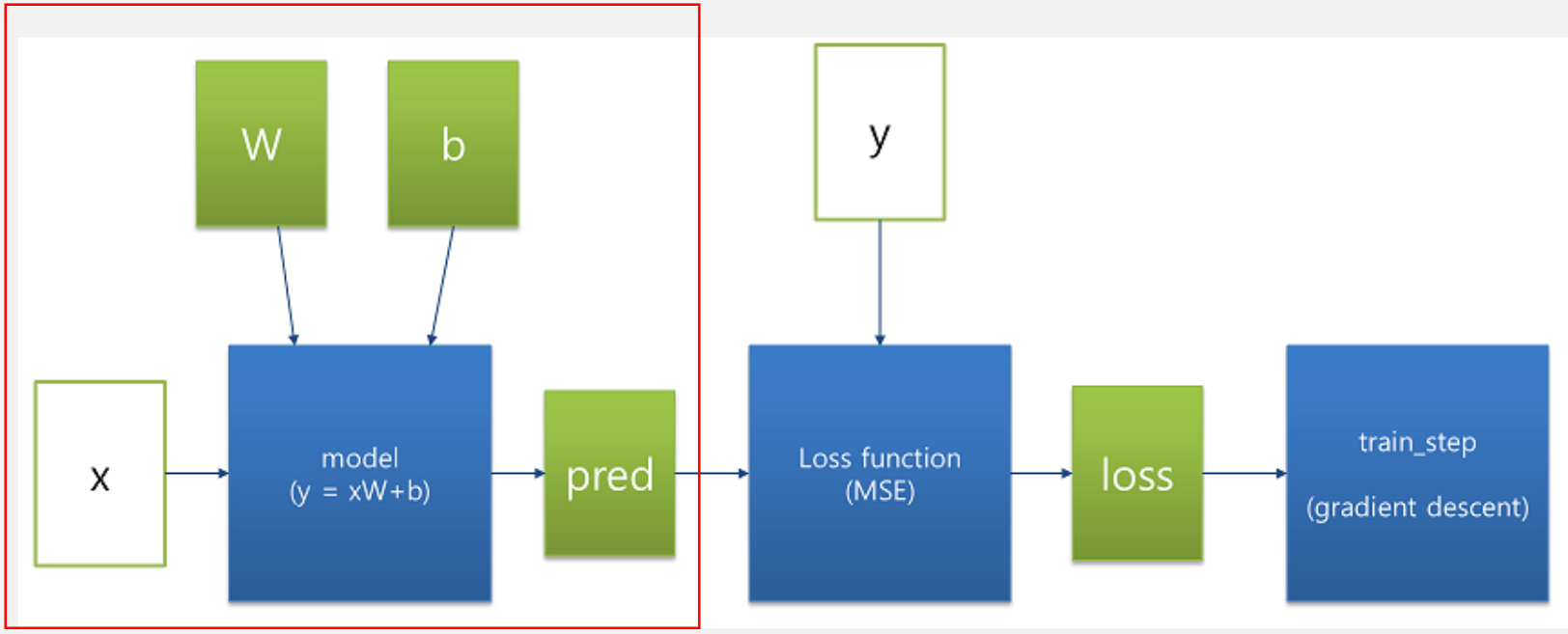


예제: Linear Regression

- 입력 및 모델 정의

```
W = tf.Variable(tf.random_normal(shape=[1], name='w'))
b = tf.Variable(tf.random_normal(shape=[1], name='b'))
x = tf.placeholder(tf.float32)

pred = W*x + b
```



예제: Linear Regression

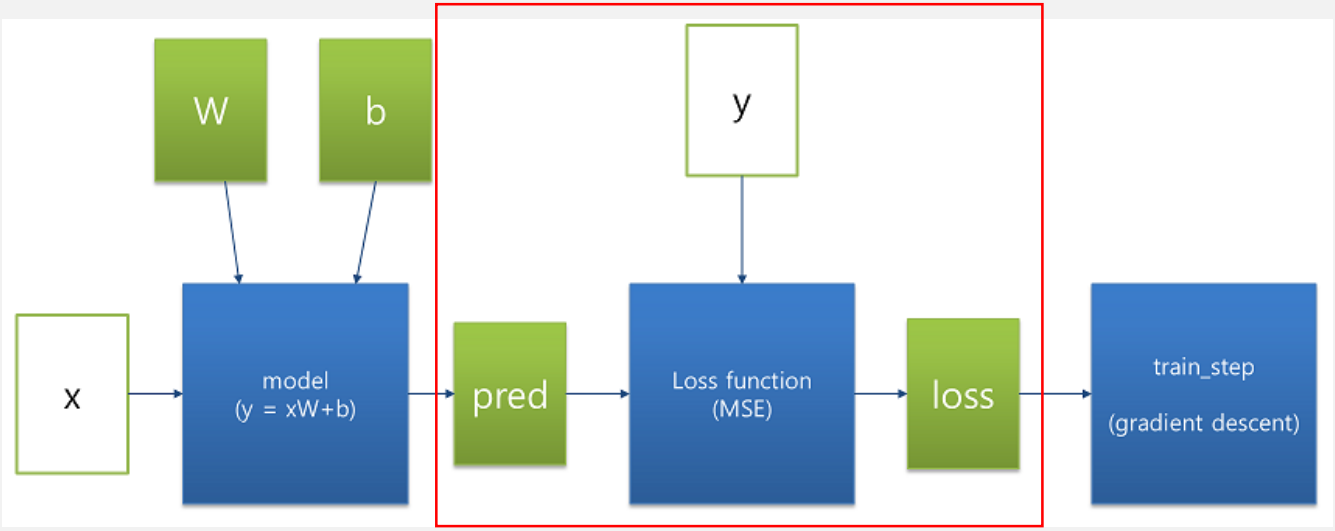
- 정답과 Cost Function 정의

```
y = tf.placeholder(tf.float32)

loss = tf.reduce_mean(tf.square(pred - y))
```

✓ Cost Function = MSE (Mean Squared Error)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$



예제: Linear Regression

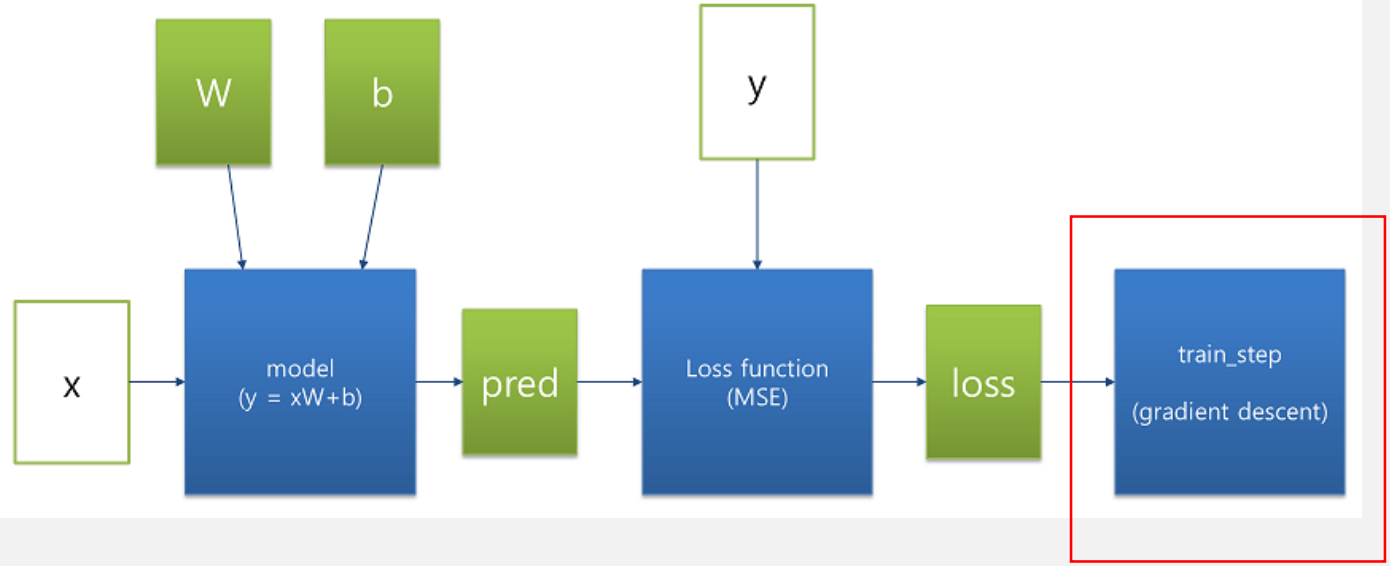
- Optimizer 정의

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

train_step = optimizer.minimize(loss)
```

✓ 학습 방법 : Gradient Descent

✓ Learning Rate : 0.01

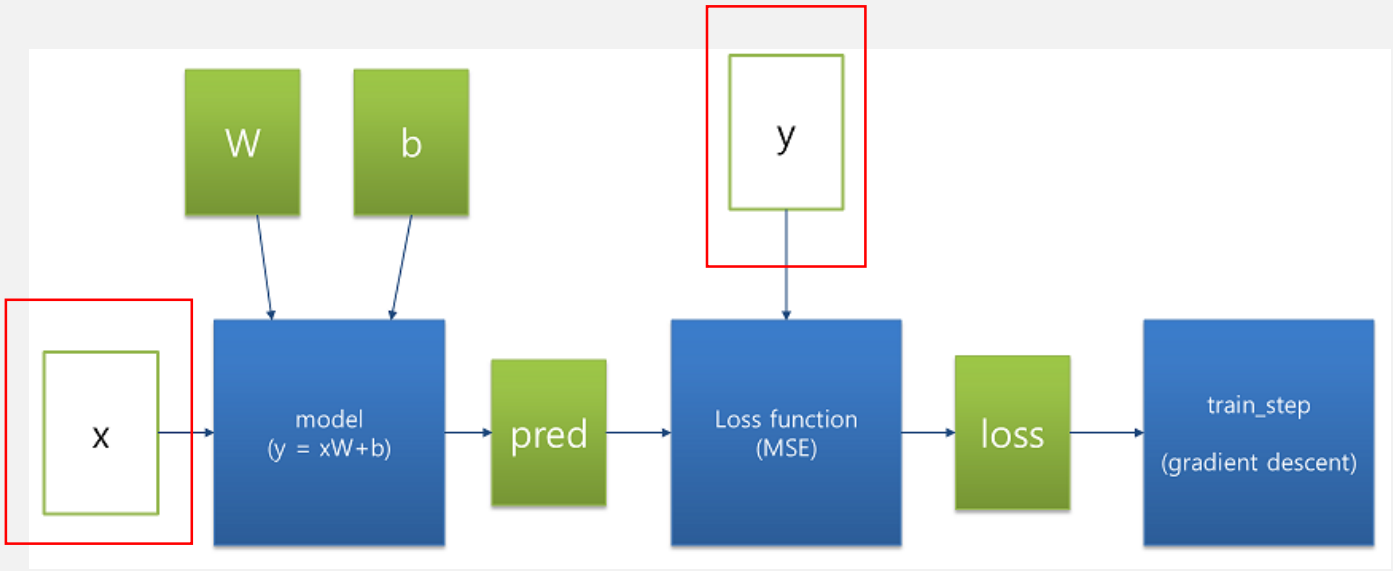


예제: Linear Regression

- 학습셋 구성

```
# y = 2x + 1
x_train = [1,2,3,4]
y_train = [3,5,7,9]
```

✓ 목표 모델 : y = 2x + 1

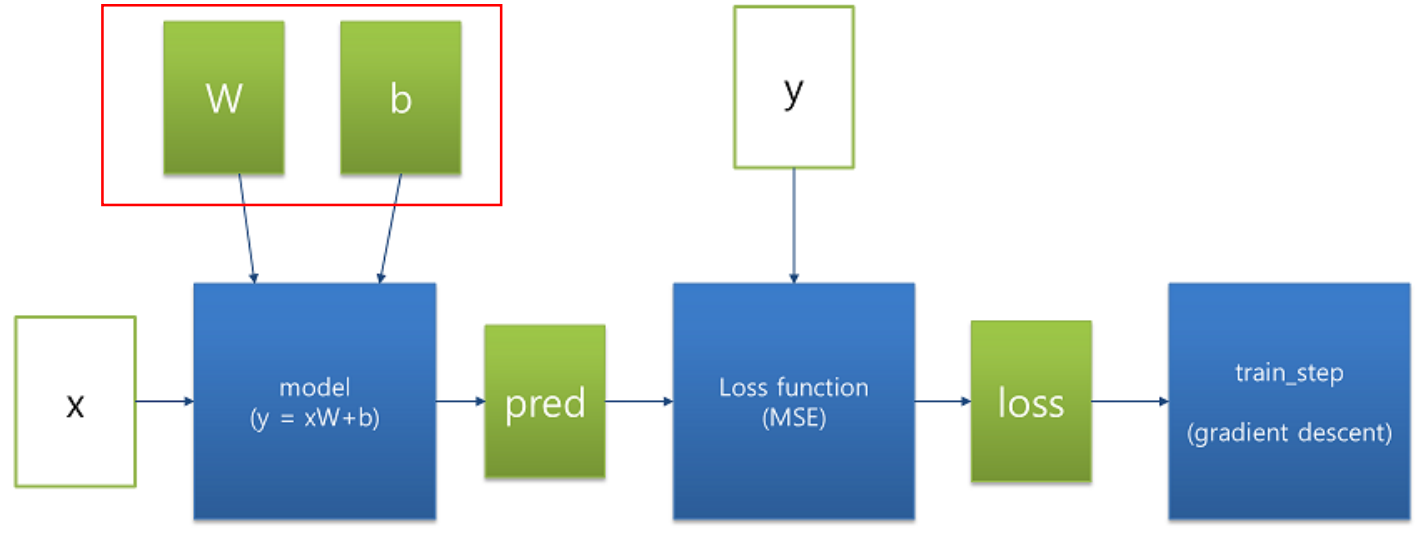


예제: Linear Regression

- 변수 초기화

```
sess = tf.Session()
result = sess.run(tf.global_variables_initializer())
```

✓ 파라미터 초기화 : w, b



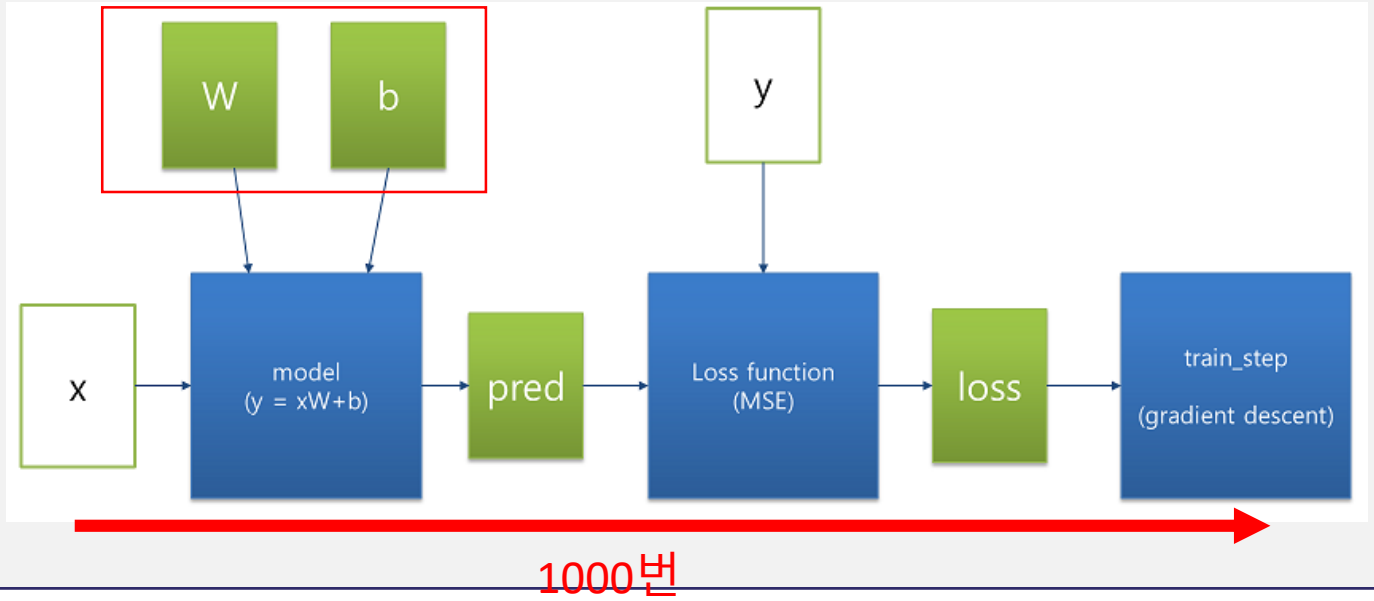
예제: Linear Regression

- 학습 진행

```
for i in range(1000):
    sess.run(train_step, feed_dict={x: x_train, y: y_train})
```

✓ 학습 진행

✓ Epoch : 1000



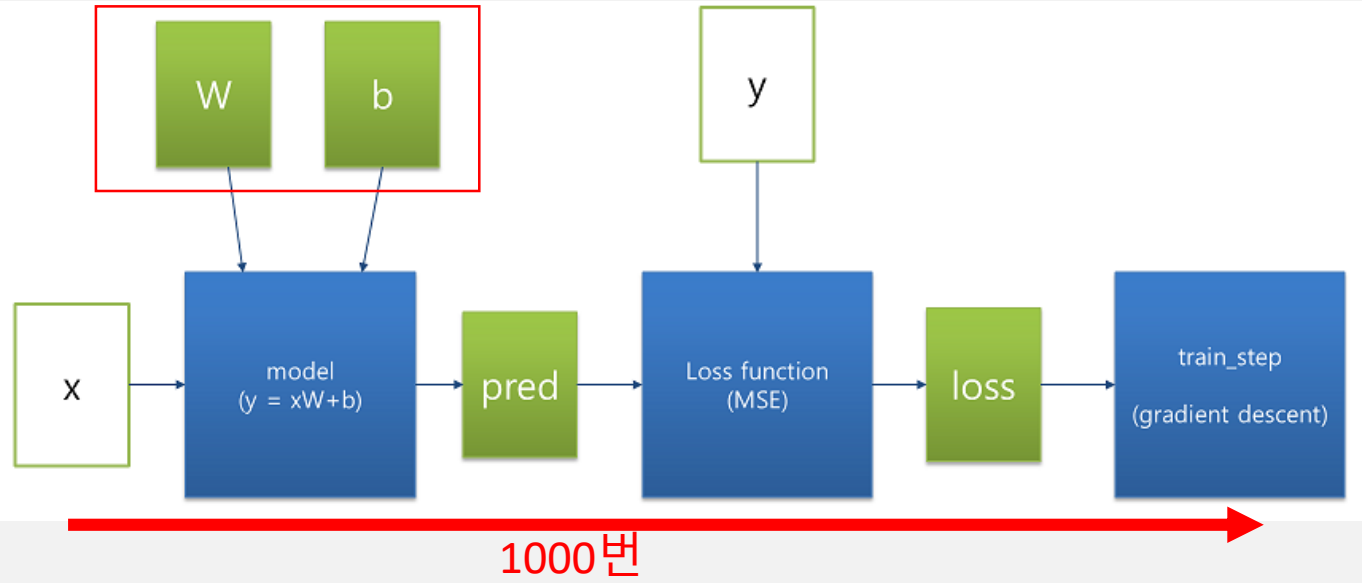
예제: Linear Regression

- 학습 진행

```
for i in range(1000):
    sess.run(train_step, feed_dict={x: x_train, y: y_train})
```

✓ 학습 진행

✓ Epoch : 1000



예제: Linear Regression

- 모델 테스트

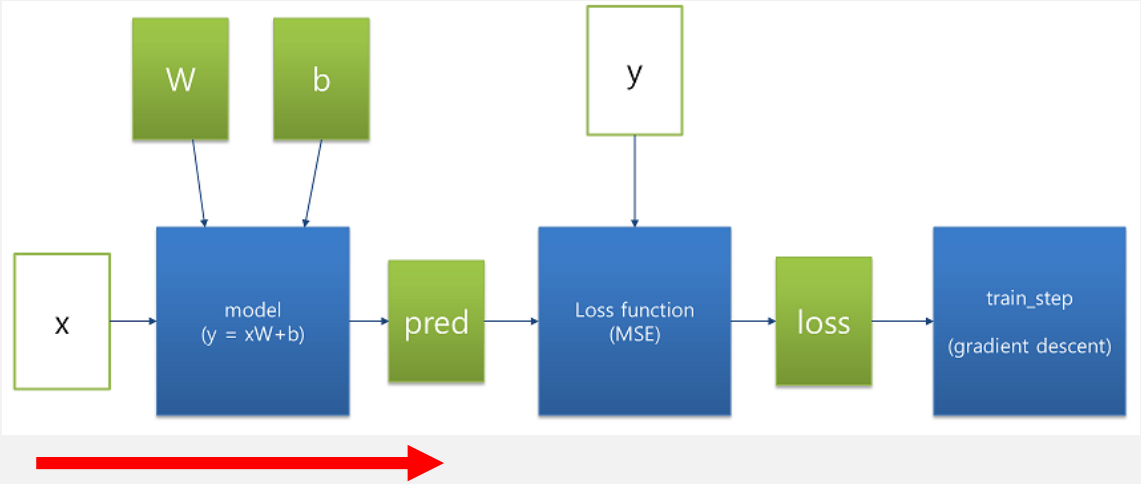
```
x_test = [3, 5, 5, 6]

print (sess.run(pred, feed_dict={x: x_test}))
```

- 결과

```
2019-03-27 17:16:13.734138: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:893] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2019-03-27 17:16:13.734411: I tensorflow/core/common_runtime/gpu/gpu_device.cc:955] Found device 0 with properties:
name: GeForce GTX 980
major: 5 minor: 2 memoryClockRate (GHz) 1.329
pciBusID 0000:01:00.0
Total memory: 3.94GiB
Free memory: 3.87GiB
2019-03-27 17:16:13.734435: I tensorflow/core/common_runtime/gpu/gpu_device.cc:976] DMA: 0
2019-03-27 17:16:13.734439: I tensorflow/core/common_runtime/gpu/gpu_device.cc:986] 0:   Y
2019-03-27 17:16:13.734443: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX 980, pci bus id: 0000:01:00.0)
[ 7.000265  11.0091095 11.0091095 13.013533 ]

Process finished with exit code 0
```



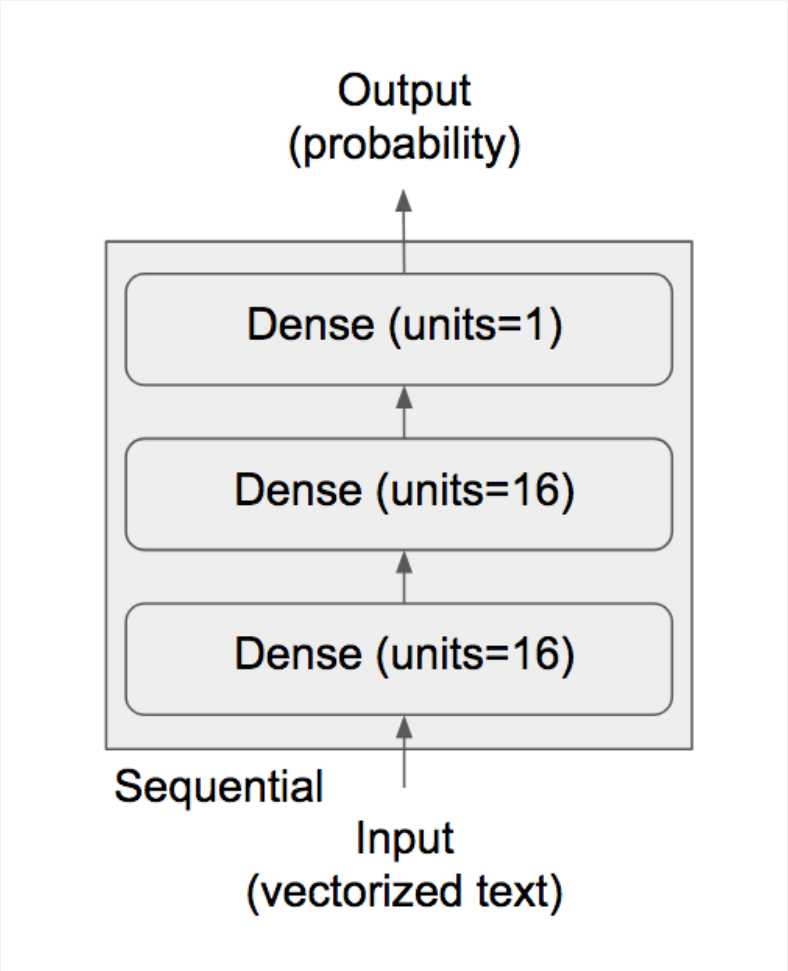
4. 예제: 영화 리뷰 분류: 이진 분류

데이터 준비

- IMDB 데이터셋
  - ✓ 인터넷 영화 데이터베이스로부터 가져온 양극단의 리뷰 50,000개로 이루어진 IMDB 데이터셋
  - ✓ 데이터셋은 훈련 데이터 25,000개와 테스트 데이터 25,000개
  - ✓ 각각 50%는 부정, 50%는 긍정 리뷰로 구성

- 데이터 준비
  - ✓ 같은 길이가 되도록 리스트에 패딩을 추가하고 (samples, sequence_length) 크기의 정수 텐서로 변환합니다
  - ✓ 리스트를 원-핫 인코딩하여 0과 1의 벡터로 변환합니다.
  - ✓ 예를 들면 시퀀스 [3, 5]를 인덱스 3과 5의 위치는 1이고 그 외는 모두 0인 10,000 차원의 벡터로 각각 변환합니다.

신경망 모델 만들기


Output
(probability)
Dense (units=1)
Dense (units=16)
Dense (units=16)
Sequential
Input
(vectorized text)

손실함수와 옵티마이저

- 손실함수
  - ✓ 이진 분류 문제이고 신경망의 출력이 확률이기 때문에(네트워크의 끝에 시그모이드 활성화 함수를 사용한 하나의 유닛으로 된 층을 놓았습니다)
  - ✓ binary_crossentropy 손실이 적합합니다. 이 함수가 유일한 선택은 아니고 예를 들어 mean_squared_error를 사용할 수도 있습니다.
- 데이터 준비
  - ✓ Optimizer → Rmsprop

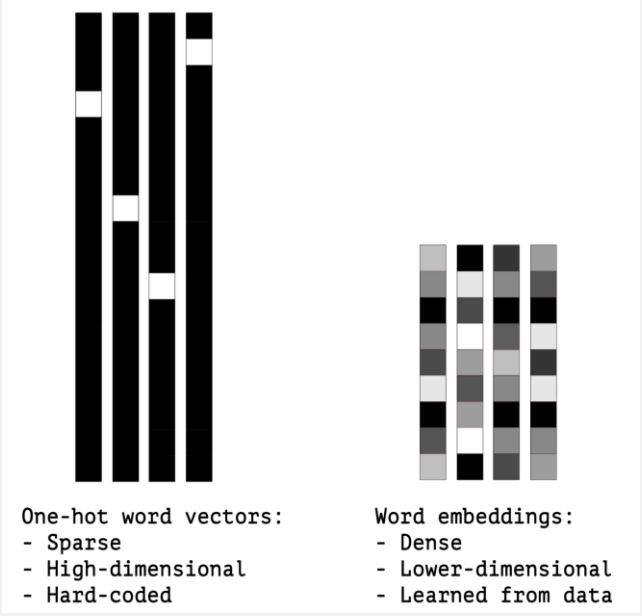
훈련 검증

- 데이터 셋 분할
  - ✓ 훈련하는 동안 처음 본 데이터에 대한 모델의 정확도를 측정하기 위해서는 원본 훈련 데이터에서 10,000의 샘플을 떼어서 검증 세트를 만들어야 합니다.
- 훈련
  - ✓ 이제 모델을 512개 샘플씩 미니 배치를 만들어 20번의 에포크 동안 훈련시킵니다(x_train과 y_train 텐서에 있는 모든 샘플에 대해 20번 반복합니다).
- 훈련 결과 가시화
  - ✓ model.fit() 메서드는 History 객체를 반환합니다.
- 새로운 데이터에 대해 예측하기
  - ✓ Test data set

5. 예제: 단어 임베딩 사용하기

단어 임베딩

- 단어와 벡터를 연관짓는 강력하고 인기 있는 또 다른 방법은 단어 임베딩이라는 밀집 단어 벡터를 사용하는 것입니다.
- 원-핫 인코딩으로 만든 벡터는 희소하고(대부분 0으로 채워집니다) 고차원입니다(어휘 사전에 있는 단어의 수와 차원이 같습니다).
- 반면 단어 임베딩은 저차원의 실수형 벡터입니다(희소 벡터의 반대인 밀집 벡터입니다).



단어 임베딩

- (문서 분류나 감성 예측과 같은) 관심 대상인 문제와 함께 단어 임베딩을 학습합니다. 이런 경우에는 랜덤한 단어 벡터로 시작해서 신경망의 가중치를 학습하는 것과 같은 방식으로 단어 벡터를 학습합니다.
- 풀려는 문제가 아니고 다른 머신 러닝 작업에서 미리 계산된 단어 임베딩을 로드합니다. 이를 사전 훈련된 단어 임베딩이라고 합니다.

단어 임베딩

- 영화 리뷰에서 가장 빈도가 높은 10,000개의 단어를 추출하고 리뷰에서 20개 단어 이후는 버립니다.
- 이 네트워크는 10,000개의 단어에 대해 8 차원의 임베딩을 학습하여 정수 시퀀스 입력(2D 정수 텐서)를 임베딩 시퀀스(3D 실수형 텐서)로 바꿀 것입니다.
- 그 다음 이 텐서를 2D로 펼쳐서 분류를 위한 Dense 층을 훈련하겠습니다.

단어 임베딩

- 영화 리뷰에서 가장 빈도가 높은 10,000개의 단어를 추출하고 리뷰에서 20개 단어 이후는 버립니다.
- 이 네트워크는 10,000개의 단어에 대해 8 차원의 임베딩을 학습하여 정수 시퀀스 입력(2D 정수 텐서)를 임베딩 시퀀스(3D 실수형 텐서)로 바꿀 것입니다.
- 그 다음 이 텐서를 2D로 펼쳐서 분류를 위한 Dense 층을 훈련하겠습니다.

단어 임베딩

- 약 75% 정도의 검증 정확도가 나옵니다. 리뷰에서 20개의 단어만 사용한 것치고 꽤 좋은 결과입니다. 하지만 임베딩 시퀀스를 펼치고 하나의 Dense 층을 훈련했으므로 입력 시퀀스에 있는 각 단어를 독립적으로 다루었습니다.

사전 훈련된 단어 임베딩 사용하기

- 훈련 데이터가 부족하면 작업에 맞는 단어 임베딩을 학습할 수 없습니다. 이럴 땐 어떻게 해야 할까요?

- 풀려는 문제와 함께 단어 임베딩을 학습하는 대신에 미리 계산된 임베딩 공간에서 임베딩 벡터를 로드할 수 있습니다.

- 연구나 산업 애플리케이션에 적용되기 시작된 것은 Word2vec 알고리즘이 등장한 이후입니다. 이 알고리즘은 2013년 구글의 토마스 미코로프가 개발하였으며 가장 유명하고 성공적인 단어 임베딩 방법입니다.

사전 훈련된 단어 임베딩 사용하기

- 케라스의 Embedding 층을 위해 내려받을 수 있는 미리 계산된 단어 임베딩 데이터베이스가 여럿 있습니다. Word2vec은 그 중 하나입니다.
- 인기 있는 또 다른 하나는 2014년 스탠포드 대학의 연구자들이 개발한 GloVe(Global Vectors for Word Representation)입니다. 이 임베딩 기법은 단어의 동시 출현 통계를 기록한 행렬을 분해하는 기법을 사용합니다.
- 이 개발자들은 위키피디아 데이터와 커먼 크롤 데이터에서 가져온 수백만 개의 영어 토큰에 대해서 임베딩을 미리 계산해 놓았습니다.

- GloVe 임베딩을 케라스 모델에 어떻게 사용하는지 알아보죠.