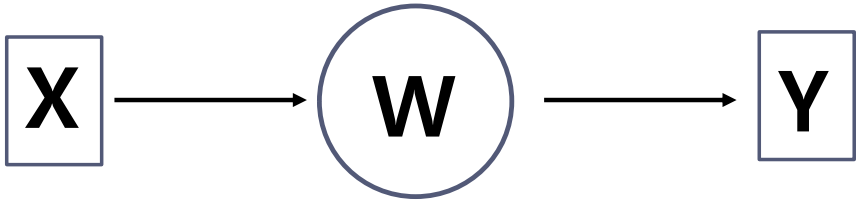
Category 1
Basic Model

Basic Model 구현

- 단항(1차원) 회귀 모델 : x변수가 1 개인 회귀 모델

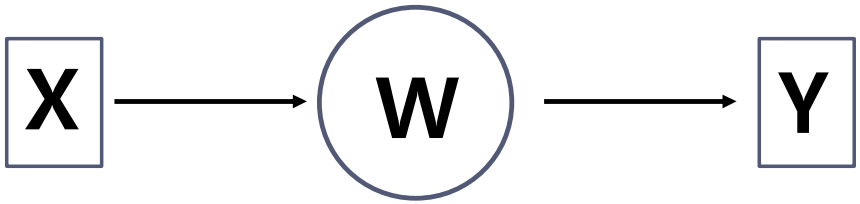


1차원 Shape [n, **1**] [**1**, 1] [n, 1]

	X	Y
0	-1.0	-3.0
1	0.0	-1.0
2	1.0	1.0
3	2.0	3.0
4	3.0	5.0
5	4.0	7.0

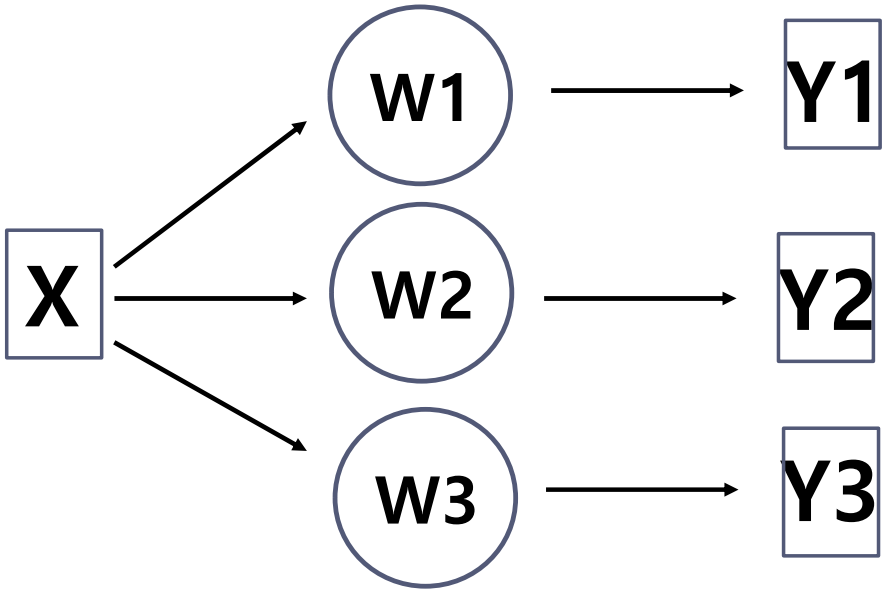
- **Units = 1 : 뉴런의 수 = 1**

```
model = tf.keras.Sequential([
    keras.layers.Dense(units=1, input_shape=[1])
])
```



- **Units = 3 : 뉴런의 수 = 3**

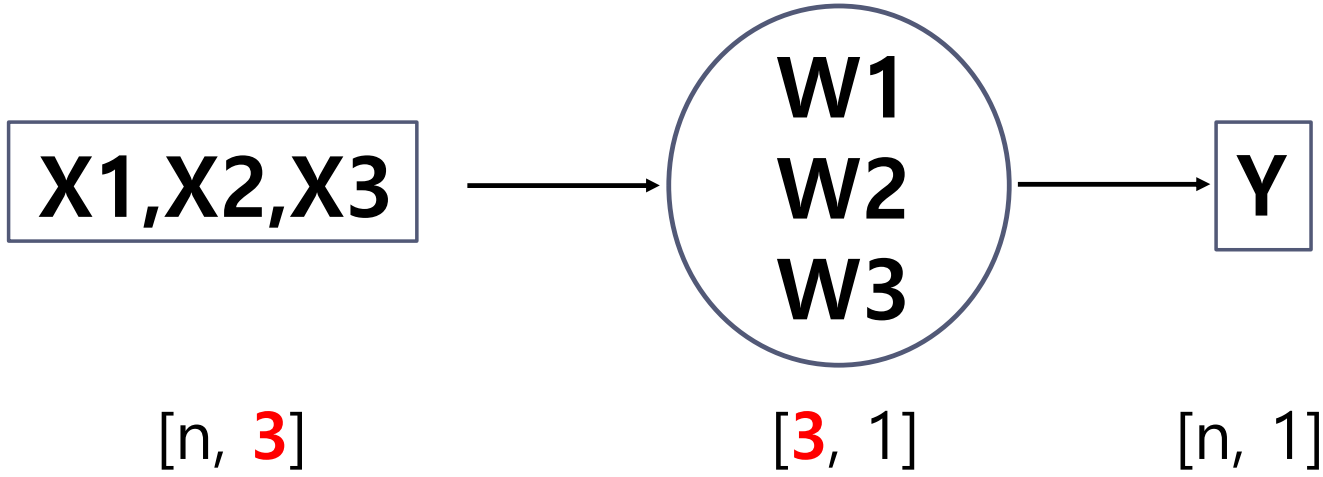
```
model = tf.keras.Sequential([
    keras.layers.Dense(units=3, input_shape=[1])
])
```



[n, 3]

Y는 [n, 1] 이어야 하므로 은닉층에서 사용가능

- 다항(다차원) 회귀 모델 : x변수가 여러 개인 회귀 모델



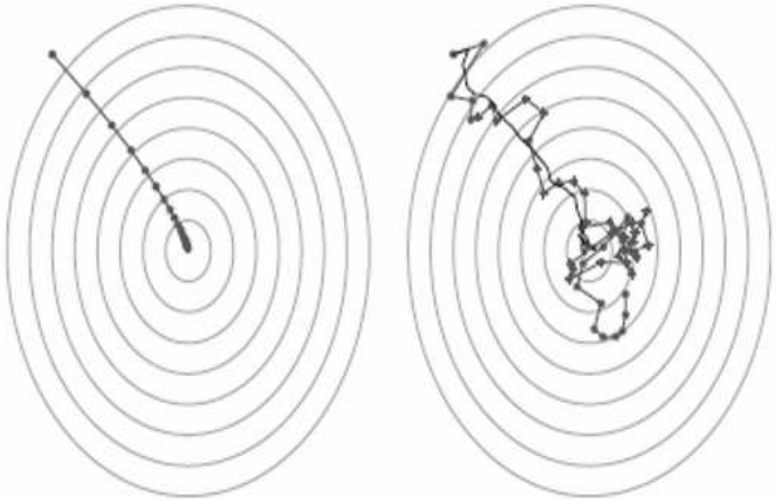
	X1	X2	X3	Y
0	-1.0	0.1	-2.0	-3.0
1	0.0	0.2	1.0	-1.0
2	1.0	0.4	2.0	1.0
3	2.0	0.6	3.0	3.0
4	3.0	0.7	4.0	5.0
5	4.0	0.9	5.0	7.0

"X변수의 개수 만큼 weight이 필요하다"

SGD Optimizer

- Stochastic Gradient Descent(SGD)

  https://mangkyu.tistory.com/62



<BGD, SGD>

- ✓ Gradient Descent를 전체 데이터(Batch)가 아닌 일부 데이터의 모음(Mini-Batch)를 사용하는 방법
- ✓ BGD(Batch Gradient Descent)는 하나의 step을 위해 전체 데이터를 계산 하므로 계산량이 많음
- ✓ SGD(Stochastic Gradient Descent)는 Mini-Batch를 사용하여 다소 부정확할 수는 있지만 계산 속도가 훨씬 빠르기 때문에, 같은 시간에 더 많은 Step을 나아갈 수 있음
- ✓ Local Minima에 빠지지 않고 Global Minima에 수렴할 가능성이 더 높음

https://seamless.tistory.com/38

Optimizer

- optimizers
  - Overview
  - Adadelta
  - Adagrad
  - Adam
  - Adamax
  - Ftrl
  - Nadam
  - Optimizer
  - RMSprop
  - SGD
  - deserialize
  - get
  - serialize
  - schedules

`class Adadelta` : Optimizer that implements the Adadelta algorithm.

`class Adagrad` : Optimizer that implements the Adagrad algorithm.

`class Adam` : Optimizer that implements the Adam algorithm. **(고성능)**

`class Adamax` : Optimizer that implements the Adamax algorithm.

`class Ftrl` : Optimizer that implements the FTRL algorithm.

`class Nadam` : Optimizer that implements the NAdam algorithm.

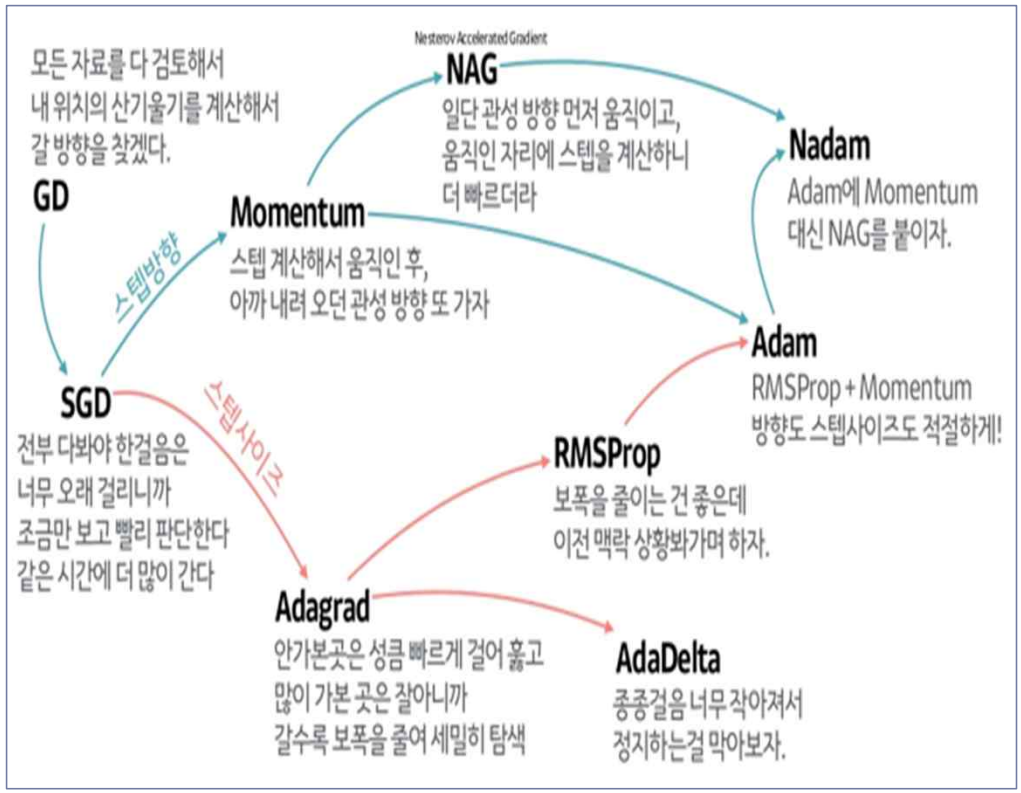
`class Optimizer` : Base class for Keras optimizers.

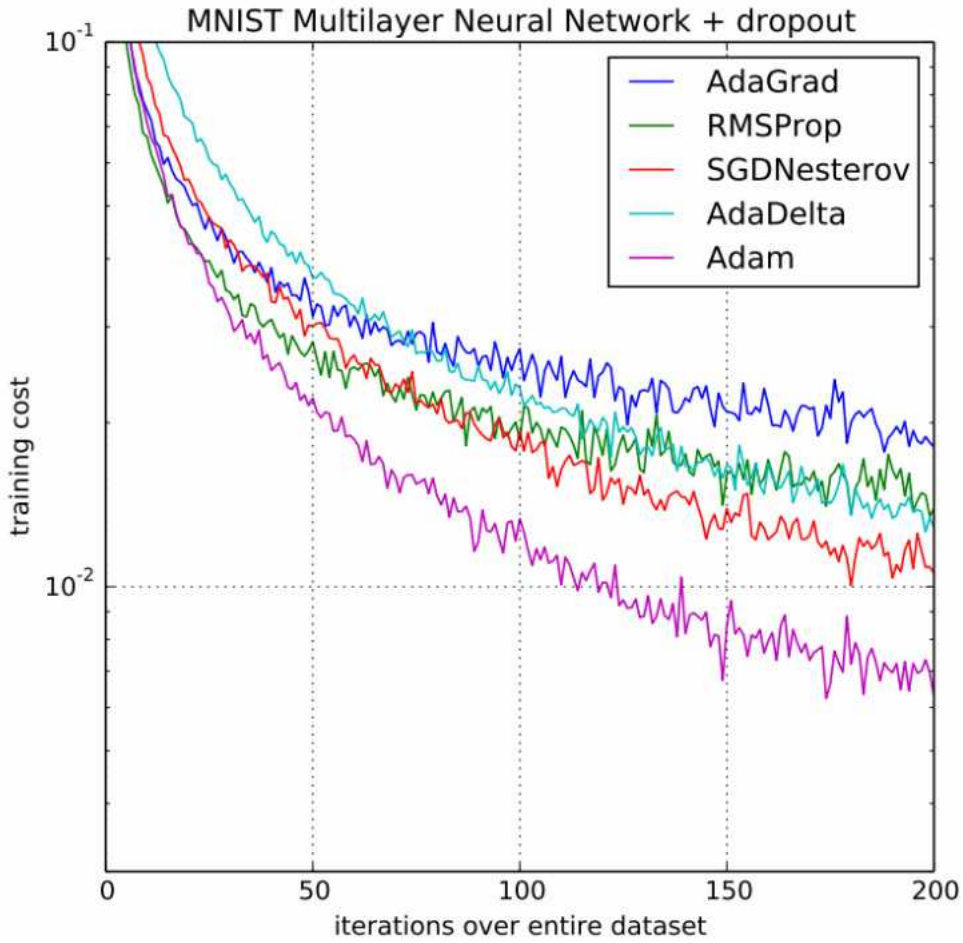
`class RMSprop` : Optimizer that implements the RMSprop algorithm.

`class SGD` : Gradient descent (with momentum) optimizer.

Adam Optimizer

Optimizer 계보





https://seamless.tistory.com/38

loss 종류

mean_squared_error : 평균 제곱 오차, 회귀 모델에 사용

huber : Huber loss, 회귀 모델에 사용 https://process-mining.tistory.com/130
평균 제곱 오차 함수와 절대 값 함수의 조합 https://ichi.pro/ko/hubeo-sonsil-wae-geuleongayo-242709802738803

binary_crossentropy : 이진 분류 loss , 이진 분류 모델에 사용

categorical_crossentropy : 다중 분류 loss, One-hot encoding 된 클래스, [0.2, 0.3, 0.5] 와 같은 출력 값과 실측 값의 오차 값을 계산한다.

sparse_categorical_crossentropy : 다중 분류 loss, Y값이 One-hot encoding 처리 안된 정수 타입인 경우에 사용

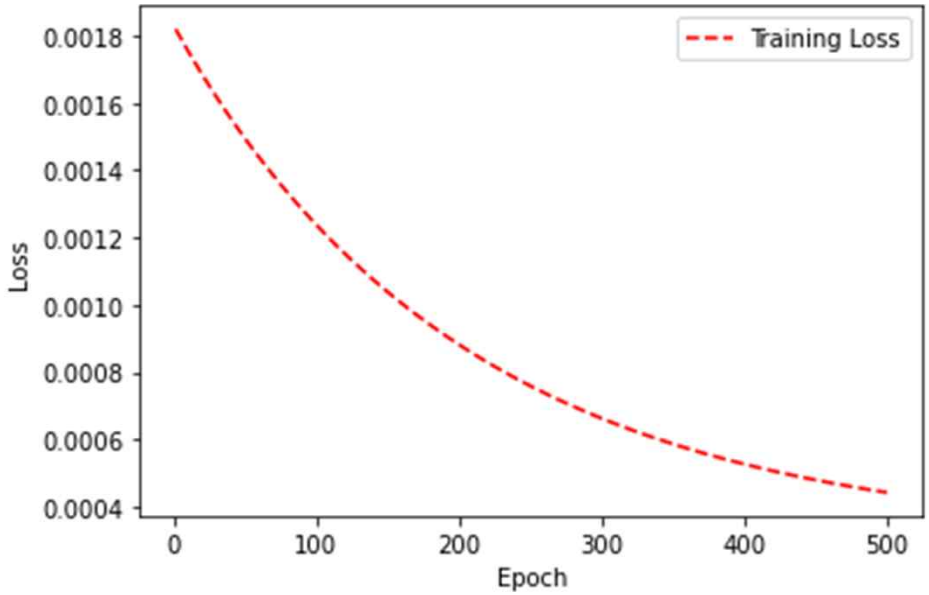
Epochs

- 한 번의 epoch는 전체 데이터 셋에 대해 forward pass/backward pass 과정을 거친 것을 말함. 즉, 전체 데이터 셋에 대해 한 번 학습을 완료한 상태

```
Epoch 1/500
1/1 [==============================] - 0s 292ms/step - loss: 23.3287
Epoch 2/500
1/1 [==============================] - 0s 2ms/step - loss: 18.6364
Epoch 3/500
1/1 [==============================] - 0s 2ms/step - loss: 14.9389
Epoch 4/500
1/1 [==============================] - 0s 2ms/step - loss: 12.0242
Epoch 5/500
1/1 [==============================] - 0s 2ms/step - loss: 9.7255
Epoch 6/500
1/1 [==============================] - 0s 2ms/step - loss: 7.9115
Epoch 7/500
1/1 [==============================] - 0s 2ms/step - loss: 6.4791
Epoch 8/500
1/1 [==============================] - 0s 2ms/step - loss: 5.3468
Epoch 9/500
1/1 [==============================] - 0s 2ms/step - loss: 4.4509
Epoch 10/500
1/1 [==============================] - 0s 996us/step - loss: 3.7410
```

Epoch와 loss의 변화

```
history = model.fit(xs, ys, epochs=500)
import matplotlib.pyplot as plt
epoch_count = range(1, len(history.history['loss']) + 1)
plt.plot(epoch_count, history.history['loss'], 'r--')
plt.legend(['Training Loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



save()함수로 모델 저장하기

```
1 # 학습 된 모델 저장
2 model.save("mymodel.h5")
```

```
1 !dir
```

```
C 드라이브의 볼륨: Windows


C:\Users\storm\바탕화면\텐서플로자격증취득과정\강의소스\Category 1 - Basic Model 디렉터리

2021-06-12  오전 02:42    <DIR>          .
2021-06-12  오전 02:42    <DIR>          ..
2021-06-11  오후 07:38    <DIR>          .ipynb_checkpoints
2021-06-12  오전 02:34           276,320 01_Basic_Model.ipynb
2021-06-12  오전 02:33             8,476 model.png
2021-06-12  오전 02:33            14,232 mymodel.h5
               3개 파일             299,028 바이트
               3개 디렉터리  66,407,227,392 바이트 남음
```

HDF5

https://www.hdfgroup.org/solutions/hdf5/

- h5 파일은 계층적 데이터 형식(Hierarchical Data Format, HDF)은 The HDF Group에 의해 관리되고 있는 대용량의 데이터를 저장하기 위한 바이너리 파일
- 장점
  - XML과 동일하게 자기기술적으로 구성되어 있어 데이터 형식을 파일 안에 기술 가능하다.
  - 많은 양의 데이터를 저장 가능하다.
  - 검색 속도가 빠르다.
  - 병렬 입출력을 지원한다.
  - 데이터의 무작위 조회(Random Access)가 가능하다.
  - 20여년 이상 개발되어온 포맷으로 안정적이다.
  - 수많은 프로그래밍 언어와 오픈소스 라이브러리를 통한 API가 지원된다.

https://www.tensorflow.org/guide/keras/save_and_serialize?hl=ko

전체 모델 저장 및 로딩

전체 모델을 단일 아티팩트(결과물)로 저장할 수 있습니다.

다음을 포함합니다.

- 모델의 아키텍처 및 구성
- 훈련 중에 학습된 모델의 가중치 값
- 모델의 컴파일 정보(compile()이 호출된 경우)
- 옵티마이저와 그 상태
  (훈련을 중단한 곳에서 다시 시작할 수 있게해줌)

load_model()함수로 저장된 모델 불러오기

```
1 # 저장된 모델 불러오기
2 new_model = tf.keras.models.load_model('mymodel.h5')
3 new_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 1)                 4
=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
```

```
1 # Load된 모델로 예측
2 print(new_model.predict([[10.1,7,8.0]]))
```

```
[[21.259745]]
```

시험 시 모델 저장하여 제출

- 텐서플로 인증 자격 시험의 5개 유형의 모든 답안은 마지막에 반드시 아래와 같이 모델을 저장해야 제출하여 채점을 할 수 있다

model.save("mymodel.h5")

- Google Colab이나 주피터 노트북에서 학습하여 저장한 파일을 PyCharm 프로젝트 디렉토리에 복사하여 제출 할 수도 있다**(PC에 GPU가 없는 경우 시간을 절약할 수 있다)**

The End