

디지털 금융 특수논제 - Introduction to Deep Learning

2023-12-26

우지환, Ph.D., MBA

지난 강의 질문

지난 강의 후 들어온 질문 사항들...



Q1. 지난주 수업 내용과 관련하여, 단순한 선형모델을 만들었는데, 실무에서 데이터가 주어지고 해당 데이터를 가지고 선형을 만드는 데는 어떠한 모델을 사용하는 것이 효율적일까요? 특히 금융데이터에서는 어떠한 모델을 사용하셨을 때 예측력이 상대적으로 좋으셨었는지 궁금합니다!

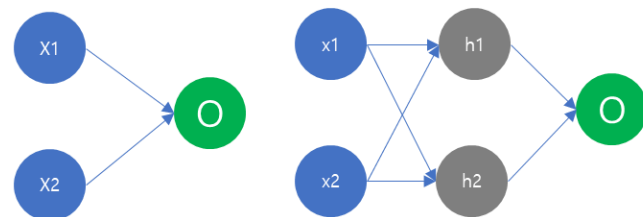
금융데이터의 정의가 넓기 때문에 기본적으로는 캐바케 입니다. 신용 평가시 머신러닝 기법을 활용한 모델이 좋았고, 시계열 데이터의 경우는 RNN→LSTM →Transformer 모델 사용

Q2. 요즘 LLM 들이 나오면서 얼마큼의 파라미터로 학습을시켰다. 라는 말들을 하는데요...에게 무슨 말인지 모르겠습니다.

170B 만큼 데이터를 학습했다.... 하는데...이게 무슨 말일까요...어디 찾아봐도 딱히 나오지는 않지만 다들 쓰는 표현이어서...이해가 잘 안가네요. xxx 개 파라미터 만큼 학습을 시켰다는게 개발할때 파라미터들가는 것 같은 그런 파라미터를 말을 하는건지 어떤건지...설명해주실수 있으면 너무 감사하겠습니다.

연결된 선 하나하나가 parameter 입니다. 엄밀하게 말하면, bias도 포함합니다.

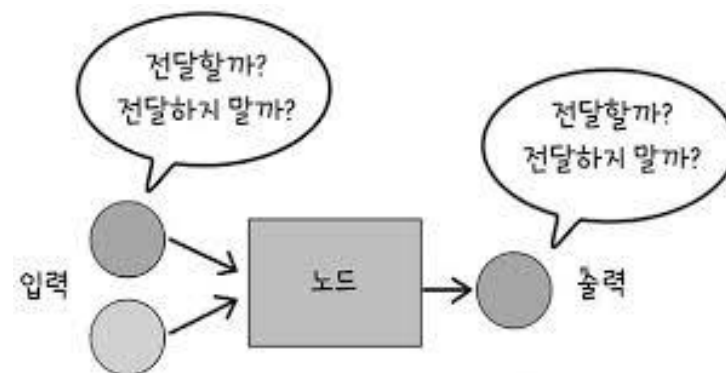
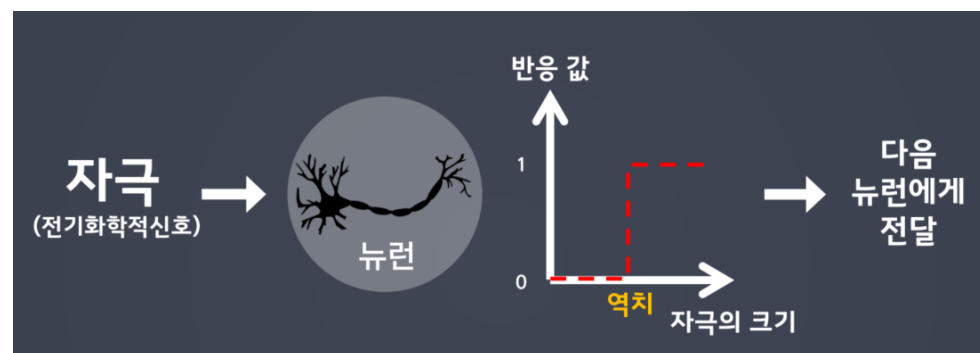
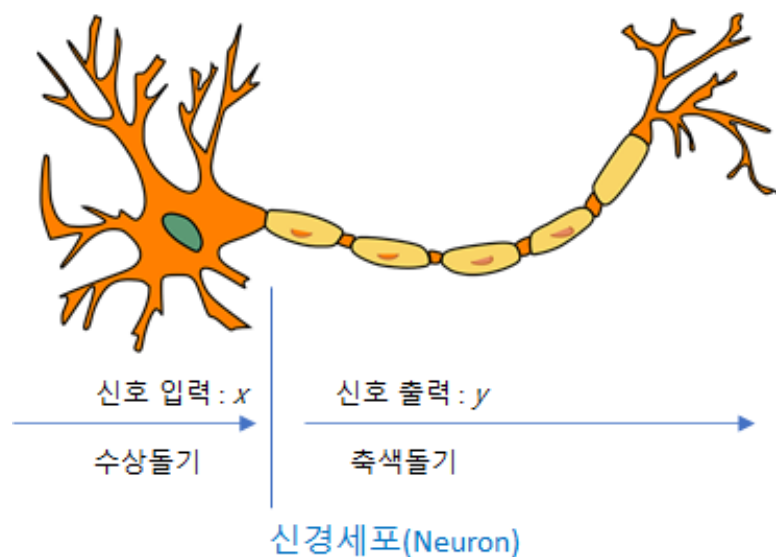
단층 신경망 깊은 신경망 (Deep Learning)



Perceptron 등장

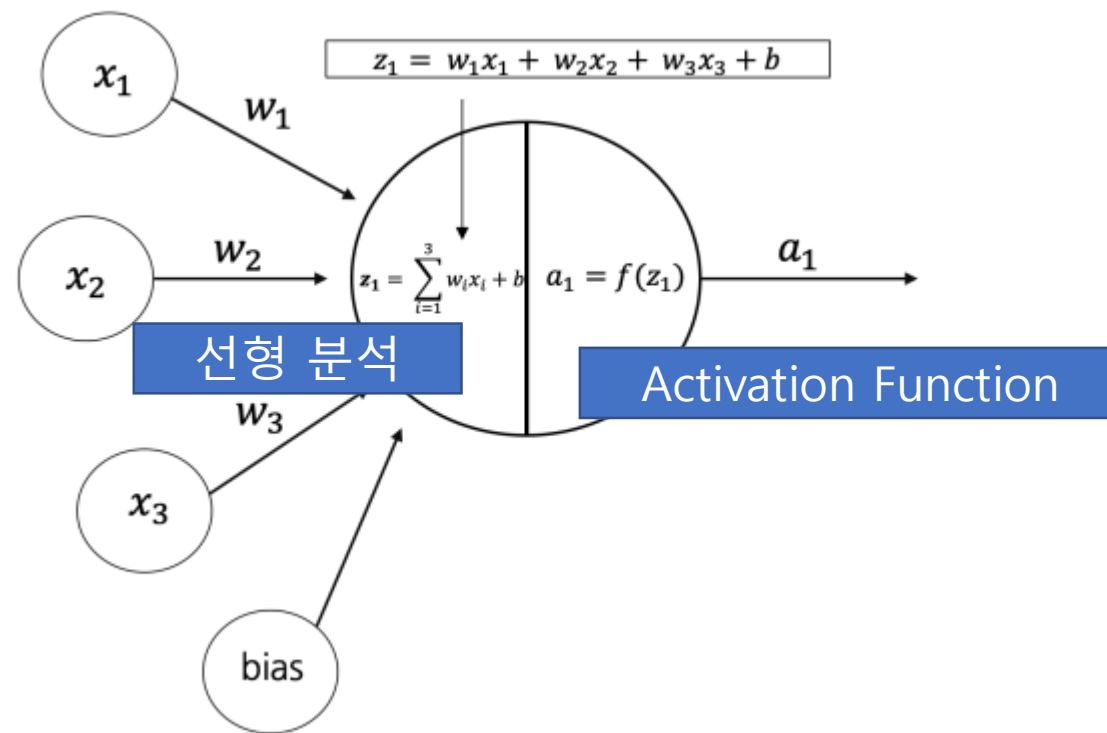
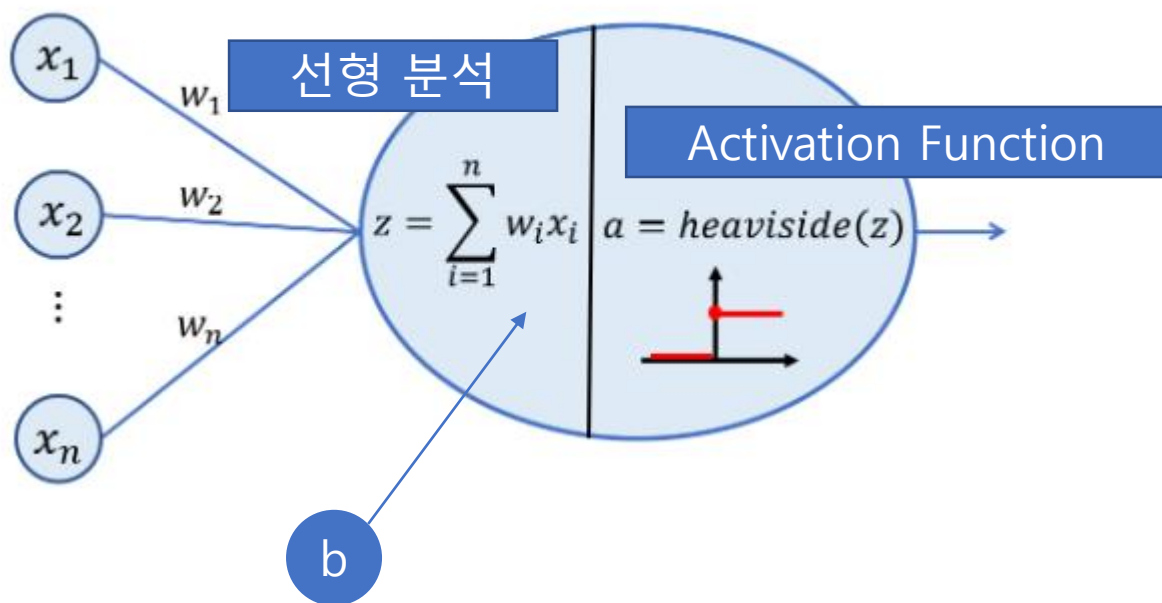
인간의 신경세포(뉴런)의 정보를 전달하는 방법에서 Perceptron을 개발

Perceptron은 입력 신호에 대해 중요도에 따라 가중치를 곱해서 더한 뒤, bias를 더하고, 이 값이 문턱치(역치)보다 클 경우에 다음 단계로 전달함



Perceptron의 수학적 표현

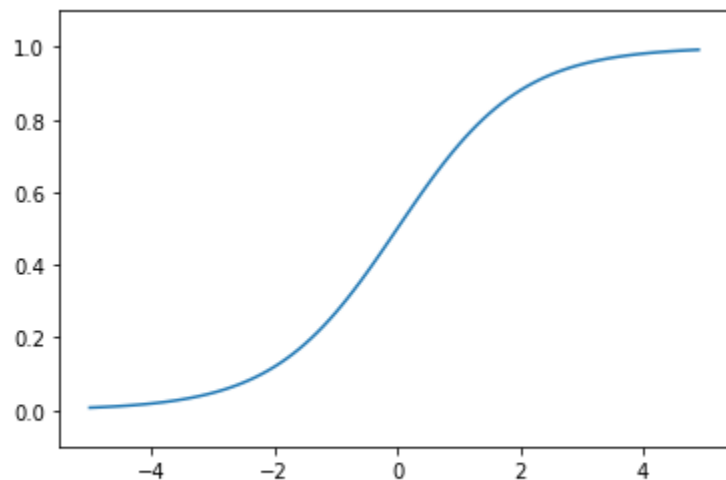
Perceptron은 입력 신호에 대해 중요도에 따라 가중치를 곱해서 것과, Bias를 더하는 Part와 총 합을 문턱치 값(Threshold)과 비교해서 다음 Perceptron으로 전달할지 말지를 결정하는 Part로 구성



Activation Function의 종류

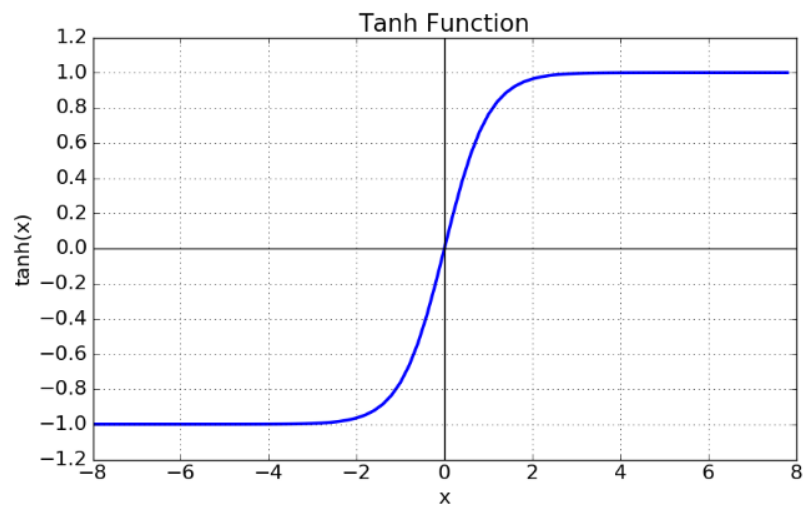
Activation Function의 존재로 인해서, 비선형성을 추가할 수 있습니다.

$$h(x) = \frac{1}{1 + \exp(-x)}$$

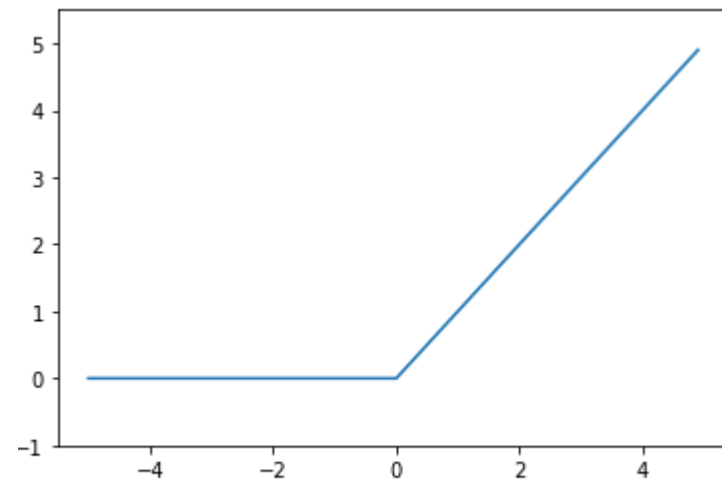


Sigmoid Function

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



$$f(x) = \max(0, x)$$



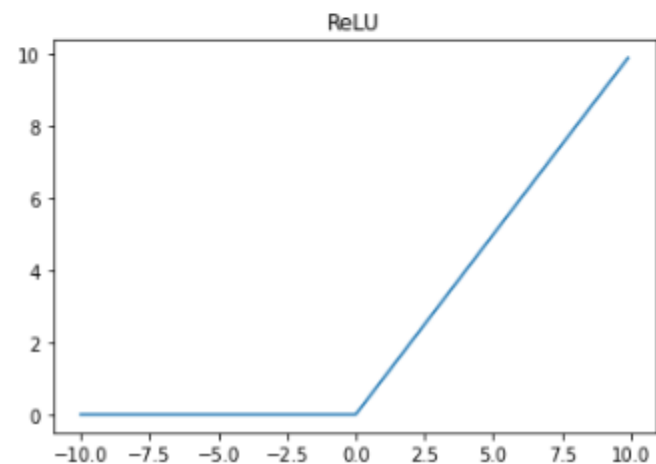
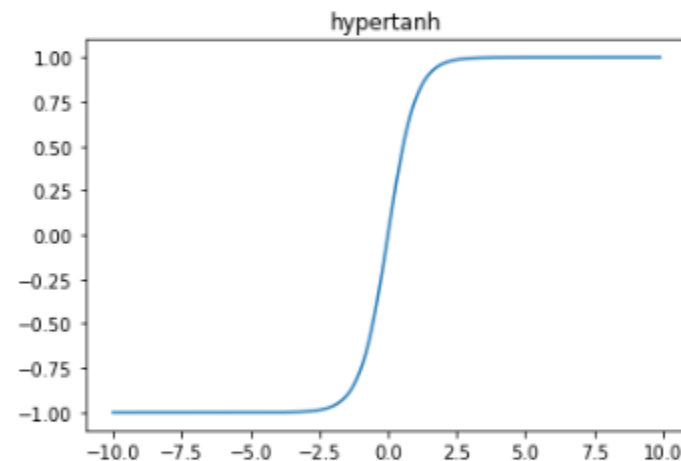
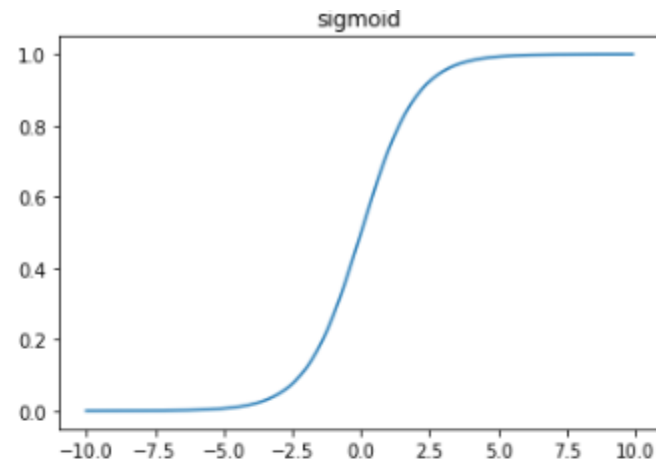
ReLU Function

실습: 활성화 함수의 구현

활성화 함수를 이용해서, 비선형을 추가할 수 있습니다

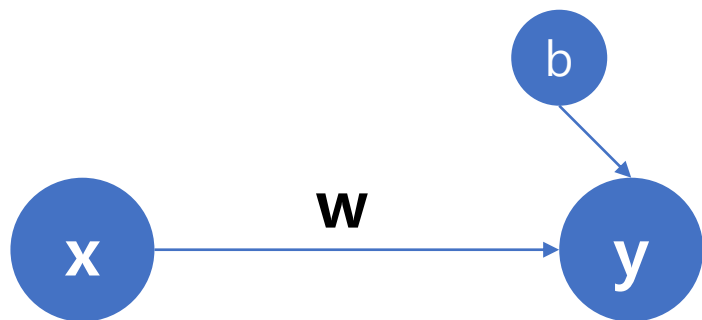
```
def sigmoid(x):  
    return 1/(1+np.exp(-x))  
def hypertanh(x):  
    return (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))  
def ReLU(x):  
    return np.maximum(0,x)
```

```
x=np.arange(-10,10,0.1)  
plt.plot(x, sigmoid(x), label='sig')  
plt.title('sigmoid')  
plt.show()  
plt.plot(x, hypertanh(x), label='hypertanh')  
plt.title('hypertanh')  
plt.show()  
plt.plot(x, ReLU(x), label='ReLU')  
plt.title('ReLU')  
plt.show()
```

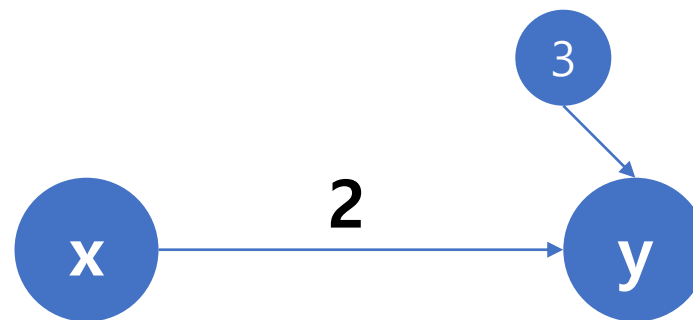


선형 분석 모델 살펴보기

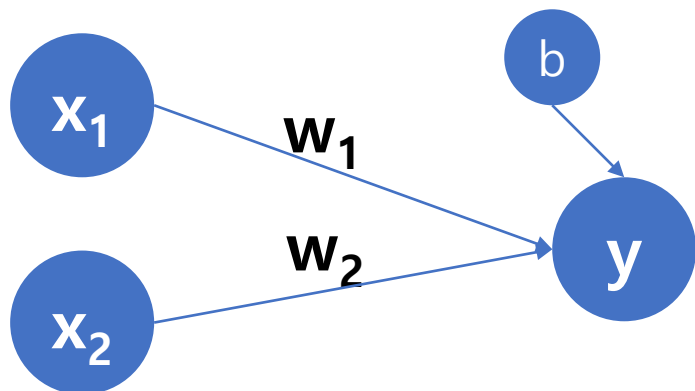
$$y = w * x + b$$



$$y = 2 * x + 3$$

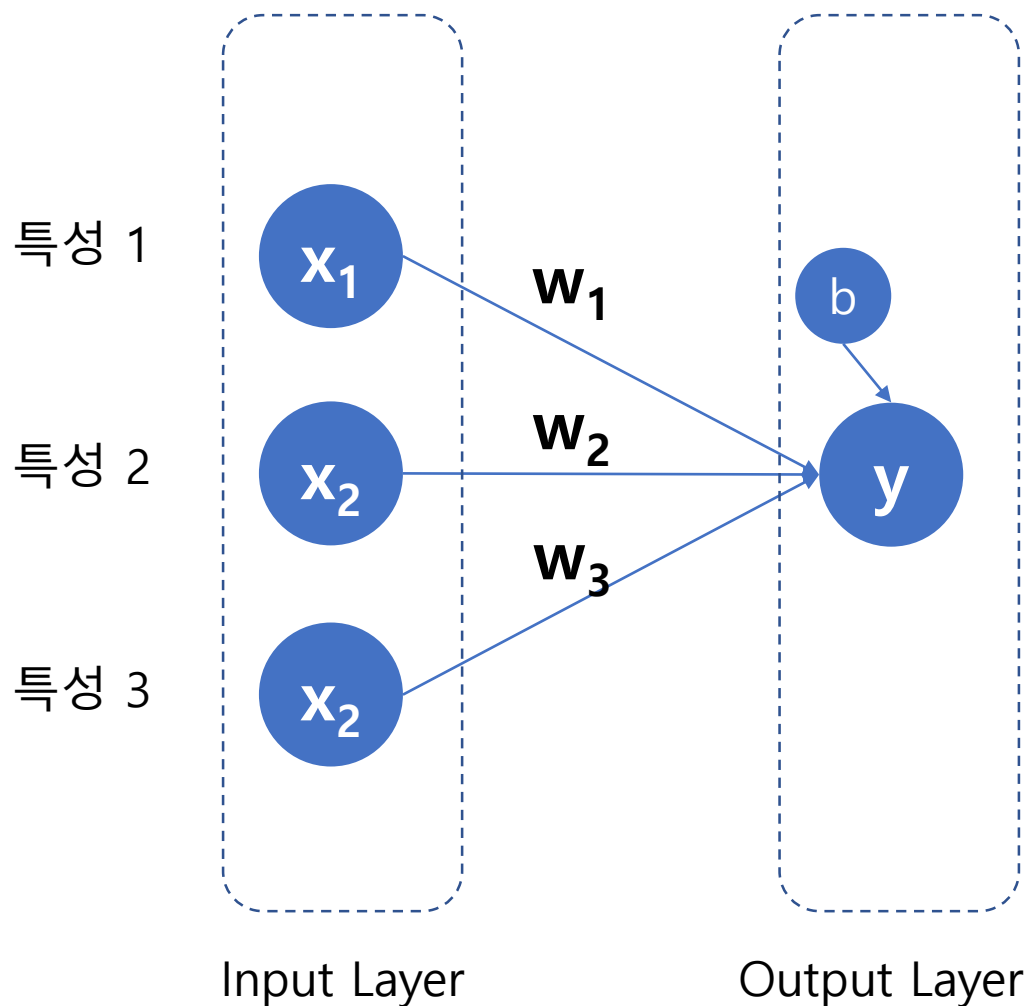


$$y = w_1 * x_1 + w_2 * x_2 + b$$



Bias(b)는 한 셀에 한 뉴런에 하나만 존재함

선형 모델의 물리적인 의미: 압축



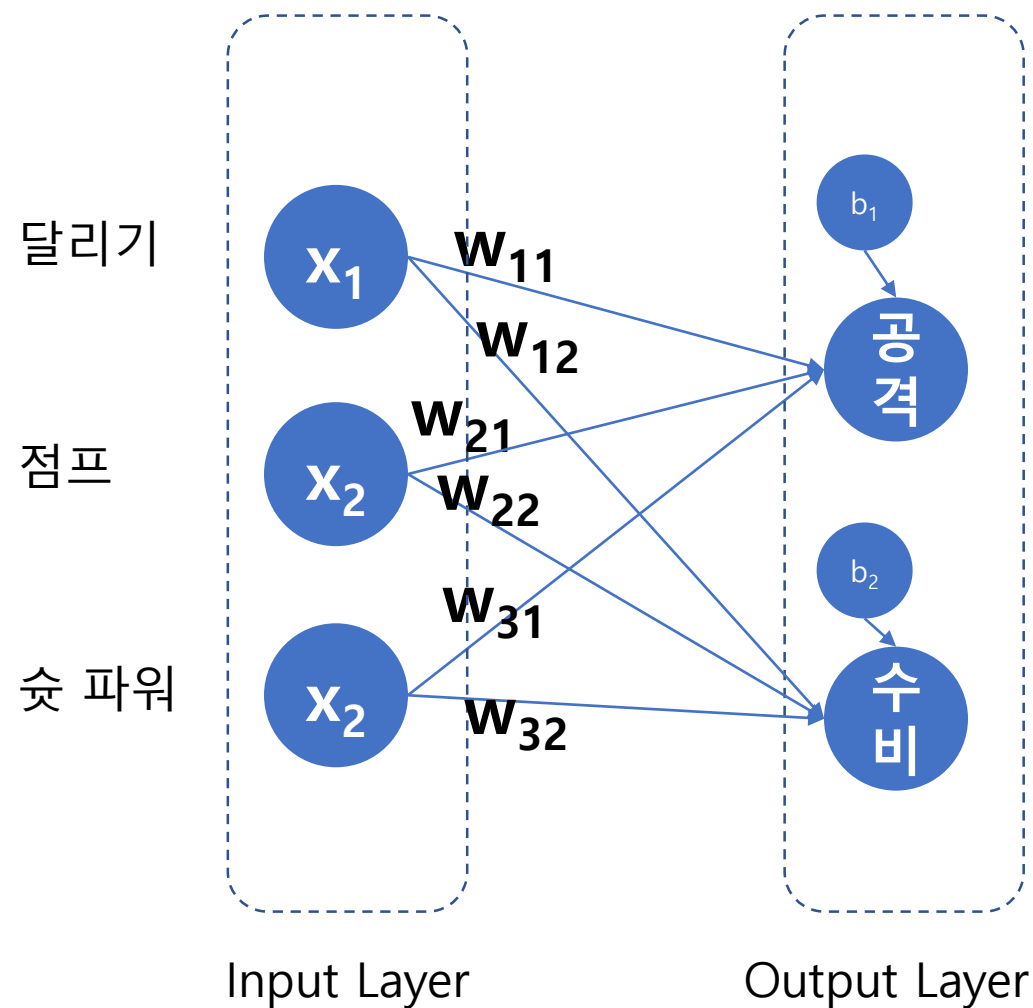
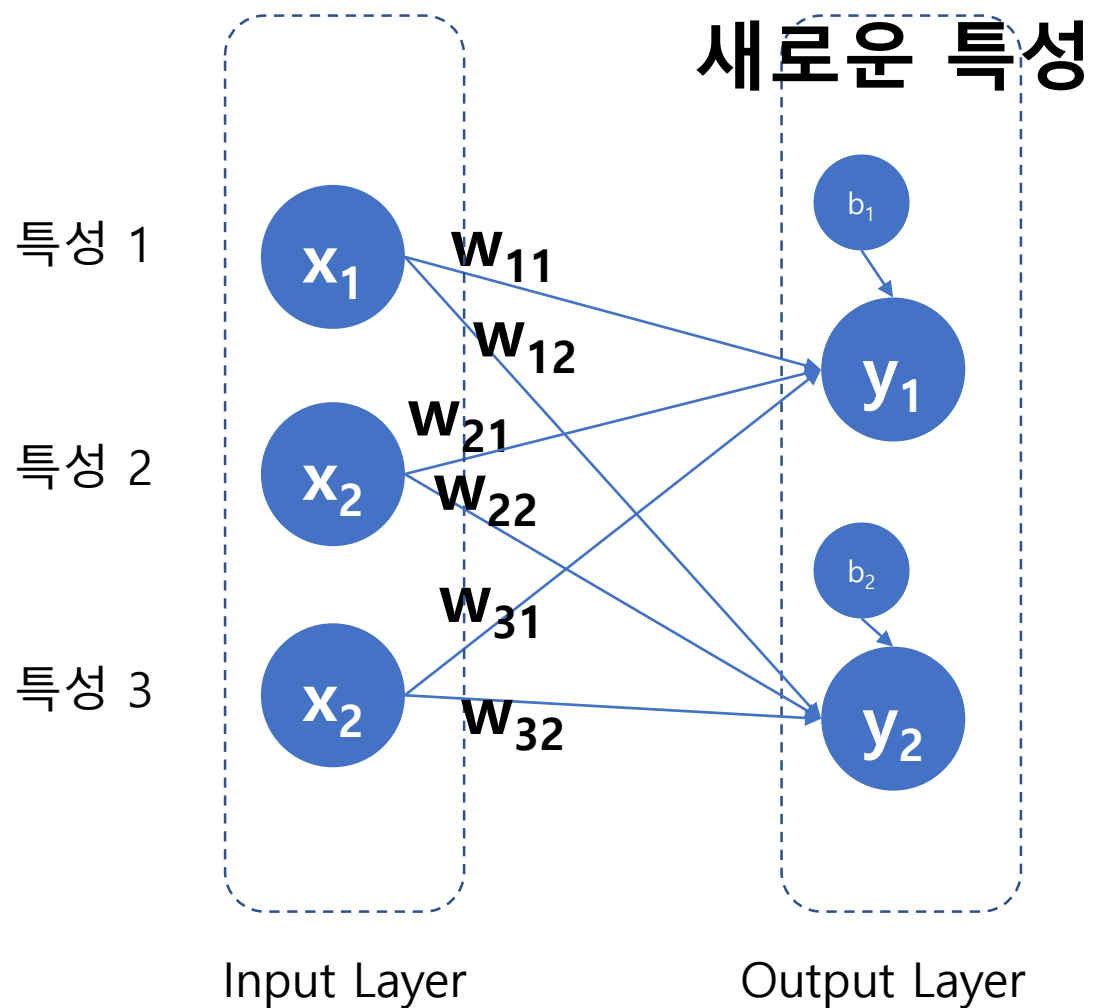
어느 결혼정보회사의 남성 회원 등급표1

| 등급 | 재산내역 | 점수 | 학 별 | 점수 | 키와 몸무게 | 점수 |
|----|----------|------|----------------------------------|-----|-------------|-----|
| 1 | 100억원 이상 | 100점 | 서울대, 카이스트, 포항 공대, 미국 명문대 | 80점 | 185cm, 75kg | 60점 |
| 2 | 60~100억원 | 95점 | 연세대, 고려대, 미국 100위권 대학교 | 77점 | 183cm, 73kg | 58점 |
| 3 | 50~60억원 | 90점 | 서강대, 성균관대, 한양 대, 서울교대 | 74점 | 181cm, 71kg | 56점 |
| 4 | 40~50억원 | 85점 | 중앙대, 경희대, 외대, 시 립대, 해외 4년제 유학 | 71점 | 179cm, 69kg | 54점 |
| 5 | 30~40억원 | 80점 | 동국대, 건국대, 홍익대 | 68점 | 177cm, 67kg | 52점 |
| 6 | 20~30억원 | 75점 | 부산대, 경북대, 전남대 | 65점 | 175cm, 65kg | 50점 |
| 7 | 10~20억원 | 70점 | 기타 서울, 수도권 4년 제 | 62점 | 173cm, 63kg | 48점 |
| 8 | 5~10억원 | 65점 | 지방 광역 국립대, 해외 칼리지 | 59점 | 171cm, 61kg | 46점 |
| 9 | 3~5억원 | 60점 | 지방 4년제 | 56점 | 169cm, 59kg | 44점 |
| 10 | 3억원 이하 | 55점 | 2년제 대학 | 53점 | 167cm, 57kg | 42점 |

* 비교 : 외모에서 얼굴과 몸매는 기본적으로 '무난'해야 함.

* 대머리 -10점, 보기심한 흉터 -5점, 키 190cm 이상이거나 167cm 미만은 비율대로 2cm당 -5점, 몸무게 5kg 증감시 -5점

선형 모델의 새로운 의미: 특성을 변화 및 압축

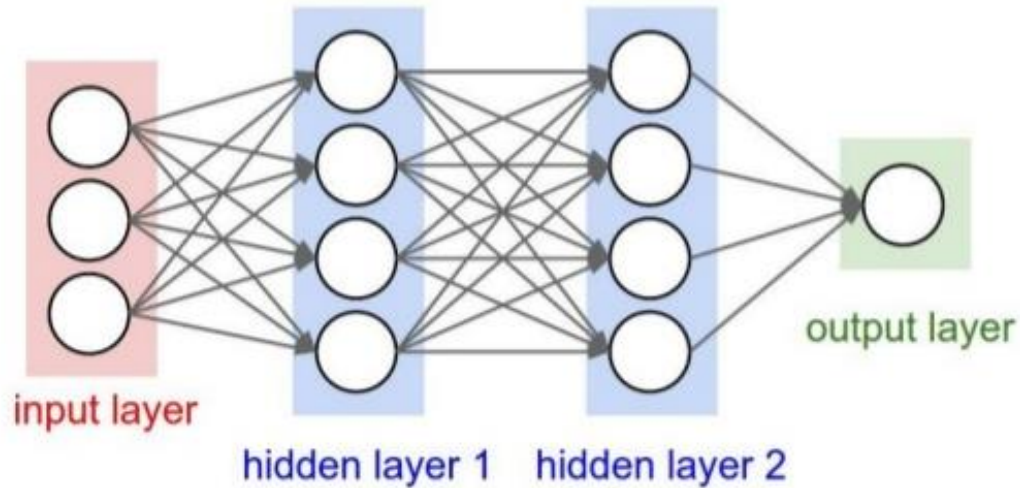


Dense Layer(Fully Connected Layer)

MLP, Fully Connected Layer, Dense Layer 모두 같은 뜻

한 Layer에 Perceptron을 여러 개 넣고, 다른 Layer와 모두 연결하여 네트워크를 구성

Neural Network라고 부름



소스 코드

```
model = Sequential([  
    Dense(4, input_shape=[3]),  
    Dense(4),  
    Dense(1),  
])
```

Dense Layer 실습(1)

- 딥러닝 실습 순서

1. import: 필요한 모듈 import
2. 전처리: 학습에 필요한 데이터 전처리 수행
3. 모델링(model): 모델 정의(모델이 어떻게 생겼는지)
4. 컴파일(compile): 모델 생성(모델의 학습에 필요한 세팅)
5. 학습(fit): 모델을 학습 (데이터를 넣고, 세팅된 상황에 맞게 학습)

Dense Layer 실습(2)

```
# 필요한 패키지 import
import numpy as np

# STEP 1: 필요한 모듈 import
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

# STEP 2: 데이터 전처리
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([5.0, 6.0, 7.0, 8.0, 9.0, 10.0], dtype=float)

# STEP 3: 모델의 정의 (modeling)
model = Sequential([
    Dense(1, input_shape=[1]),
])

# STEP 4: 모델의 생성 (compile)
model.compile(optimizer='sgd', loss='mse')

# STEP 5: 학습 (fit)
model.fit(xs, ys, epochs=1200, verbose=0)

# 검증
# 16.000046
model.predict([10.0])
```

모델 정의 (Sequential)

tensorflow 2.0은 keras의 Sequential 방식과 동일하게 블록쌓기 방식으로 매우 쉽게 모델링을 할 수 있습니다.

Dense Layer는 가장 기본적인 신경망 층이라고 이해하시면 됩니다.

Dense == **Fully Connected Layer** 라고도 불리웁니다.

Sequential()은 마치 레고 블록을 쌓듯이 layer를 순서대로 쌓아주면 됩니다.

- 학습 순서는 위에서 아래방향으로 진행됩니다.
- 반드시, 첫번째 layer에는 `input_shape` 을 지정해 주어야 합니다.

컴파일(compile): 적절한 optimizer와 loss 선정하기

우리는 적절한 회귀 값을 예측하는 모델을 만들어야 합니다.

regression(회귀) 예측을 위해서는 `loss=mse` 를 선택합니다.

`optimizer` 와 `loss` 를 지정합니다.

- `optimizer` 는 'sgd'(Stochastic Gradient Descent)를 지정합니다.
- `loss` 는 'mse'를 지정합니다.

학습 (fit)

이제, 모델에 학습을 진행해야합니다. `fit()` 메소드를 활용하여 학습을 진행합니다.

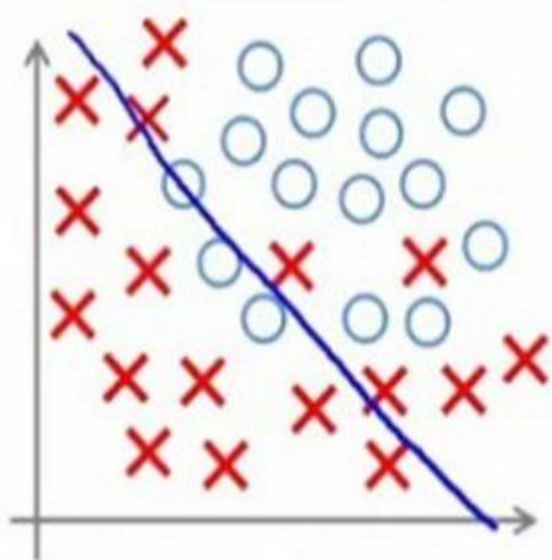
학습시, `feature`, `label` 값 지정 그리고 `epochs` 을 지정합니다.

Dense Layer 실습(3): Training 과 Validation

```
Epoch 1/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.2259 - acc: 0.9304
Epoch 2/30
60000/60000 [=====] - 4s 66us/sample - loss: 0.1004 - acc: 0.9704
Epoch 3/30
60000/60000 [=====] - 4s 66us/sample - loss: 0.0736 - acc: 0.9787
Epoch 4/30
60000/60000 [=====] - 4s 67us/sample - loss: 0.0563 - acc: 0.9832
Epoch 5/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.0467 - acc: 0.9864
Epoch 6/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.0404 - acc: 0.9882
Epoch 7/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.0329 - acc: 0.9900
Epoch 8/30
60000/60000 [=====] - 4s 66us/sample - loss: 0.0307 - acc: 0.9908
Epoch 9/30
60000/60000 [=====] - 4s 67us/sample - loss: 0.0262 - acc: 0.9922
Epoch 10/30
60000/60000 [=====] - 4s 67us/sample - loss: 0.0238 - acc: 0.9930
Epoch 11/30
60000/60000 [=====] - 4s 66us/sample - loss: 0.0217 - acc: 0.9936
Epoch 12/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.0214 - acc: 0.9943
Epoch 13/30
60000/60000 [=====] - 4s 69us/sample - loss: 0.0209 - acc: 0.9938
Epoch 14/30
60000/60000 [=====] - 4s 66us/sample - loss: 0.0188 - acc: 0.9946
Epoch 15/30
60000/60000 [=====] - 4s 66us/sample - loss: 0.0156 - acc: 0.9958
```

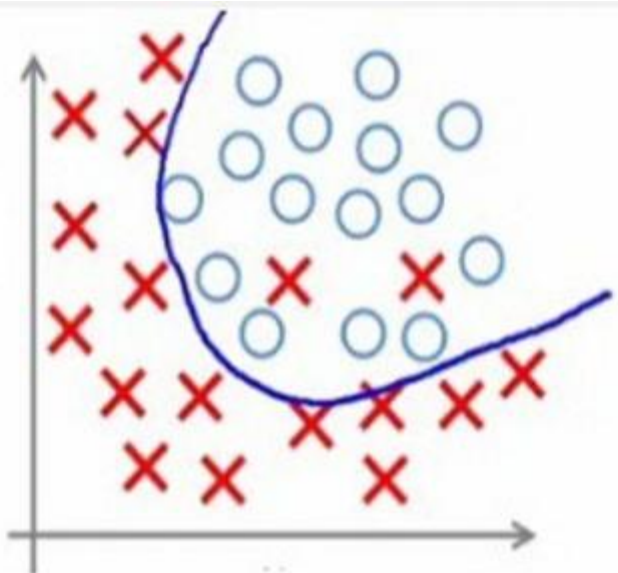
```
Epoch 16/30
60000/60000 [=====] - 4s 69us/sample - loss: 0.0154 - acc: 0.9958
Epoch 17/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.0133 - acc: 0.9966
Epoch 18/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.0159 - acc: 0.9961
Epoch 19/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.0136 - acc: 0.9964
Epoch 20/30
60000/60000 [=====] - 4s 69us/sample - loss: 0.0123 - acc: 0.9966
Epoch 21/30
60000/60000 [=====] - 4s 67us/sample - loss: 0.0114 - acc: 0.9969
Epoch 22/30
60000/60000 [=====] - 4s 66us/sample - loss: 0.0130 - acc: 0.9966
Epoch 23/30
60000/60000 [=====] - 4s 66us/sample - loss: 0.0118 - acc: 0.9968
Epoch 24/30
60000/60000 [=====] - 4s 67us/sample - loss: 0.0115 - acc: 0.9974
Epoch 25/30
60000/60000 [=====] - 4s 67us/sample - loss: 0.0114 - acc: 0.9975
Epoch 26/30
60000/60000 [=====] - 4s 69us/sample - loss: 0.0121 - acc: 0.9971
Epoch 27/30
60000/60000 [=====] - 4s 68us/sample - loss: 0.0105 - acc: 0.9976
Epoch 28/30
60000/60000 [=====] - 4s 69us/sample - loss: 0.0078 - acc: 0.9983
Epoch 29/30
60000/60000 [=====] - 4s 69us/sample - loss: 0.0109 - acc: 0.9973
Epoch 30/30
60000/60000 [=====] - 4s 67us/sample - loss: 0.0130 - acc: 0.9969
```

과소 적합과 과대 적합

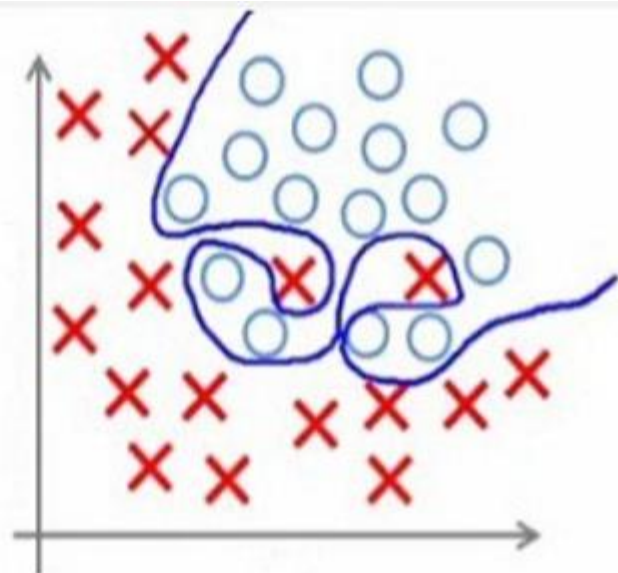


Under-fitting

(too simple to
explain the
variance)



Appropriate-fitting

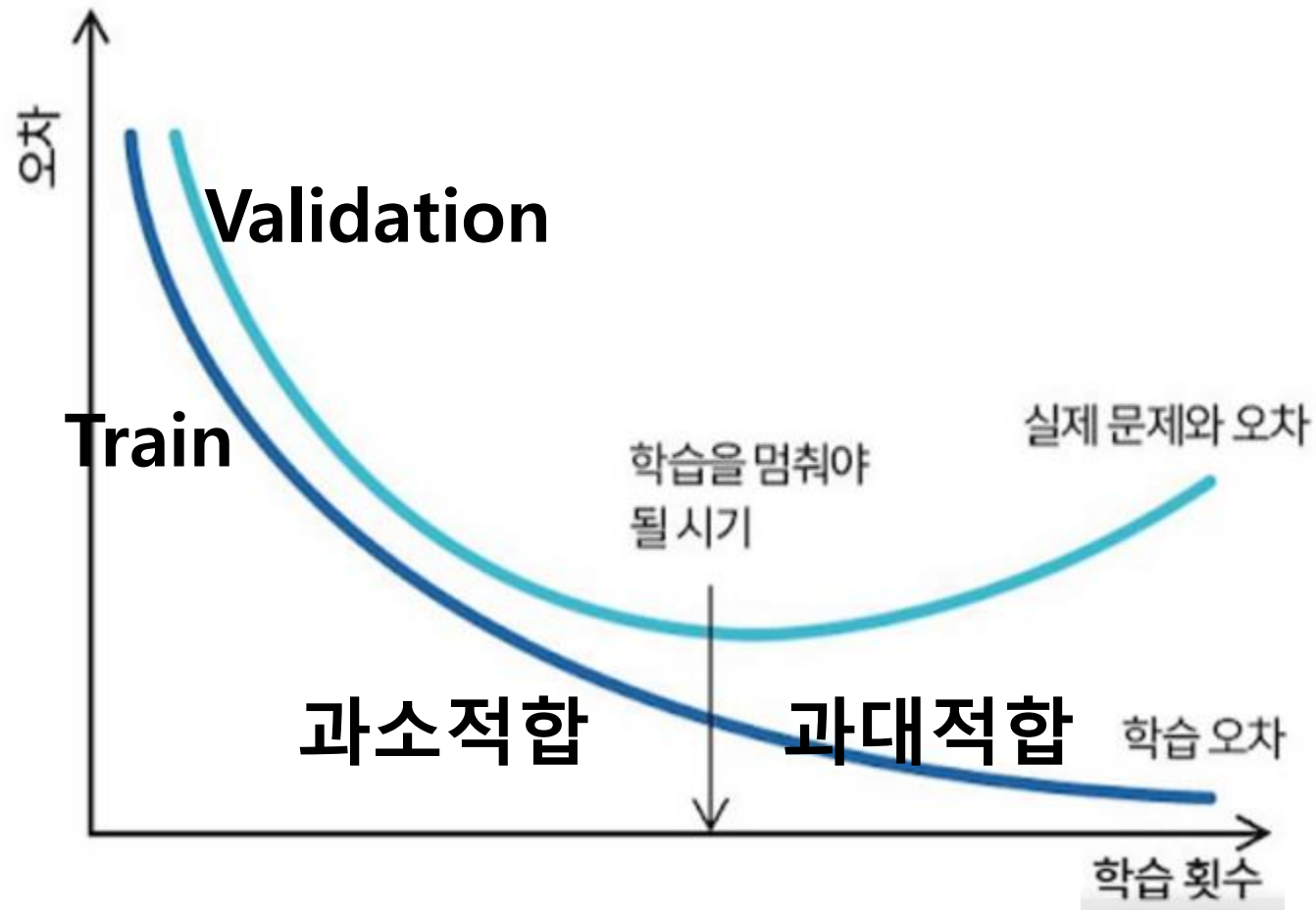


Over-fitting

(forcefitting -- too
good to be true)

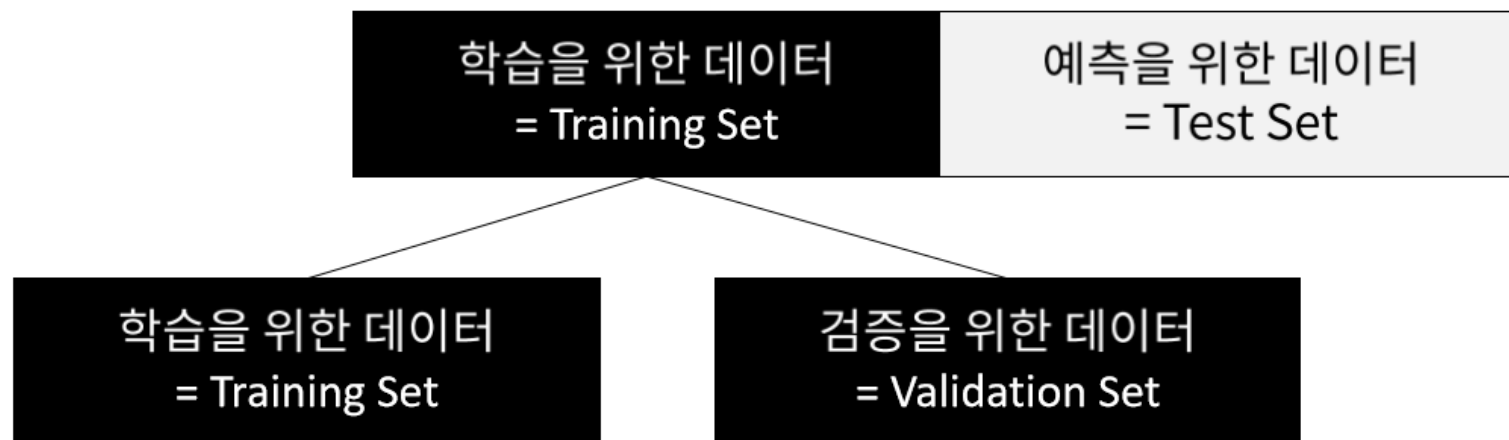
과소적합과 과대적합을 피하고 적절한 학습 포인트 찾기(1)

데이터를 분리해서, 학습과 검증 과정에 활용하는 방법으로 진행합니다.

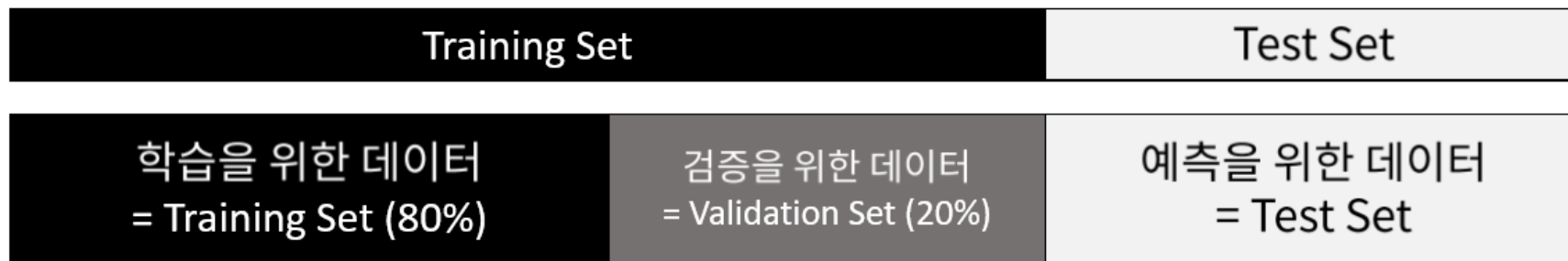


과소적합과 과대적합을 피하고 적절한 학습 포인트 찾기(2)

데이터를 분리해서, 학습과 검증 과정에 활용하는 방법으로 진행합니다.



Train / Validation 세트 구성 예시



활성화 함수 정리

원hat(o): categorical_crossentropy
원hat(x): sparse_categorical_crossentropy

마지막 출력층

Loss

Dense(**1**, activation='sigmoid')

loss='binary_crossentropy'

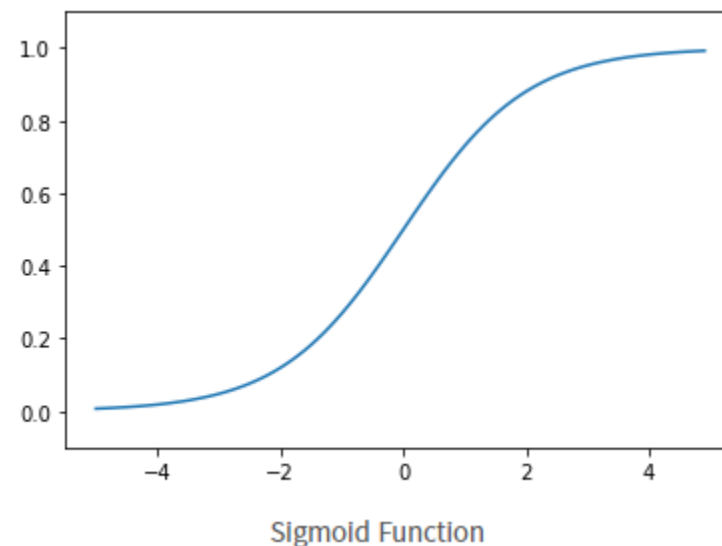
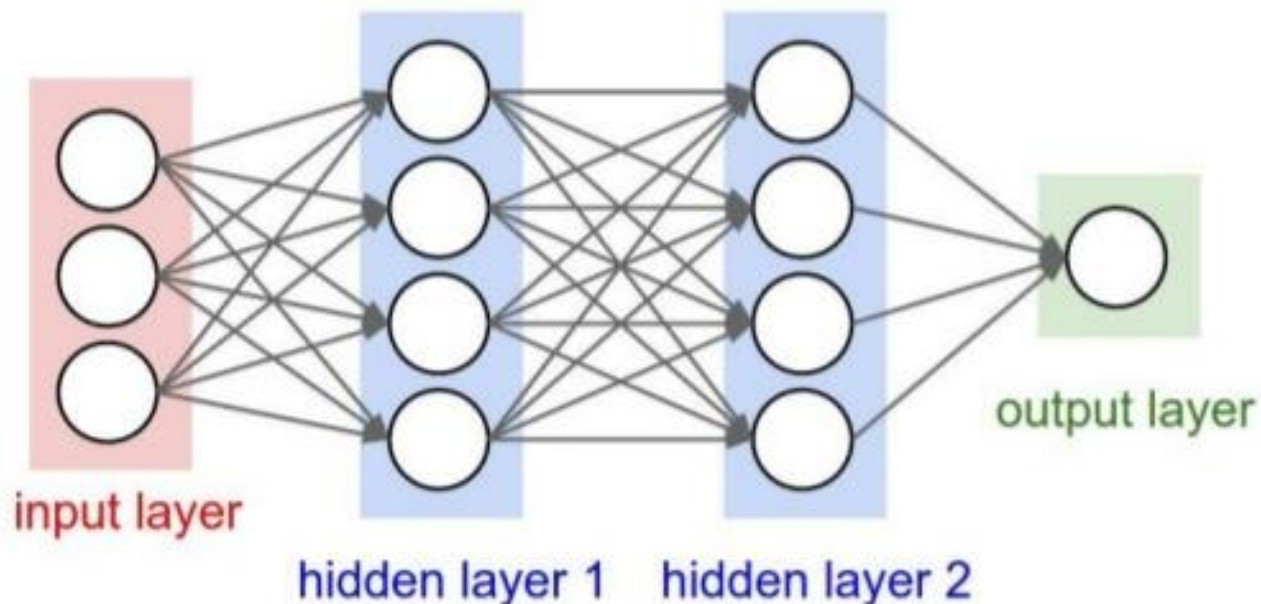
Dense(**2 이상**, activation='softmax')

loss='categorical_crossentropy'
loss='sparse_categorical_crossentropy'

| Loss function | Usage | Examples | |
|--------------------------------|---------------------------|---|--------------------------------------|
| | | Using probabilities | Using logits |
| | | <i>from_logits=False</i> | <i>from_logits=True</i> |
| BinaryCrossentropy | Binary classification | y_true: 1 y_pred: 0.69 | y_true: 1 y_pred: 0.8 |
| CategoricalCrossentropy | Multiclass classification | y_true: 0 0 1 y_pred: 0.30 0.15 0.55 | y_true: 0 0 1 y_pred: 1.5 0.8 2.1 |
| Sparse CategoricalCrossentropy | Multiclass classification | y_true: 2 y_pred: 0.30 0.15 0.55 | y_true: 2 y_pred: 1.5 0.8 2.1 |

활성화 함수 : Output이 Dense(1) 인 경우

Output Layer의 node가 1개인 경우에는 분류 문제에서는 Sigmoid 함수를 활성화 함수로 사용합니다.
Output Layer의 node가 1개인 경우에는 분류 문제에서는 Yes/No 문제만 풀 수 있습니다.
회귀 문제에서는 활성화 함수가 필요 없습니다.



활성화 함수: softmax

활성화 함수 출력 값을 각 카테고리의 확률값으로 변화하는 방법

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{i=1}^K e^{z_i}}$$

$1 \leq j \leq K$

K 차원 벡터

The diagram illustrates the softmax function's operation. On the left, a column vector $\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$ is shown. A blue arrow points from this vector to a yellow rounded rectangle labeled "softmax". Another blue arrow points from the "softmax" box to the resulting probability vector $\begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$.

Binary Cross Entropy

이진 분류를 풀기 위해서 만든 Loss Function

EX) 개와 고양이 분류에서 $y=1$ (개인 경우), $y=0$ (고양이인 경우), P (개일 확률)

기존 오차의 개념

$$E = \frac{1}{2} \sum_k (y_k - t_k)^2$$

예측 값과 실제 값의 차이가 클 수록 증가

Binary Cross Entropy

$$L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p))$$

마이너스 값이 되는 것 방지

딥러닝 실습 (1)

가장 대표적인 Iris 데이터의 분류 문제를 딥러닝을 이용해서 풀어보겠습니다.
총 3개의 꽃의 형태에 대한 분류이므로, 마지막 Layer는 Dense(3)이 되어야 합니다.



Iris Versicolor



Iris Setosa



Iris Virginica

원hat(o): categorical_crossentropy
원hat(x): sparse_categorical_crossentropy

마지막 출력층

Loss

Dense(1, activation='sigmoid')

loss='binary_crossentropy'

Dense(2 이상, activation='softmax')

loss='categorical_crossentropy'
loss='sparse_categorical_crossentropy'

딥러닝 실습 (2)

필요한 모듈을 import 합니다.

1. **import: 필요한 모듈 import**

2. 전처리: 학습에 필요한 데이터 전처리 수행
3. 모델링(model): 모델 정의
4. 컴파일(compile): 모델 생성
5. 학습(fit): 모델을 학습

```
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
```

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import ModelCheckpoint
```

딥러닝 실습 (3)

Train 데이터와 validation 데이터를 80대 20으로 분할

참고 <https://www.tensorflow.org/datasets/catalog/iris?hl=ko>

| Split | Examples |
|---------|----------|
| 'train' | 150 |

• 기능 구조:

```
FeaturesDict({  
  'features': Tensor(shape=(4,), dtype=float32),  
  'label': ClassLabel(shape=(), dtype=int64, num_classes=3),  
})
```

1. import: 필요한 모듈 import
2. 전처리: 학습에 필요한 데이터 전처리 수행
3. 모델링(model): 모델 정의
4. 컴파일(compile): 모델 생성
5. 학습(fit): 모델을 학습

```
train_dataset = tfds.load('iris', split='train[:80%]')  
valid_dataset = tfds.load('iris', split='train[80%:]')
```

딥러닝 실습 (3)

Train 데이터와 validation 데이터를 80대 20으로 분할

참고 <https://www.tensorflow.org/datasets/catalog/iris?hl=ko>

| Split | Examples |
|---------|----------|
| 'train' | 150 |

• 기능 구조:

```
FeaturesDict({  
  'features': Tensor(shape=(4,), dtype=float32),  
  'label': ClassLabel(shape=(), dtype=int64, num_classes=3),  
})
```

1. import: 필요한 모듈 import
2. 전처리: 학습에 필요한 데이터 전처리 수행
3. 모델링(model): 모델 정의
4. 컴파일(compile): 모델 생성
5. 학습(fit): 모델을 학습

```
train_dataset = tfds.load('iris', split='train[:80%]')  
valid_dataset = tfds.load('iris', split='train[80%:]')
```


딥러닝 실습 (4)

데이터 형태 확인

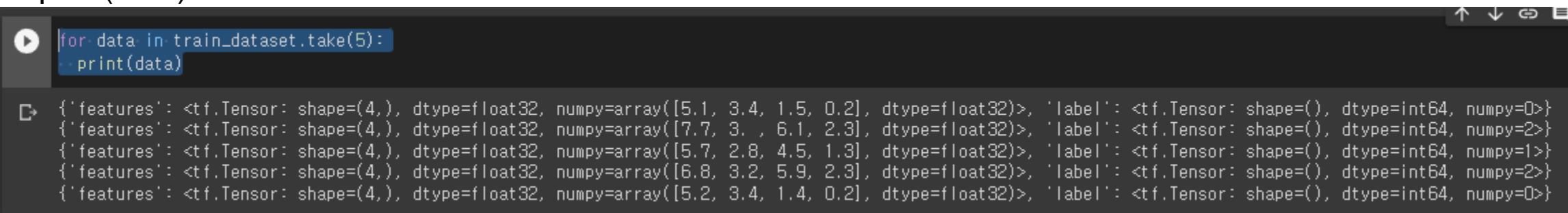
```
for data in train_dataset:  
    print(data)
```

일단 데이터를 한번 찍어 보겠습니다.

```
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.1, 3.4, 1.5, 0.2], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=0>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([7.7, 3. , 6.1, 2.3], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=2>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.7, 2.8, 4.5, 1.3], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=1>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([6.8, 3.2, 5.9, 2.3], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=2>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.2, 3.4, 1.4, 0.2], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=0>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.6, 2.9, 3.6, 1.3], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=1>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.5, 2.6, 4.4, 1.2], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=1>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.5, 2.4, 3.7, 1. ], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=1>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([4.6, 3.4, 1.4, 0.3], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=0>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([7.7, 2.8, 6.7, 2. ], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=2>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([7. , 3.2, 4.7, 1.4], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=1>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([4.6, 3.2, 1.4, 0.2], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=0>}
```

```
for data in train_dataset.take(5):  
    print(data)
```

너무 많으니 5개만 출력!



```
for data in train_dataset.take(5):  
    print(data)
```

```
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.1, 3.4, 1.5, 0.2], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=0>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([7.7, 3. , 6.1, 2.3], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=2>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.7, 2.8, 4.5, 1.3], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=1>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([6.8, 3.2, 5.9, 2.3], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=2>}  
{'features': <tf.Tensor: shape=(4,), dtype=float32, numpy=array([5.2, 3.4, 1.4, 0.2], dtype=float32)>, 'label': <tf.Tensor: shape=(), dtype=int64, numpy=0>}
```

딥러닝 실습 (5)

Feature(x)와 Label(y)로 데이터 분리
Label을 one hot encoding으로 변환

```
for data in train_dataset.take(5):  
    x = data['features']  
    y = data['label']  
    y = tf.one_hot(y,3)  
    print(x)  
    print(y)
```

```
tf.Tensor([5.1 3.4 1.5 0.2], shape=(4,), dtype=float32)  
tf.Tensor([1. 0. 0.], shape=(3,), dtype=float32)  
tf.Tensor([7.7 3. 6.1 2.3], shape=(4,), dtype=float32)  
tf.Tensor([0. 0. 1.], shape=(3,), dtype=float32)  
tf.Tensor([5.7 2.8 4.5 1.3], shape=(4,), dtype=float32)  
tf.Tensor([0. 1. 0.], shape=(3,), dtype=float32)  
tf.Tensor([6.8 3.2 5.9 2.3], shape=(4,), dtype=float32)  
tf.Tensor([0. 0. 1.], shape=(3,), dtype=float32)  
tf.Tensor([5.2 3.4 1.4 0.2], shape=(4,), dtype=float32)  
tf.Tensor([1. 0. 0.], shape=(3,), dtype=float32)
```

데이터 분할 함수 만들기

```
def preprocess(data):  
    x = data['features']  
    y = data['label']  
    y = tf.one_hot(y, 3)  
    return x, y
```

딥러닝 실습 (6)

Feature(x)와 Label(y)로 데이터 분리
Label을 one hot encoding으로 변환

1. import: 필요한 모듈 import
2. 전처리: 학습에 필요한 데이터 전처리 수행
3. 모델링(model): 모델 정의
4. 컴파일(compile): 모델 생성
5. 학습(fit): 모델을 학습

`batch_size=10`

`train_data = train_dataset.map(preprocess).batch(batch_size)`

`valid_data = valid_dataset.map(preprocess).batch(batch_size)`

배치 사이즈(1)

데이터 셋을 미니 배치로 나누어서 학습
가중치 업데이트는 한 배치가 끝났을 때 진행됩니다.

10,000개의 데이터를 배치 사이즈를 10으로 진행시, 배치는 1,000개, 배치 사이즈가 1,000으로 진행시 10개 배치
업데이트를 1,000번(배치가 1000), 업데이트를 10번 진행 (배치가 10)



1 Epoch : 모든 데이터 셋을 한 번 학습

1 iteration : 1회 학습

minibatch : 데이터 셋을 batch size 크기로 쪼개서 학습

ex) 총 데이터가 100개, batch size가 10이면,

1 iteration = 10개 데이터에 대해서 학습

1 Epoch = $100 / \text{batch size} = 10$ iteration



속도 느림.

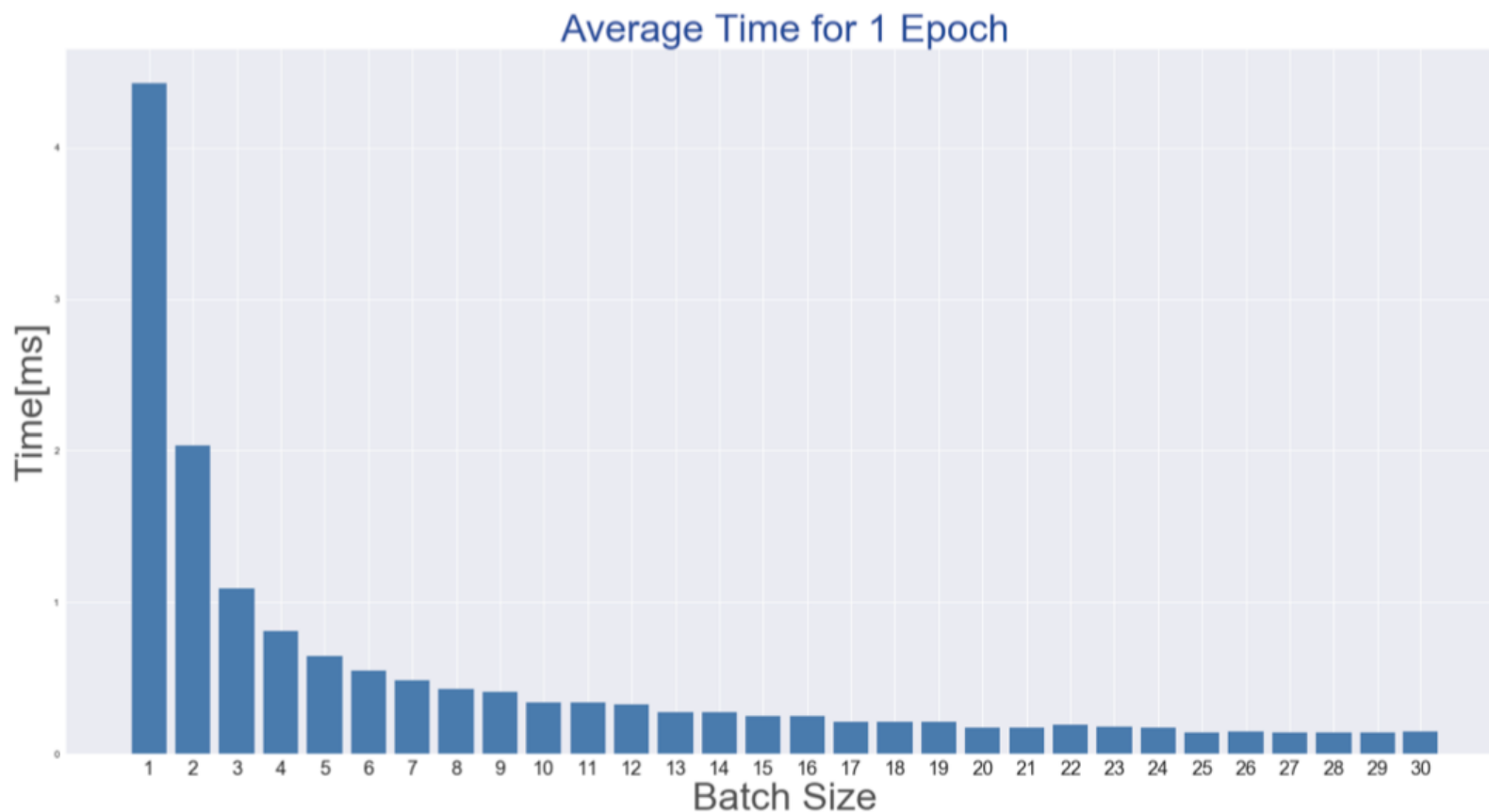


배치 사이즈(2)

배치 사이즈가 작으면, 업데이트를 천천히 진행합니다.

잘못된 방향으로 업데이트 시에는 수정이 어렵습니다.

배치가 클 경우에는 가중치 업데이트 변화량이 크고, 최종 목적지에 도달 할 수 없을 수도 있습니다.



딥러닝 실습 (7)

Sequential 모델 안에서 층을 깊게 쌓아 올리면 됩니다.

- `Input_shape`은 iris 꽃 데이터셋의 x의 feature 개수가 4이므로, (4,)로 지정
- 깊은 출력층과 더 많은 Layer를 쌓습니다.
- Dense Layer에 `activation='relu'`를 적용합니다.
- 분류의 마지막 출력 층의 출력 숫자는 분류하고자 하는 클래스의 개수와 같아야 합니다.

1. import: 필요한 모듈 import
2. 전처리: 학습에 필요한 데이터 전처리 수행
3. 모델링(model): 모델 정의
4. 컴파일(compile): 모델 생성
5. 학습(fit): 모델을 학습

```
model = Sequential([  
    # input_shape는 X의 feature 갯수가 4개 이므로 (4, )로 지정  
    Dense(512, activation='relu', input_shape=(4,)),  
    Dense(256, activation='relu'),  
    Dense(128, activation='relu'),  
    Dense(64, activation='relu'),  
    Dense(32, activation='relu'),  
    # Classification을 위한 Softmax, 클래스 갯수 = 3개  
    Dense(3, activation='softmax'),  
])
```

딥러닝 실습 (8)

모델을 살펴보겠습니다.
파라미터의 수를 어떻게 계산하는지 살펴보세요!
파라미터 수는 천천히 줄여 나가야 합니다.

```
model.summary()
```

| Layer (type) | Output Shape | Param # |
|-----------------|--------------|---------|
| dense (Dense) | (None, 512) | 2560 |
| dense_1 (Dense) | (None, 256) | 131328 |
| dense_2 (Dense) | (None, 128) | 32896 |
| dense_3 (Dense) | (None, 64) | 8256 |
| dense_4 (Dense) | (None, 32) | 2080 |
| dense_5 (Dense) | (None, 3) | 99 |

=====
Total params: 177,219
Trainable params: 177,219
Non-trainable params: 0
=====

```
print('첫번째 layer 파라미터 수', 512*4+512)  
print('두번째 layer 파라미터 수', 256*512+256)  
print('세번째 layer 파라미터 수', 128*256+128)  
print('네번째 layer 파라미터 수', 64*128+64)  
print('다섯번째 layer 파라미터 수', 32*64+32)  
print('여섯번째 layer 파라미터 수', 3*32+3)  
  
print('total prameter numbers', 512*4+512 + 256*512+256 + 128*256+128 + 64*128+64 + 32*64+32 + 3*32+3)
```

```
첫번째 layer 파라미터 수 2560  
두번째 layer 파라미터 수 131328  
세번째 layer 파라미터 수 32896  
네번째 layer 파라미터 수 8256  
다섯번째 layer 파라미터 수 2080  
여섯번째 layer 파라미터 수 99  
total prameter numbers 177219
```

딥러닝 실습 (9): 컴파일

컴파일

1. ``optimizer``는 가장 최적화가 잘되는 알고리즘인 **'adam'**을 사용합니다.
2. ``loss`` 설정
 - * 출력층 activation이 ``sigmoid`` 인 경우: ``binary_crossentropy``
 - * 출력층 activation이 ``softmax`` 인 경우:
 - * 원핫인코딩(O): **``categorical_crossentropy``**
 - * 원핫인코딩(X): ``sparse_categorical_crossentropy``
3. ``metrics``를 **'acc'** 혹은 **'accuracy'**로 지정하면, 학습시 정확도를 모니터링 할 수 있습니다.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```


딥러닝 실습 (10): ModelCheckpoint 생성

컴파일

`val_loss` 기준으로 epoch 마다 최적의 모델을 저장하기 위하여, ModelCheckpoint를 만듭니다.

* `checkpoint_path`는 모델이 저장될 파일 이름을 설정합니다.

* `ModelCheckpoint`을 선언하고, 적절한 옵션 값을 지정합니다.

```
checkpoint_path = "my_checkpoint.ckpt"
```

```
checkpoint = ModelCheckpoint(filepath=checkpoint_path,  
                             save_weights_only=True,  
                             save_best_only=True,  
                             monitor='val_loss',  
                             verbose=1)
```

딥러닝 실습 (11): 학습

validation_data를 반드시 지정합니다.

epochs을 적절하게 지정합니다.

callbacks에 바로 위에서 만든 checkpoint를 지정합니다.

```
history = model.fit(train_data,  
                    validation_data=(valid_data),  
                    epochs=20,  
                    callbacks=[checkpoint],  
                    )
```

딥러닝 실습 (12): Load Weights(ModelCheckpoint)

학습이 완료된 후에는 반드시 `load_weights`를 해주어야 합니다.

그렇지 않으면, 열심히 ModelCheckpoint를 만든 의미가 없습니다.

checkpoint 를 저장한 파일명을 입력합니다.

```
model.load_weights(checkpoint_path)
```

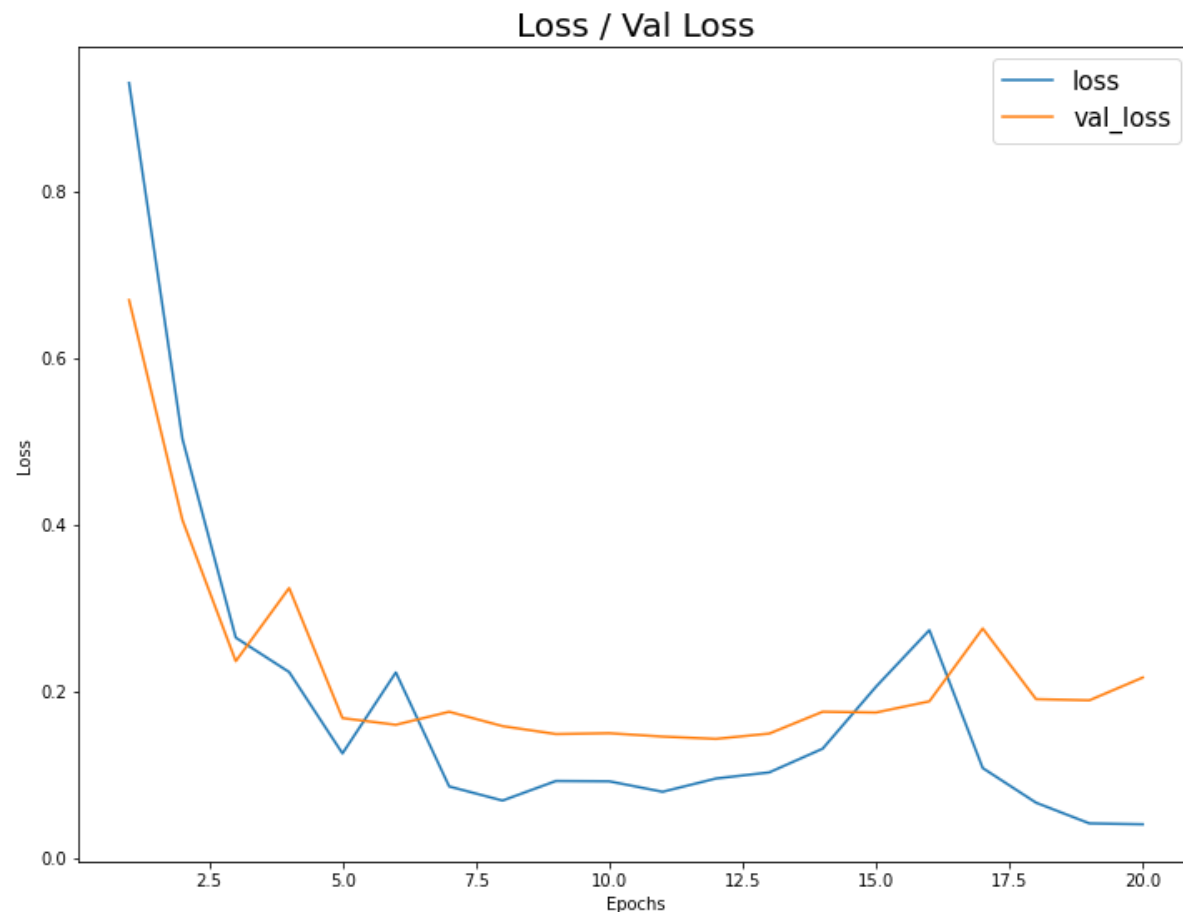
```
[24] # checkpoint 를 저장한 파일명을 입력합니다.  
      model.load_weights(checkpoint_path)
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7ff2924a2b80>
```

딥러닝 실습 (13): 학습에 대한 Loss 시각화

```
import matplotlib.pyplot as plt
```

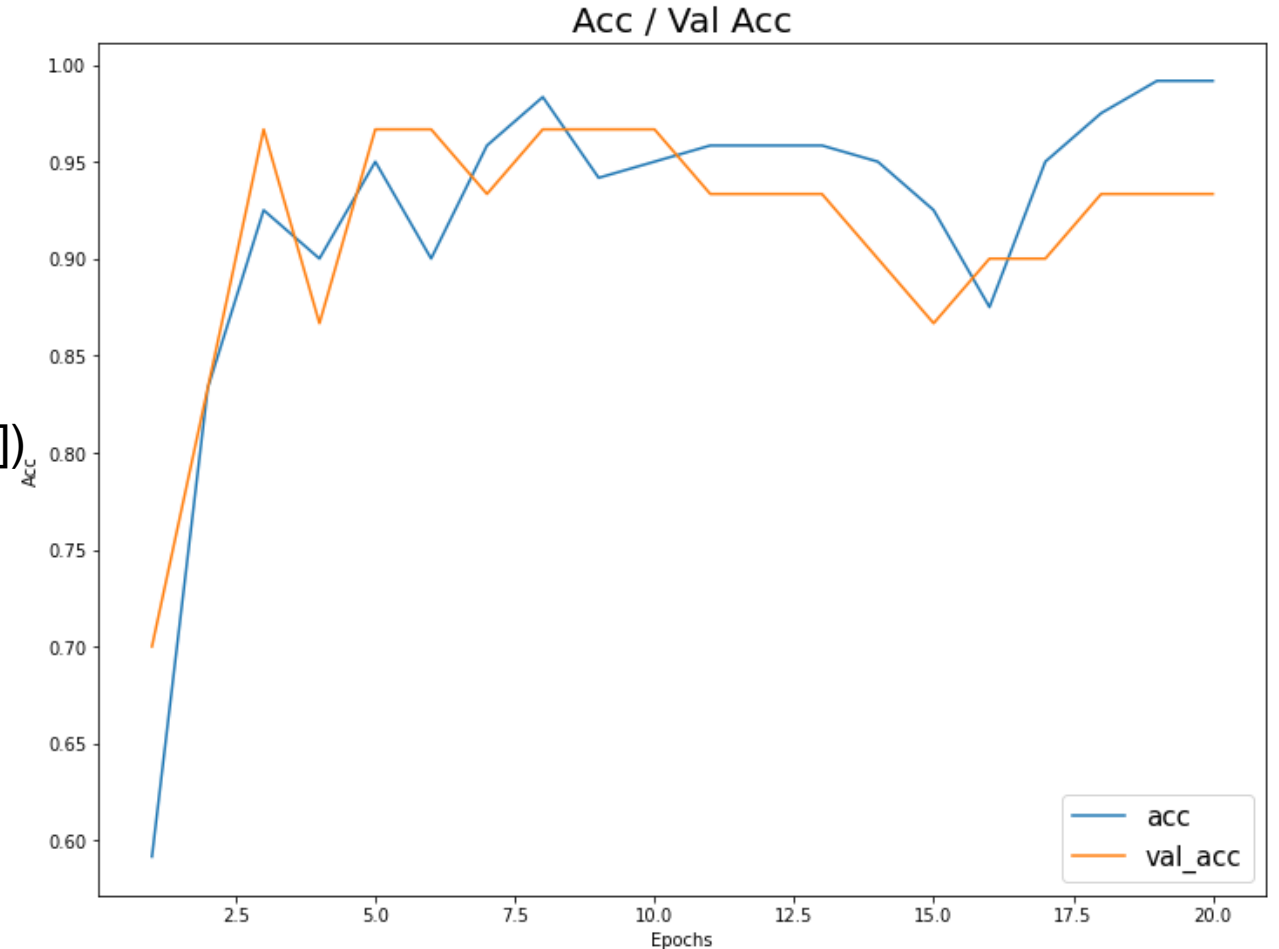
```
plt.figure(figsize=(12, 9))  
plt.plot(np.arange(1, 21), history.history['loss'])  
plt.plot(np.arange(1, 21), history.history['val_loss'])  
plt.title('Loss / Val Loss', fontsize=20)  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.legend(['loss', 'val_loss'], fontsize=15)  
plt.show()
```



딥러닝 실습 (14): 학습에 대한 Accuracy 시각화

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(12, 9))  
plt.plot(np.arange(1, 21), history.history['acc'])  
plt.plot(np.arange(1, 21), history.history['val_acc'])  
plt.title('Acc / Val Acc', fontsize=20)  
plt.xlabel('Epochs')  
plt.ylabel('Acc')  
plt.legend(['acc', 'val_acc'], fontsize=15)  
plt.show()
```



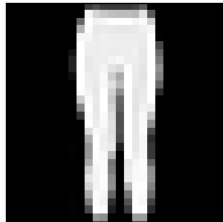
숙제: 여기까지 코딩 후, jupyter notebook 파일 다운로드 후 제출합니다.

Image 데이터를 이용한 실습

https://www.tensorflow.org/datasets/catalog/fashion_mnist?hl=en



Pullover (2)



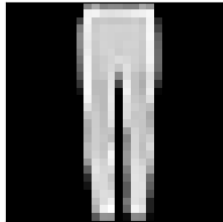
Trouser (1)



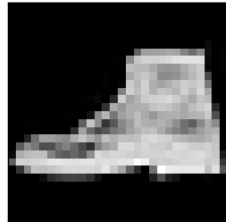
Bag (8)



Coat (4)



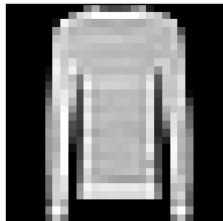
Trouser (1)



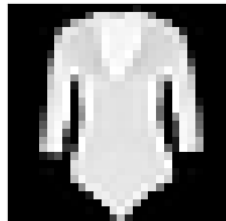
Ankle boot (9)



Pullover (2)



Pullover (2)



T-shirt/top (0)

- Feature structure:

```
FeaturesDict({  
  'image': Image(shape=(28, 28, 1), dtype=uint8),  
  'label': ClassLabel(shape=(), dtype=int64, num_classes=10),  
})
```

- Feature documentation:

| Feature | Class | Shape | Dtype | Description |
|--------------|------------|-------------|-------|-------------|
| FeaturesDict | | | | |
| image | Image | (28, 28, 1) | uint8 | |
| label | ClassLabel | | int64 | |

1. 필요한 모듈 import

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
```

```
from tensorflow.keras.layers import Dense, Flatten
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.callbacks import ModelCheckpoint
```

2. 데이터 불러오기

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

```
(x_train, y_train), (x_valid, y_valid) = fashion_mnist.load_data()
```

```
#테스트
```

```
x_train.shape, x_valid.shape
```

```
y_train.shape, y_valid.shape
```

```
#정규화
```

```
x_train = x_train / 255.0
```

```
x_valid = x_valid / 255.0
```


2. 데이터 불러오기:Visualization

시각화

```
fig, axes = plt.subplots(2, 5)
```

```
fig.set_size_inches(10, 5)
```

```
for i in range(10):
```

```
    axes[i//5, i%5].imshow(x_train[i], cmap='gray')
```

```
    axes[i//5, i%5].set_title(str(y_train[i]), fontsize=15)
```

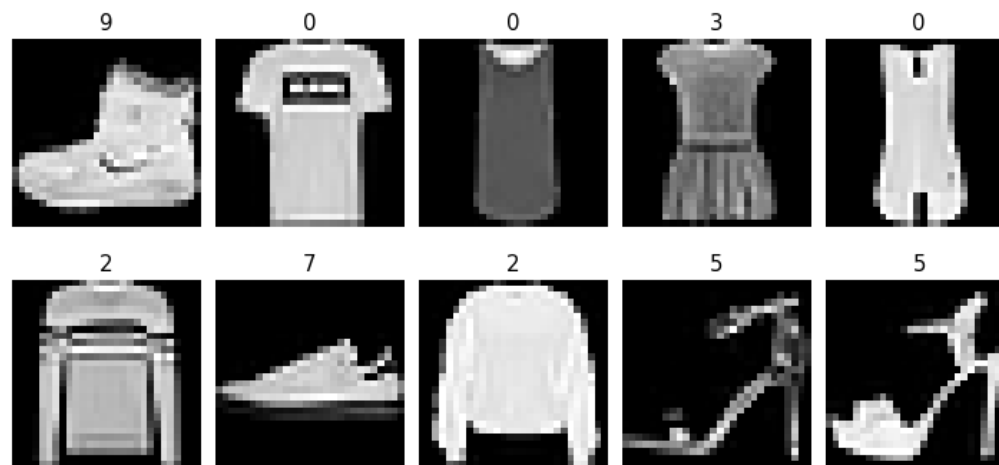
```
    plt.setp( axes[i//5, i%5].get_xticklabels(), visible=False)
```

```
    plt.setp( axes[i//5, i%5].get_yticklabels(), visible=False)
```

```
    axes[i//5, i%5].axis('off')
```

```
plt.tight_layout()
```

```
plt.show()
```



3. 모델링: Flatten Layer

- 고차원을 1D로 변환하여 “Dense Layer에 전달”해 주기 위하여 사용합니다.
- 28 X 28 의 “2D”로 되어 있는 이미지를 784로 “1D로 펼쳐 주는 작업”입니다.

```
x = Flatten(input_shape=(28, 28))
```

```
model = Sequential([  
    # Flatten으로 shape 펼치기  
    Flatten(input_shape=(28, 28)),  
    # Dense Layer  
    Dense(1024, activation='relu'),  
    Dense(512, activation='relu'),  
    Dense(256, activation='relu'),  
    Dense(128, activation='relu'),  
    Dense(64, activation='relu'),  
    # Classification을 위한 Softmax  
    Dense(10, activation='softmax'),  
])
```

```
model.summary()
```

| Layer (type) | Output Shape | Param # |
|-------------------|--------------|---------|
| flatten (Flatten) | (None, 784) | 0 |
| dense (Dense) | (None, 1024) | 803840 |
| dense_1 (Dense) | (None, 512) | 524800 |
| dense_2 (Dense) | (None, 256) | 131328 |
| dense_3 (Dense) | (None, 128) | 32896 |
| dense_4 (Dense) | (None, 64) | 8256 |
| dense_5 (Dense) | (None, 10) | 650 |

```
Total params: 1,501,770  
Trainable params: 1,501,770  
Non-trainable params: 0
```

3. 컴파일

- 1. `optimizer`는 가장 최적화가 잘되는 알고리즘인 'adam'을 사용합니다.
- 2. `loss` 설정
 - * 출력층 activation이 `sigmoid` 인 경우: `binary_crossentropy`
 - * 출력층 activation이 `softmax` 인 경우:
 - * 원핫인코딩(O): `categorical_crossentropy`
 - * 원핫인코딩(X): `sparse_categorical_crossentropy`
- 3. `metrics`를 'acc' 혹은 'accuracy'로 지정하면, 학습시 정확도를 모니터링 할 수 있습니다.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])
```

3. 컴파일: 체크 포인트 설정

```
checkpoint_path = "my_checkpoint.ckpt"  
checkpoint = ModelCheckpoint(filepath=checkpoint_path,  
                             save_weights_only=True,  
                             save_best_only=True,  
                             monitor='val_loss',  
                             verbose=1)
```

4. 컴파일: 학습

```
history = model.fit(x_train, y_train,  
                    validation_data=(x_valid, y_valid),  
                    epochs=20,  
                    callbacks=[checkpoint],  
                    )
```

checkpoint 를 저장한 파일명을 입력합니다.
model.load_weights(checkpoint_path)

