

# 심층 신경망

DNN (Deep Neural Networks)



# 학습 목표

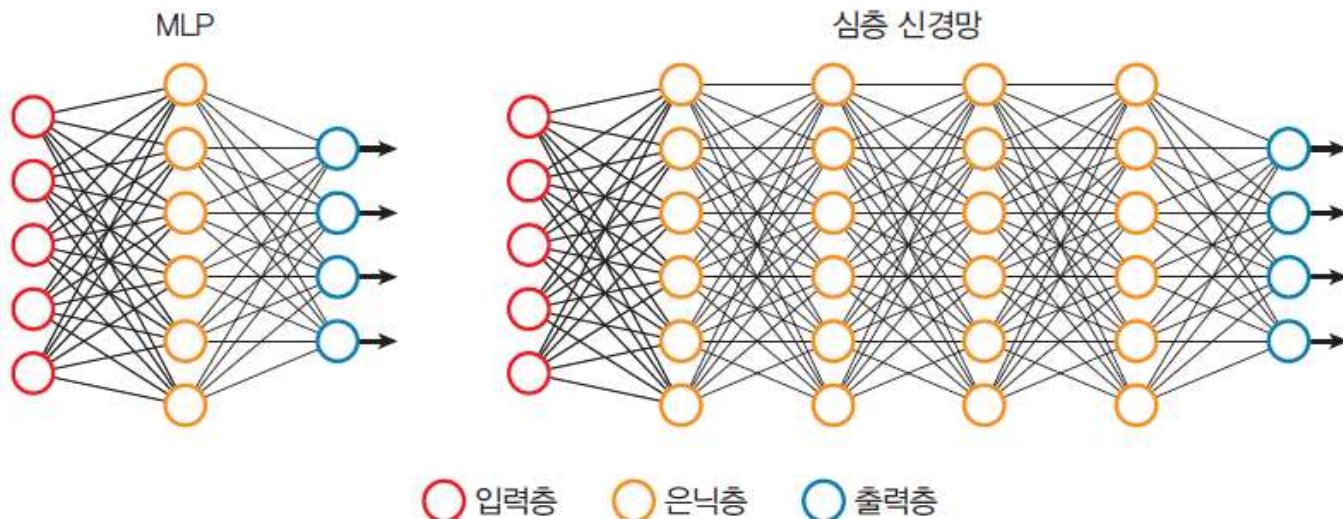
- **심층 신경망**의 구조를 이해한다.
- 은닉층의 역할을 이해한다.
- **그래디언트 소실 문제**를 이해한다.
- 새로 등장한 여러 가지 활성화 함수를 이해한다.
- 다양한 손실 함수를 이해한다.





# 심층신경망 (DNN)

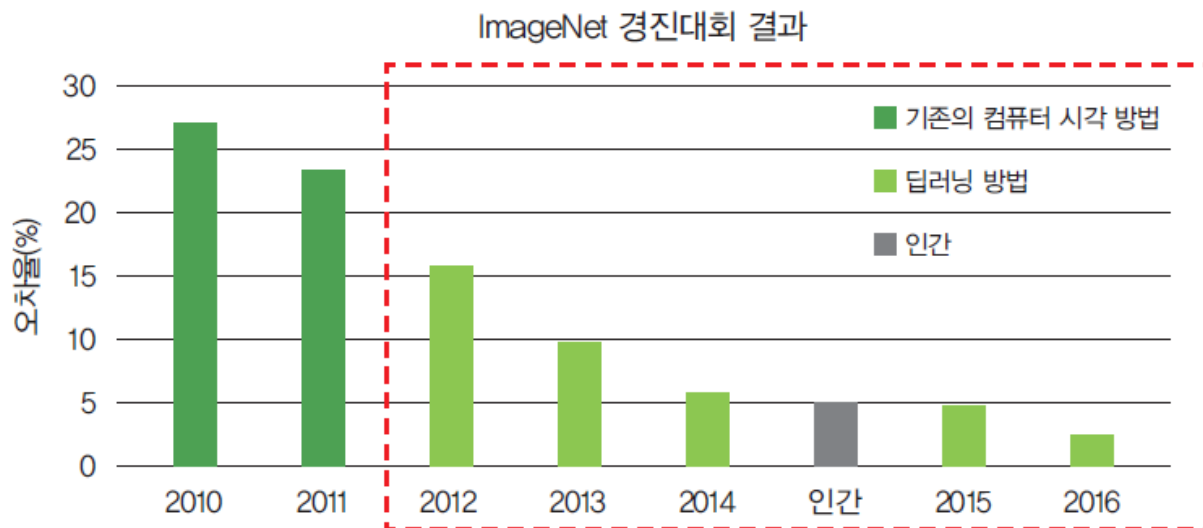
- 심층 신경망(DNN: Deep Neural Networks)은 MLP(다층 퍼셉트론)에서 은닉층의 개수를 증가시킨 것이다.
- 은닉층을 하나만 사용하는 것이 아니고 여러 개를 사용한다.
- 최근에 딥러닝은 컴퓨터 시각, 음성 인식, 자연어 처리, 소셜 네트워크 필터링, 기계 번역 등에 적용되어서 인간 전문가에 필적하는 결과를 얻고 있다.





# MLP의 문제점 해결

- 은닉층이 많아지면 출력층에서 계산된 그래디언트가 역전파되다가 값이 점점 작아져서 없어지는 문제점 -> 해결
- 훈련 데이터가 충분하지 못하면, 과잉 적합(over fitting)이 될 가능성도 높아진다. -> 해결
- 2012년에는 AlexNet이 다른 머신러닝 방법들을 큰 차이로 물리치고 ImageNet 경진대회에서 우승한다.

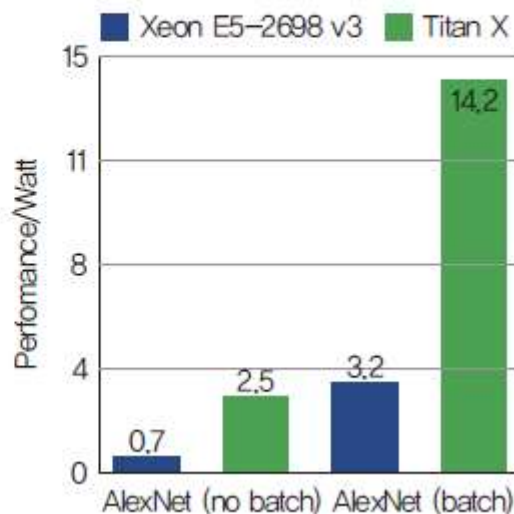
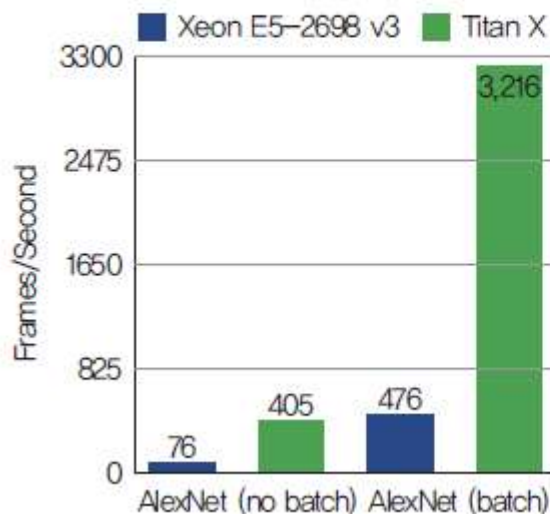




# GPU의 도움

- DNN의 학습 속도는 상당히 느리고 계산 집약적이기 때문에 학습에 시간과 자원이 많이 소모되었다.
- 최근 GPU(Graphic Processor Unit) 기술이 엄청나게 발전하면서 GPU가 제공하는 데이터 처리 기능을 딥러닝도 사용할 수 있게 되었다. 딥러닝 혁명에는 게이머들의 도움도 컸다.

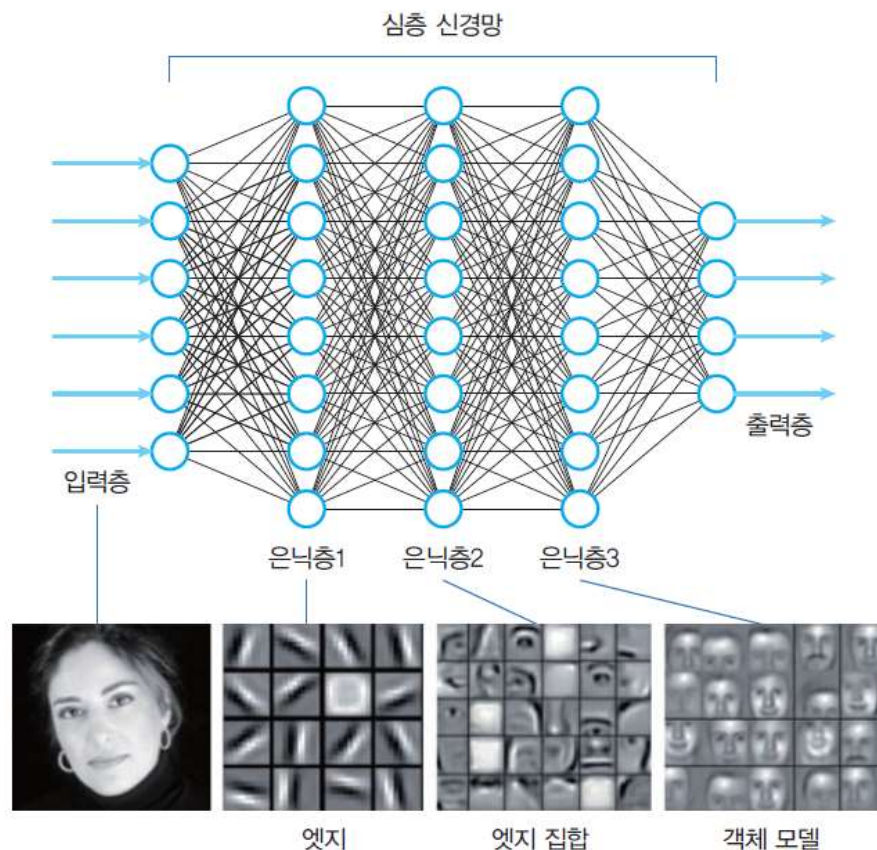
CPU와 GPU의 성능비교





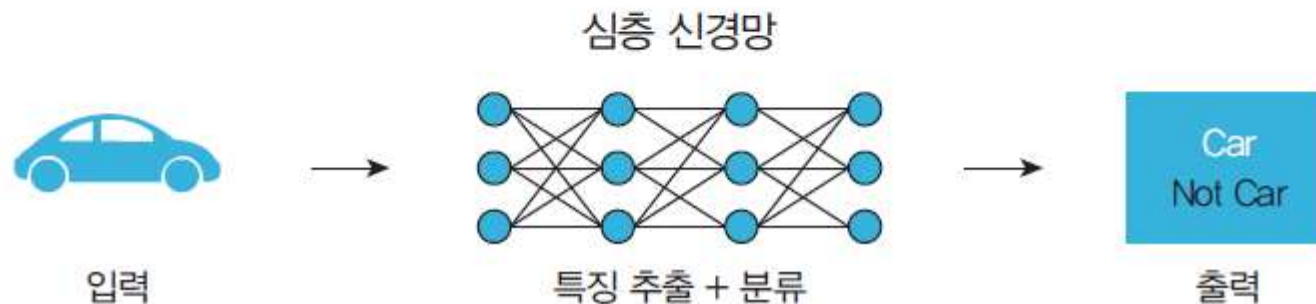
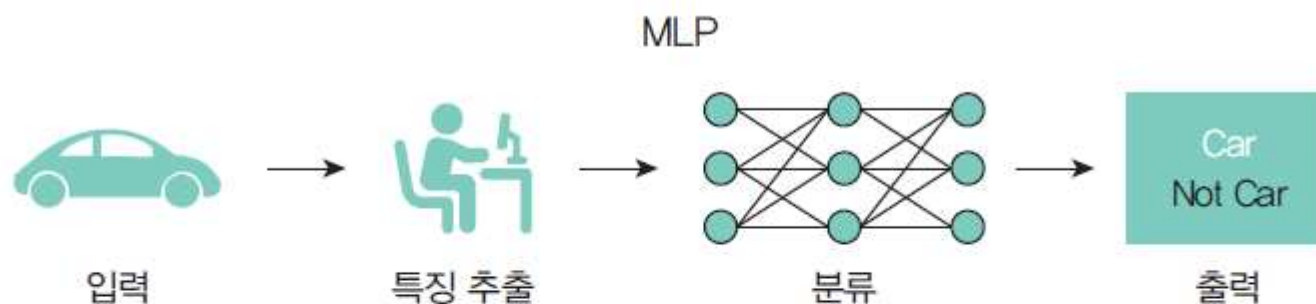
# 은닉층의 역할: 에지 → 부분 객체 → 객체

- 은닉층은 무슨 역할을 하고 있는 것일까? -> 여러 개의 은닉층 중에서 앞단은 경계선(에지)과 같은 저급 특징들을 추출하고 뒷단은 부분 객체, 객체와 같은 고급 특징들을 추출한다.





# MLP vs DNN

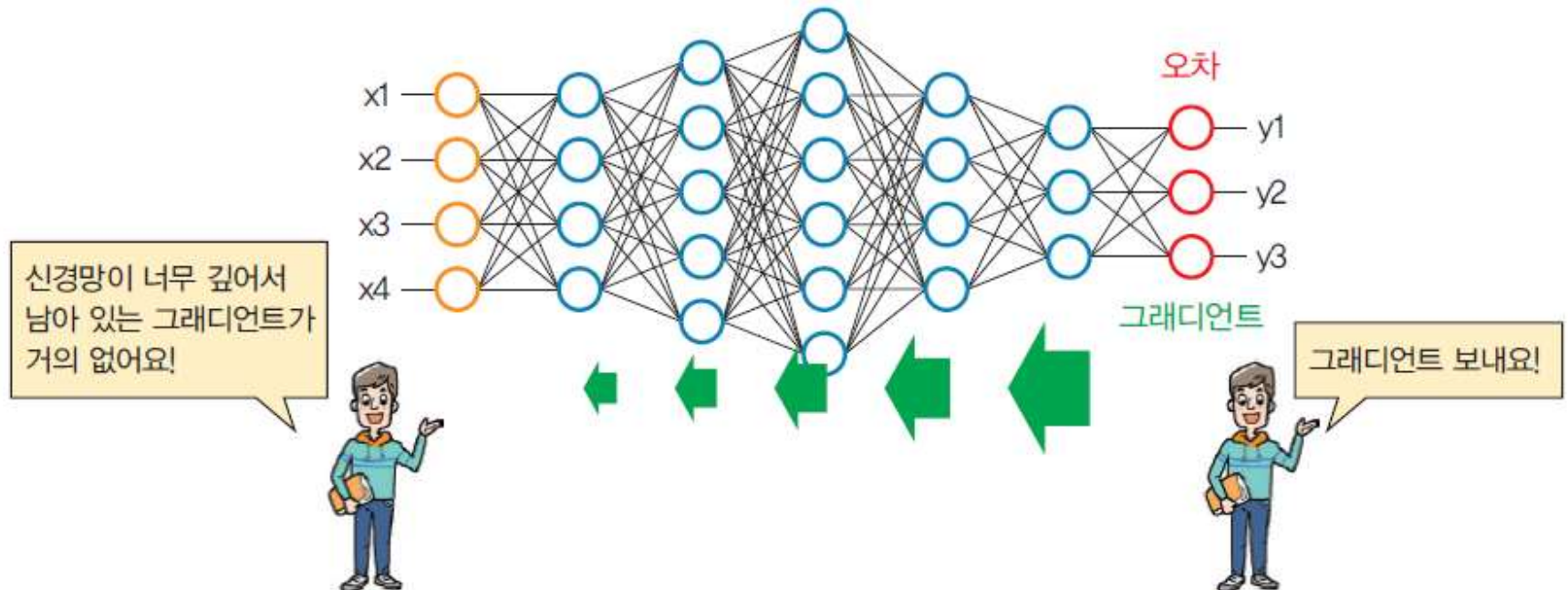






# 그래디언트 소실 문제

- 심층 신경망에서 그래디언트가 전달되다가 점점 0에 가까워지는 현상이다







# 범인은?

- 범인은 시그모이드 활성화 함수로 드러난다.
- 시그모이드 함수의 특성상, 아주 큰 양수나 음수가 들어오면 출력이 포화되어서 거의 0이 된다.

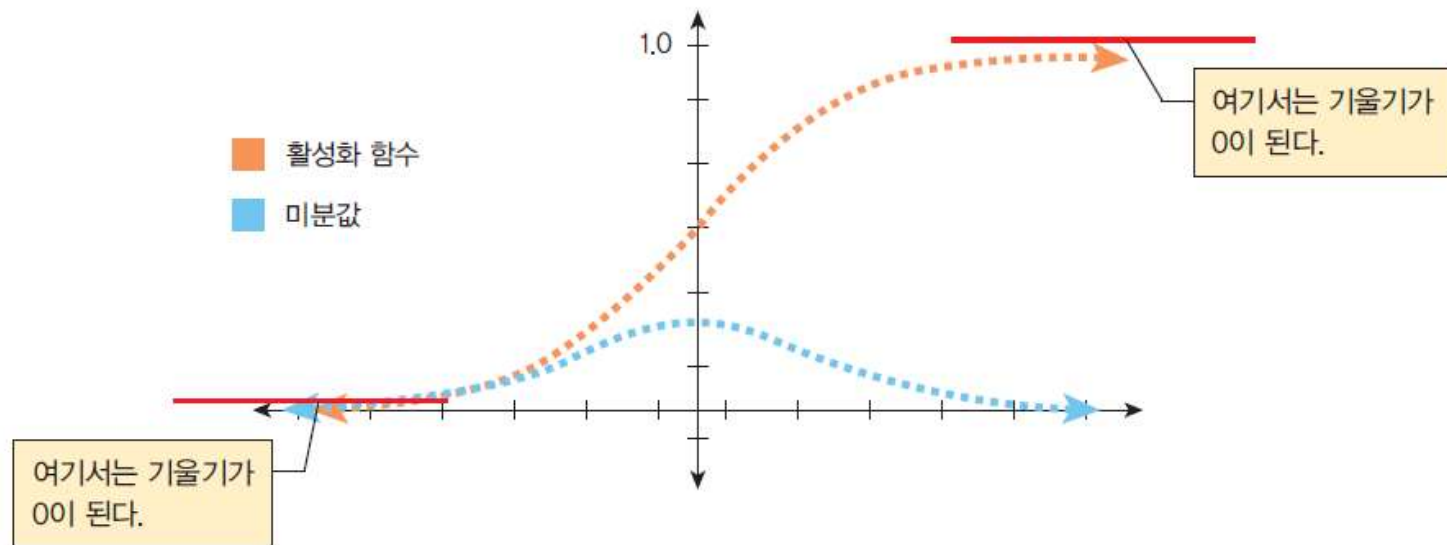


그림 8-5 시그모이드 함수와 그래디언트 소실



# 역전파 알고리즘의 복잡: **vanishing gradient**

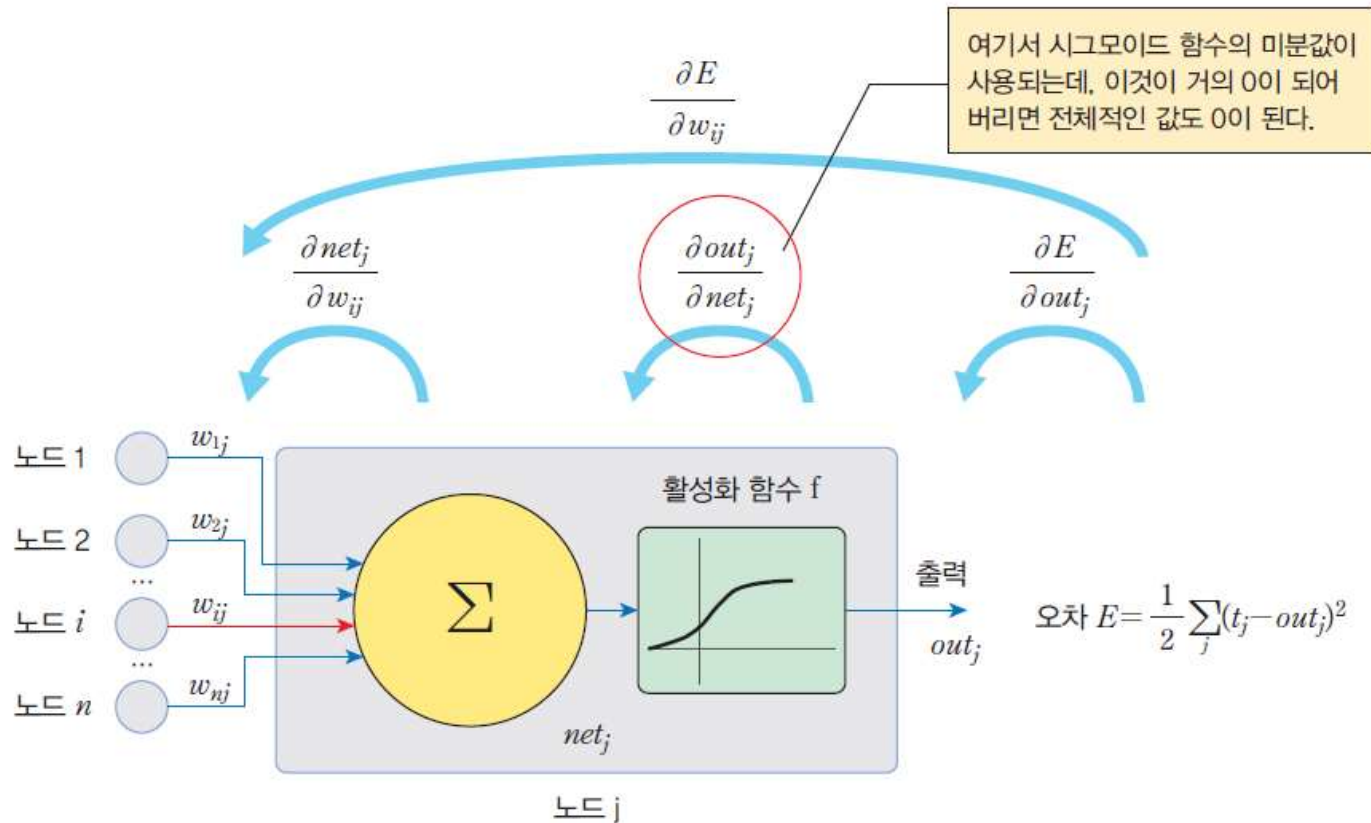
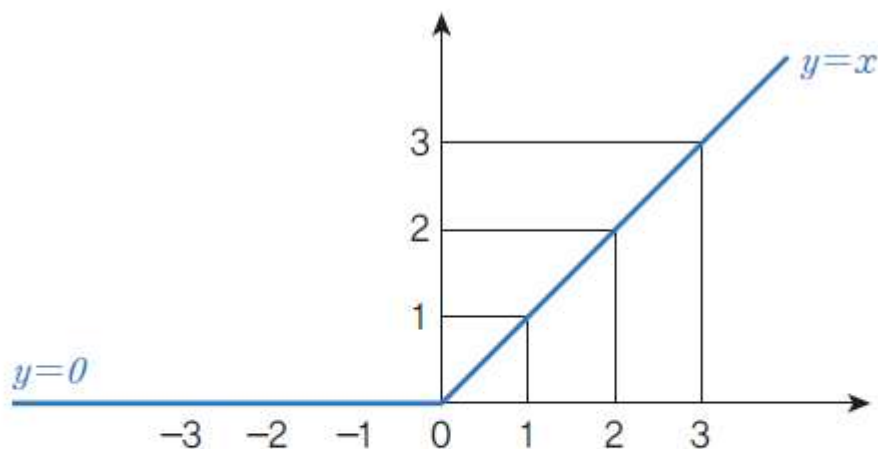


그림 8-6 그래디언트 소실 문제



# 새로운 활성화 함수

- 그래디언트 소실 문제를 해결하기 위하여 심층 신경망에서는 활성화 함수로 **ReLU 함수**를 많이 사용한다.



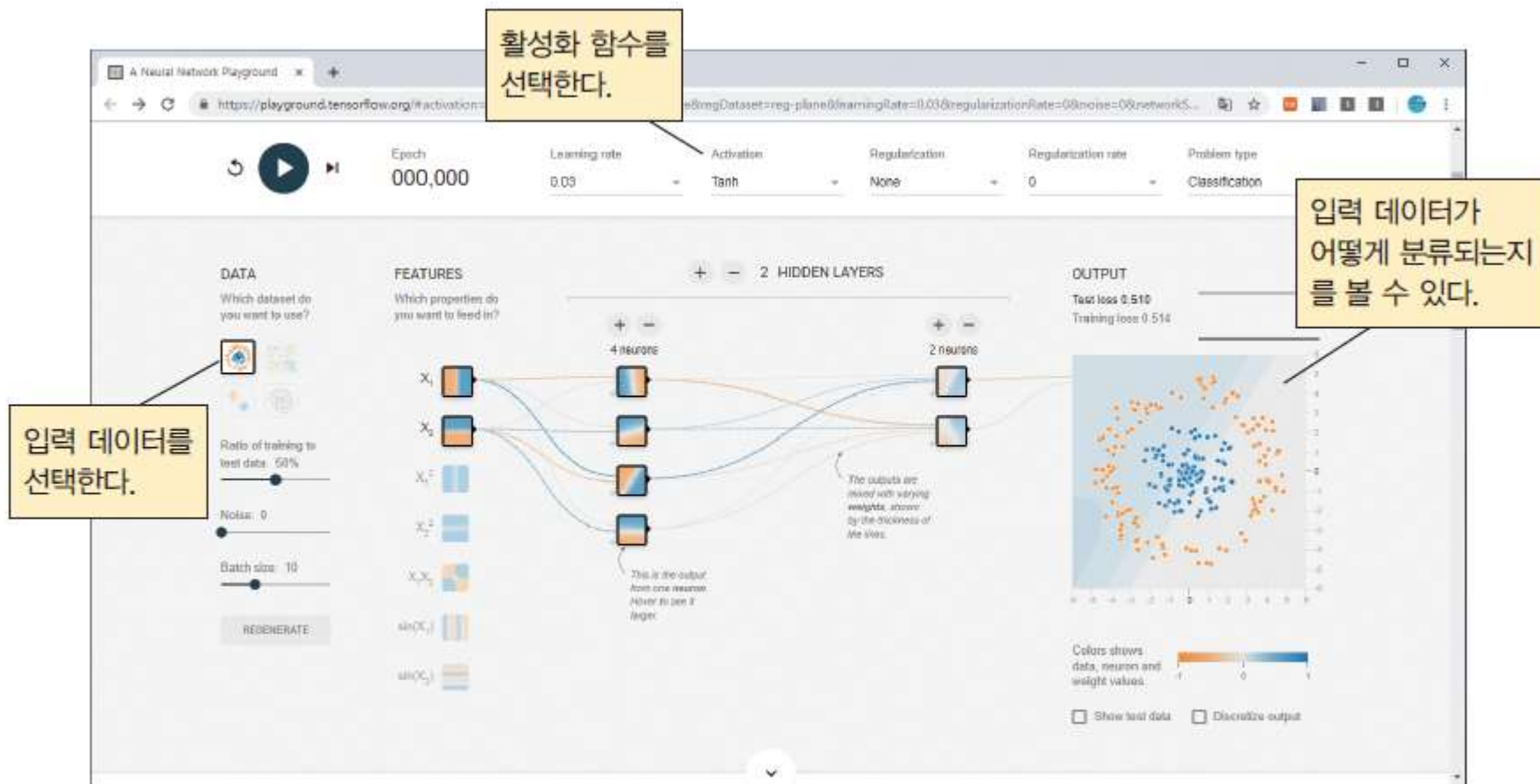
$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

그림 8-7 ReLU 함수



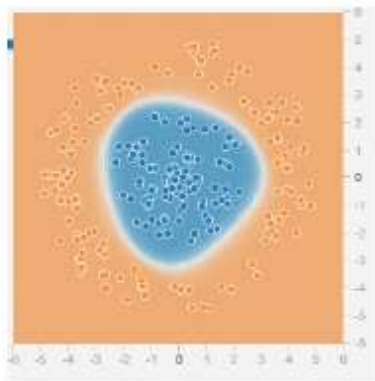
# Lab: 활성화 함수 실험

- <https://playground.tensorflow.org>로 실험

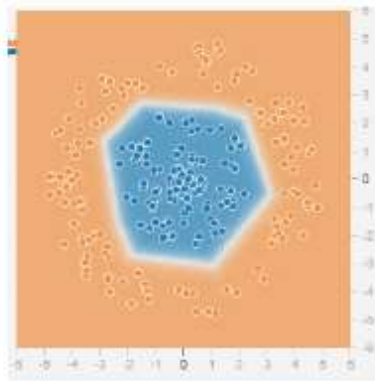




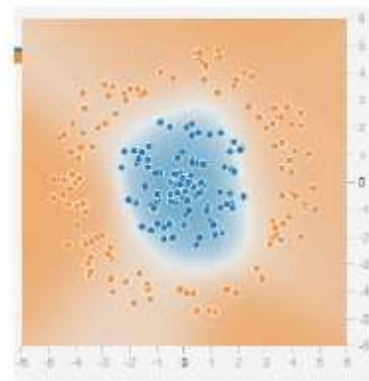
# 다양한 활성화 함수 변경



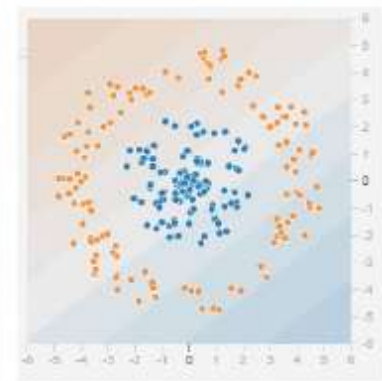
tanh



ReLU



Sigmoid



Linear



# 손실 함수 선택 문제

- 손실 함수로는 이제까지는 평균 제곱 오차(Mean Squared Error: MSE)를 사용

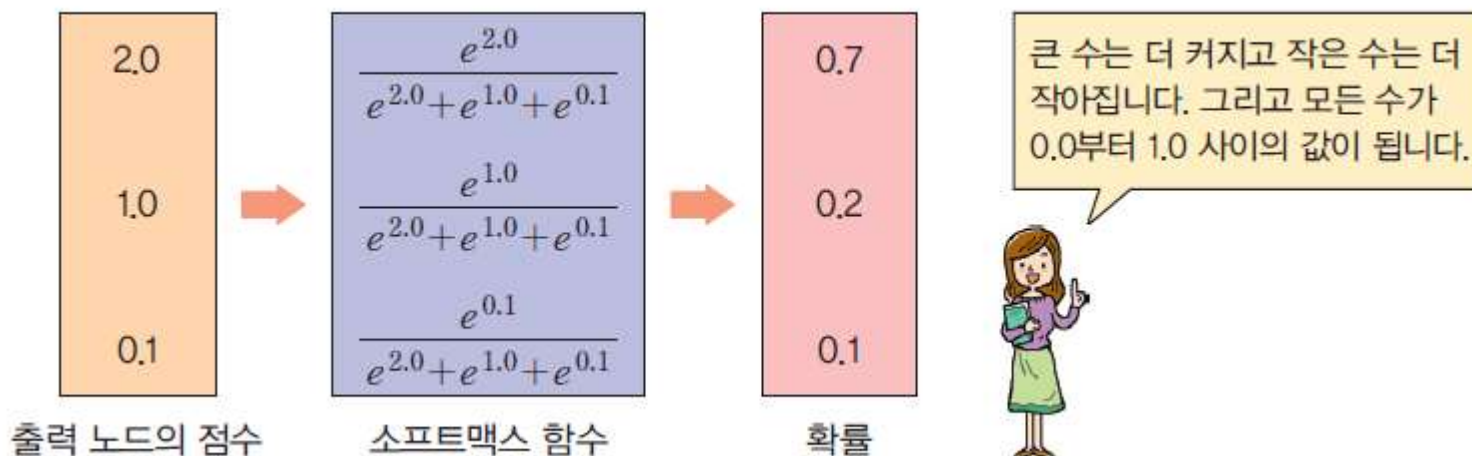
$$E = \frac{1}{m} \sum (y_i - \hat{y}_i)^2$$

- 정답과 예측값의 차이가 커지면 MSE도 커지므로 충분히 사용할 수 있는 손실 함수이다.
- 하지만 분류 문제를 위해서는 MSE보다는 더 성능이 좋은 손실 함수가 개발되어 있다.



# 소프트맥스 (softmax) 활성화 함수

- “소프트맥스”라는 이름이 붙은 이유는 이것이 max 함수의 소프트한 버전이기 때문이다. Max 함수의 출력은 전적으로 최대 입력값에 의하여 결정된다







# 교차 엔트로피 손실 함수 : **cross-entropy**

- 교차 엔트로피는 2개의 확률 분포  $p$ ,  $q$ 에 대해서 다음과 같이 정의된다.

$$H(p, q) = - \sum_x p(x) \log_n q(x)$$

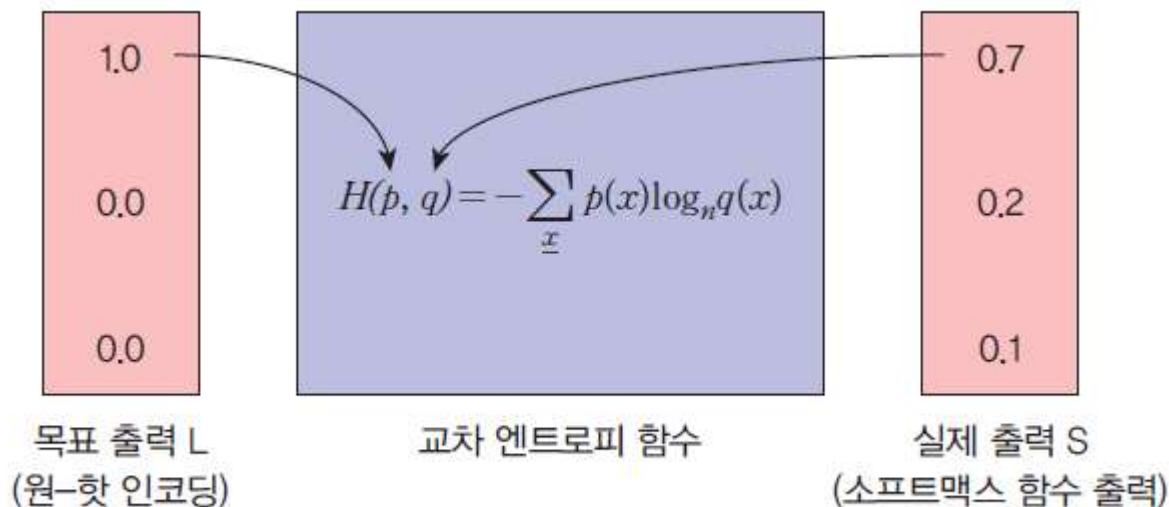


그림 8-9 교차 엔트로피 함수



# 교차 엔트로피 손실 함수

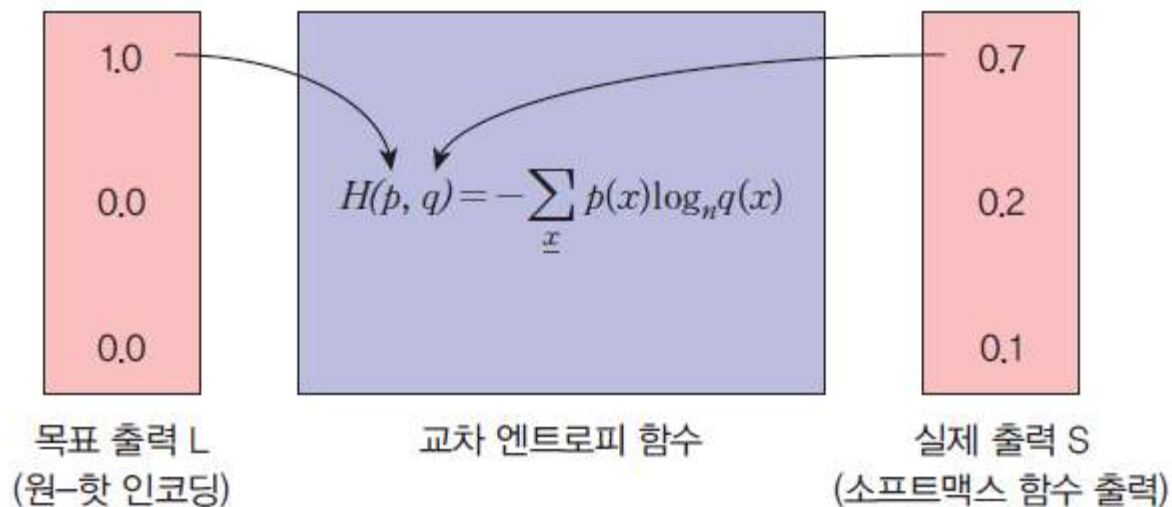


그림 8-9 교차 엔트로피 함수

$$\begin{aligned} H(p, q) &= - \sum_x p(x) \log_n q(x) \\ &= -(1.0 * \log 0.7 + 0.0 * \log 0.2 + 0.0 * \log 0.1) \\ &= 0.154901 \end{aligned}$$



# 완벽하게 일치한다면

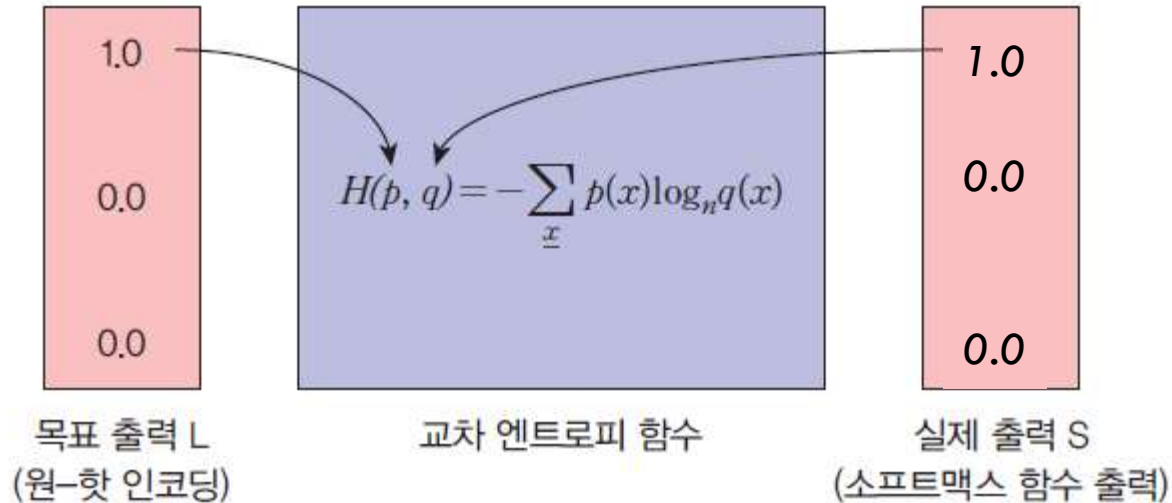


그림 8-9 교차 엔트로피 함수

$$\begin{aligned} H(p, q) &= - \sum_x p(x) \log_n q(x) \\ &= -(1.0 * \log 1.0 + 0.0 * \log 0.0 + 0.0 * \log 0.0) \\ &= 0 \end{aligned}$$



# Lab: 교차 엔트로피의 계산

입력 샘플	실제 출력			목표 출력		
샘플 #1	0.1	0.3	0.6	0	0	1
샘플 #2	0.2	0.6	0.2	0	1	0
샘플 #3	0.3	0.4	0.3	1	0	0

▷ 첫 번째 샘플에 대하여 교차 엔트로피를 계산해보자.  $-(\log(0.1) * 0 + \log(0.3) * 0 + \log(0.6) * 1) = -(0 + 0 - 0.51) = 0.51$ 이 된다.

▷ 두 번째 샘플의 교차 엔트로피는  $-(\log(0.2) * 0 + \log(0.6) * 1 + \log(0.2) * 0) = -(0 - 0.51 + 0) = 0.51$ 이다.

▷ 세 번째 샘플의 교차 엔트로피는  $-(\log(0.3) * 1 + \log(0.4) * 0 + \log(0.3) * 0) = -(-1.2 + 0 + 0) = 1.20$ 이다.

▷ 따라서 3개 샘플의 평균 교차 엔트로피 오류는  $(0.51 + 0.51 + 1.20) / 3 = 0.74$ 가 된다.



# 평균 제곱 오차 vs 교차 엔트로피

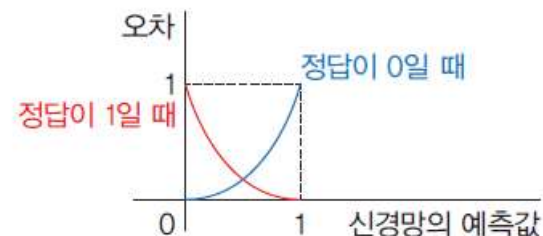
- 문제를 간단히 하기 위하여 출력 유닛이 하나인 경우만 생각하자

- 평균 제곱 오차:  $E = (y - \hat{y}_i)^2$

- 이진 교차 엔트로피:  $E = -[y \log_n(\hat{y}) + (1 - y) \log_n(1 - \hat{y})]$

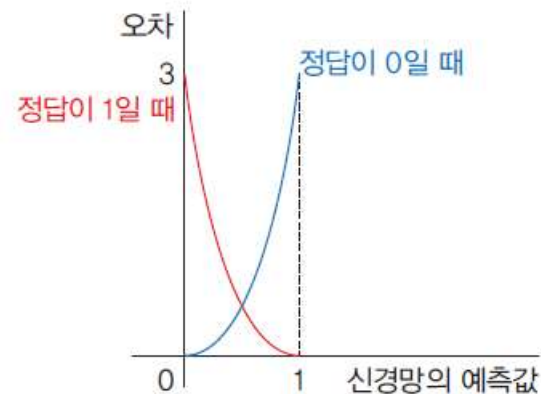
- 평균 제곱 오차(MSE):  $E = \hat{y}_i^2$  (정답이 0일 때)

$$E = (1 - \hat{y}_i)^2 \text{ (정답이 1일 때)}$$



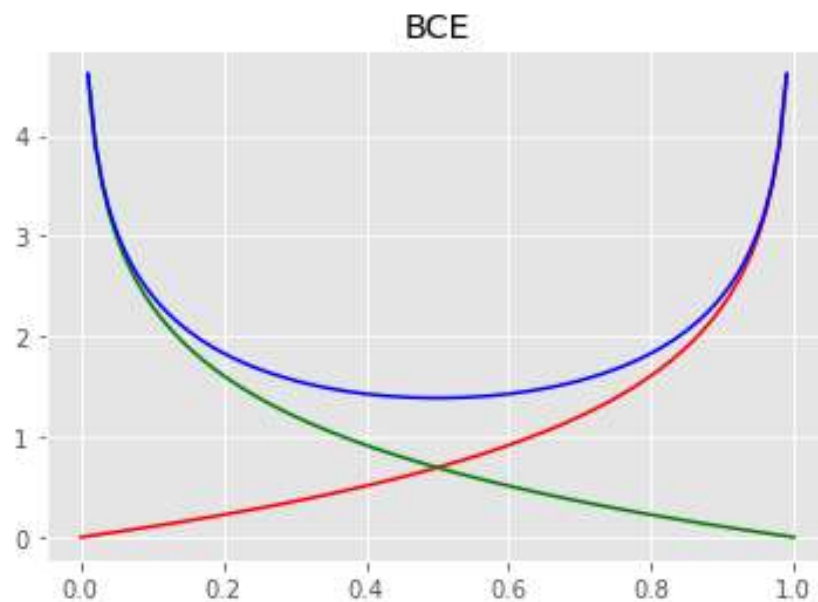
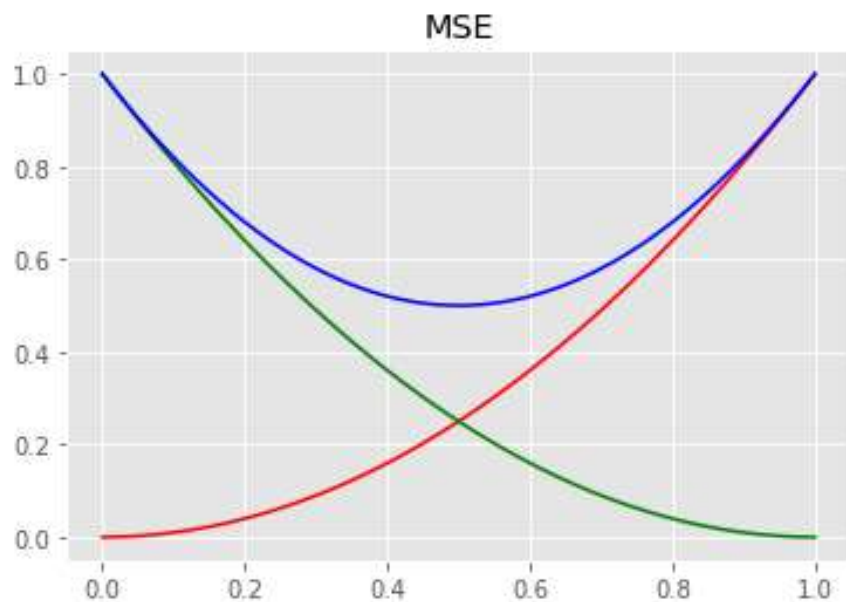
- 이진 교차 엔트로피(BCE) 오차:  $E = -\log_n(1 - \hat{y})$  (정답이 0일 때)

$$E = -\log_n(\hat{y}) \text{ (정답이 1일 때)}$$





# 평균 제곱 오차 vs 교차 엔트로피 : Graph

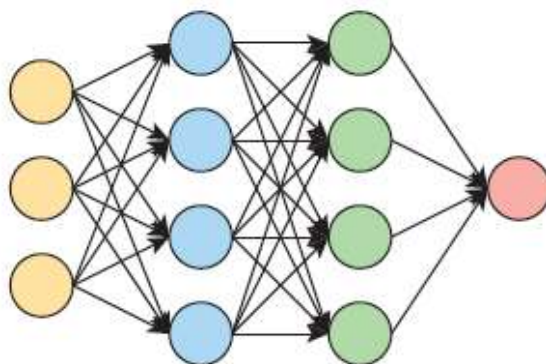


<https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>



# 케라스에서 손실 함수 - BinaryCrossentropy

- **BinaryCrossentropy**: 이진 교차 엔트로피(BCE)는 이진 분류 문제를 해결하는 데 사용되는 손실 함수이다.
- 즉 우리가 분류해야 하는 부류가 두 가지뿐일 때 사용한다. 예를 들어 이미지를 “강아지”, “강아지 아님”의 두 부류로 분류할 때 BinaryCrossentropy를 사용한다.



1 → 강아지  
0 → 강아지 아님

$$BCE = -\frac{1}{n} \sum_{i=1}^n (y_i \log_n(\hat{y}_i) + (1 - y_i) \log_n(1 - \hat{y}_i))$$





# 실제 계산예

실제 레이블 $y$	1	0	0	1
예측 값 $\hat{y}$	0.8	0.3	0.5	0.9

샘플 1:  $BCE1 = -(1 \cdot \log(0.8) + (1-1) \cdot \log(1-0.8))$

샘플 2:  $BCE2 = -(0 \cdot \log(0.3) + (1-0) \cdot \log(1-0.3))$

샘플 3:  $BCE3 = -(0 \cdot \log(0.5) + (1-0) \cdot \log(1-0.5))$

샘플 4:  $BCE4 = -(1 \cdot \log(0.9) + (1-1) \cdot \log(1-0.9))$

$$BCE = \frac{BCE1 + BCE2 + BCE3 + BCE4}{4} \approx 0.345$$

```
y_true = [ [1], [0], [0], [1] ]  
y_pred = [[0.8], [0.3], [0.5], [0.9]]  
bce = tf.keras.losses.BinaryCrossentropy()  
print(bce(y_true, y_pred).numpy())
```

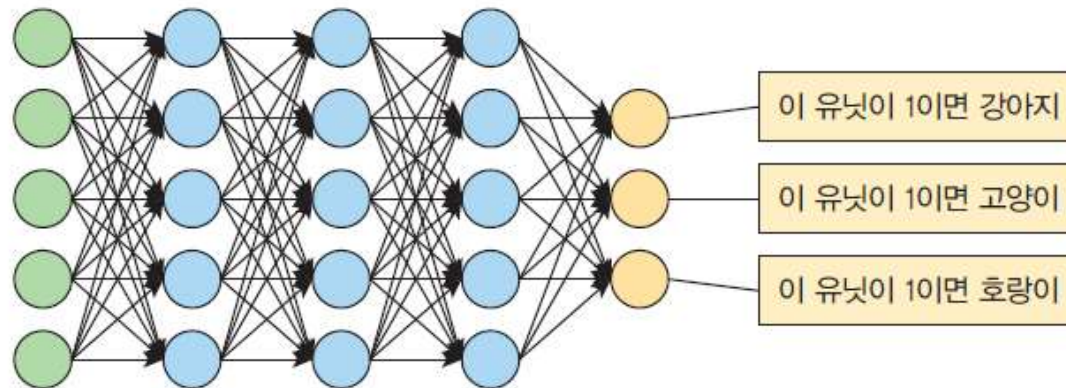
0.3445814|



다중 분류 :

# CategoricalCrossentropy

- **CategoricalCrossentropy**: 우리가 분류해야 할 부류가 두 개 이상이라면 (즉 다중 분류 문제라면) 을 사용
- 정답 레이블은 원-핫 인코딩으로 제공한다. 예를 들어서 입력 이미지를 “강아지(1,0, 0)”, “고양이(0, 1, 0)”, “호랑이(0, 0, 1)” 중의 하나로 분류



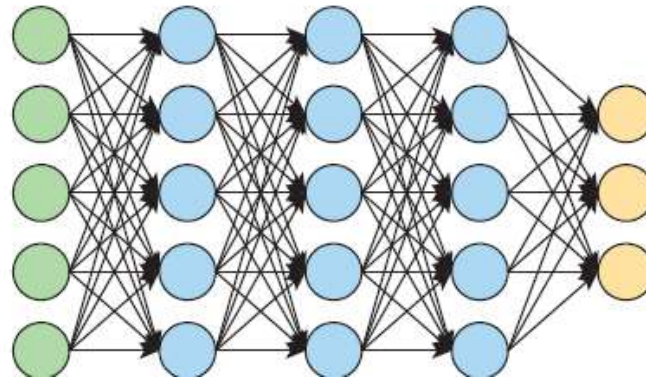
```
y_true = [[0.0, 1.0, 0.0], [0.0, 0.0, 1.0], [1.0, 0.0, 0.0]] # 고양이, 호랑이, 강아지
y_pred = [[0.6, 0.3, 0.1], [0.3, 0.6, 0.1], [0.1, 0.7, 0.2]]
cce = tf.keras.losses.CategoricalCrossentropy ()
print(cce(y_true, y_pred).numpy ())
```

1.936381



# SparseCategoricalCrossentropy

- SparseCategoricalCrossentropy: 정답 레이블이 원-핫 인코딩이 아니고 정수로 주어지면 SparseCategoricalCrossentropy를 사용
- 예를 들어서 정답 레이블이 0(강아지), 1(고양이), 2(호랑이)로 주어지면 SparseCategoricalCrossentropy를 사용한다.



강아지이면 0  
고양이이면 1  
호랑이이면 2

케라스가 자동으로  
원-핫 인코딩으로  
변환한다.

```
y_true = [1, 2, 0] # 고양이, 호랑이, 강아지
y_pred = [[0.6, 0.3, 0.1], [0.3, 0.6, 0.1], [0.1, 0.7, 0.2]]
scce = tf.keras.losses.SparseCategoricalCrossentropy()
print(scce(y_true, y_pred).numpy())
```

1.936381



# 가중치 초기화 문제

- 가중치의 초기값도 성능에 많은 영향을 끼친다. 신경망의 초기에는 연구가 부족하여서 초기값을 0으로 주는 경우도 많았다. -> 오차의 역전파가 되지 않는다.

$$\delta_j = \begin{cases} (out_j - t_j) f'(net_j) & j \text{가 출력층 노드이면} \\ (\sum_k w_{jk} \delta_k) f'(net_j) & j \text{가 은닉층 노드이면} \end{cases}$$

가중치가 0이라면 델타가 전달되지 않는다.



# 가중치 초기화 문제

- 가중치  $w_1, w_2, w_3, w_4$ 와 가중치  $w_5, w_6, w_7, w_8$ 이 모두 같은 값(예를 들어서 0.2)으로 설정되어 있다면, 노드 E와 노드 G는 완벽하게 동일한 일을 하게 된다. 이것을 방지하는 것을 “균형 깨뜨리기(breaking the symmetry)”라고 부르고 있다.

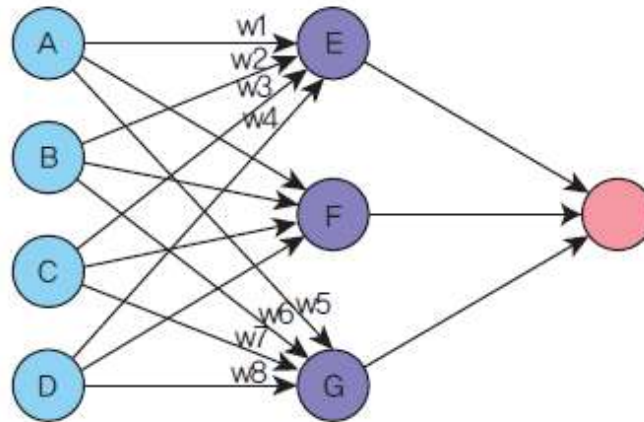
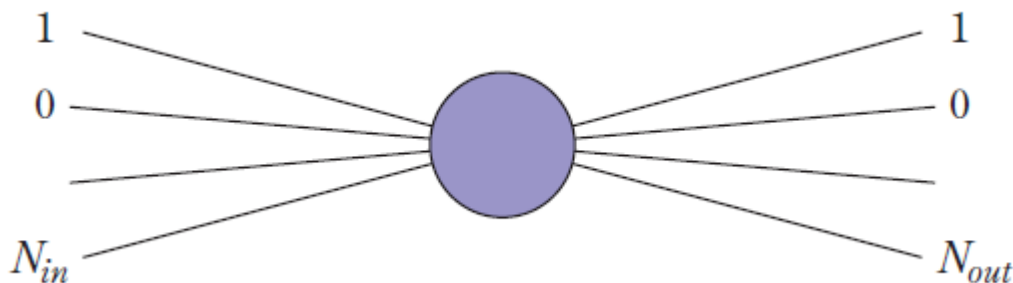


그림 8-10 가중치가 동일할 때의 문제점



# 난수 가중치 초기값 - Xavier의 방법

- 이상과 같은 이유로 **가중치의 초기값은 난수로 결정**되어야 한다.
- 또 반대로 너무 큰 가중치는 그래디언트 폭발(exploding gradients)을 일으킨다고 한다. 이 경우에는 학습이 수렴하지 않고 발산하게 된다
- Xavier의 방법

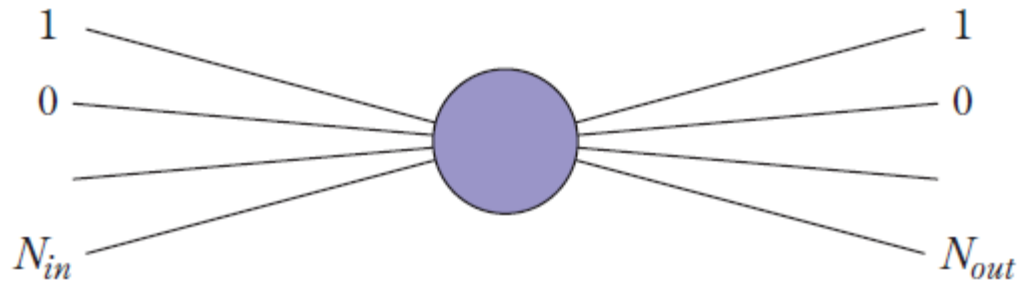


$$\text{var}(w_i) = \frac{1}{N_{in}}$$



# 난수 가중치 초기화 - He의 방법

- He의 방법



$$var(w_i) = \frac{2}{(N_{in} + N_{out})}$$

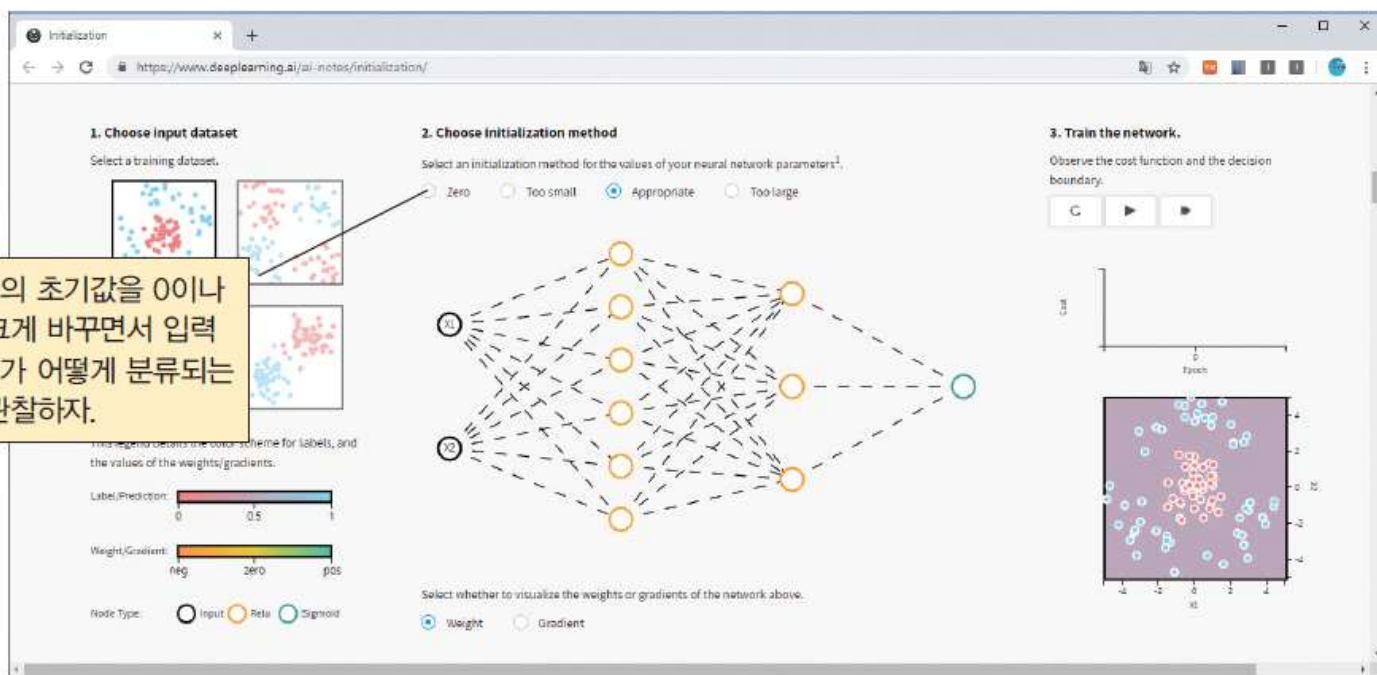




# Lab: 가중치 초기화 실험

- <https://www.deeplearning.ai/ai-notes/initialization/>에 가보면 여러분들은 다양한 초기값을 가지고 오차가 얼마나 빨리 줄어드는지를 볼 수 있다.

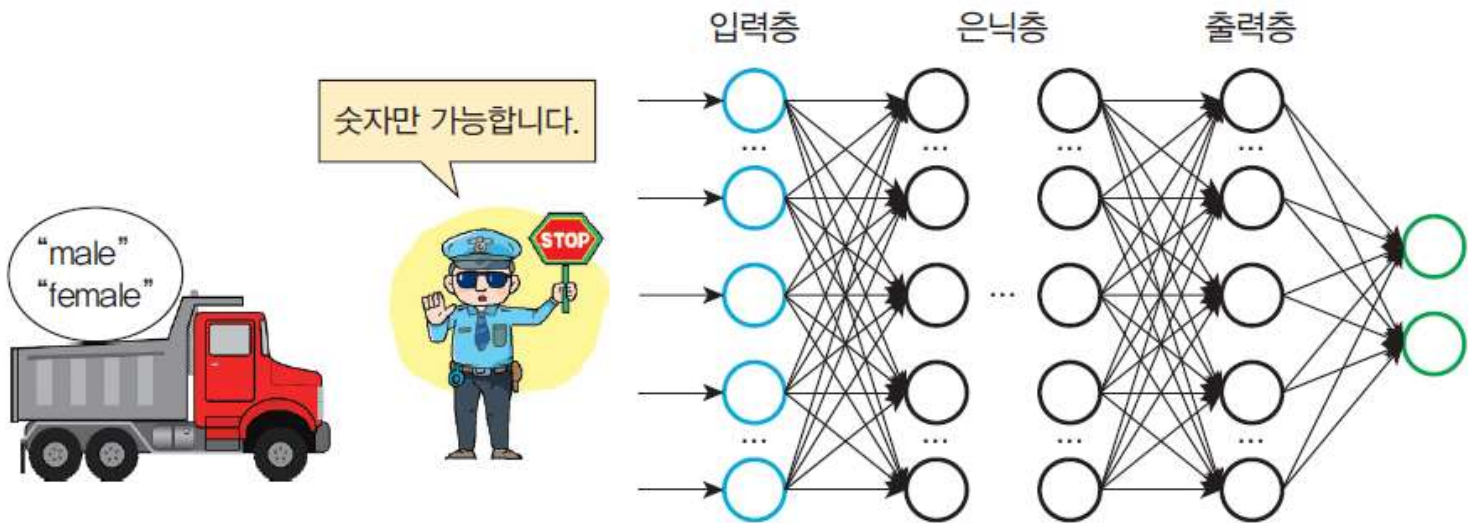
가중치의 초기값을 0이나 작게 크게 바꾸면서 입력 데이터가 어떻게 분류되는지를 관찰하자.





# 버퍼형 데이터 처리

- 입력 데이터 중에는 “male”, “female”과 같이 카테고리 (Category)를 가지는 데이터들이 아주 많다. -> 숫자로 바꾸어야 한다.





# 가단하 경우

```
for ix in train.index:  
    if train.loc[ix, 'Sex']=="male":  
        train.loc[ix, 'Sex']=1  
    else:  
        train.loc[ix, 'Sex']=0
```



# 일반적인 범주형 데이터 변환 방법

- 범주형 변수를 인코딩하는 3가지 방법
  - 정수 인코딩(Integer Encoding): 각 레이블이 정수로 매핑되는 경우.
  - 원-핫 인코딩(One Hot Encoding): 각 레이블이 이진 벡터에 매핑되는 경우.
  - 임베딩(Embedding): 범주의 분산된 표현이 학습되는 경우이다.

자연어 처리에서 설명



# 정수 인코딩

- sklearn 라이브러리가 제공하는 Label Encoder 클래스를 사용

```
import numpy as np
X = np.array([[ 'Korea', 44, 7200],
              [ 'Japan', 27, 4800],
              [ 'China', 30, 6100]])
```

이것만 바꾸는 예제

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
X[:, 0] = labelencoder.fit_transform(X[:, 0])
print(X)
```

```
[[ '2' '44' '7200']
 [ '1' '27' '4800']
 [ '0' '30' '6100']]
```



# 원-핫 인코딩

- 원-핫 인코딩은 단 하나의 값만 1이고 나머지는 모두 0인 인코딩을 의미한다. -> 아주 많이 사용된다.

Country	Age	Salary
Korea	38	7200
Japan	27	4800
China	30	3100



Korea	Japan	China	Age	Salary
1	0	0	38	7200
0	1	0	27	4800
0	0	1	30	3100



# 원-핫 인코딩

```
import numpy as np
X = np.array(['Korea', 44, 7200],
              ['Japan', 27, 4800],
              ['China', 30, 6100])

from sklearn.preprocessing import OneHotEncoder
onehotencoder = OneHotEncoder()

# 원하는 열을 뽑아서 2차원 배열로 만들어서 전달하여야 한다.
XX = onehotencoder.fit_transform(X[:,0]).toarray()
print(XX)

X = np.delete(X, [0], axis=1)      # 0번째 열 삭제
X = np.concatenate((XX, X), axis = 1) # X와 XX를 붙인다.
print(X)
```





```
[[0. 0. 1.]  
 [0. 1. 0.]  
 [1. 0. 0.]]  
[['0.0' '0.0' '1.0' '44' '7200']  
 ['0.0' '1.0' '0.0' '27' '4800']  
 ['1.0' '0.0' '0.0' '30' '6100']]
```



# 원-핫 인코딩(케라스 사용)

- 원-핫 인코딩은 케라스의 `to_categorical()`을 호출해서도 만들 수 있다.-> 아주 많이 사용된다.

```
class_vector =[2, 6, 6, 1]
```

```
from tensorflow.keras.utils import to_categorical  
output = to_categorical(class_vector, num_classes = 7, dtype ="int32")  
print(output)
```

```
[[0 0 1 0 0 0 0]  
 [0 0 0 0 0 0 1]  
 [0 0 0 0 0 0 1]  
 [0 1 0 0 0 0 0]]
```



# 원-핫 인코딩 (케라스 사용-mnist)

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape, y_train.shape)
y_train[:10]
```

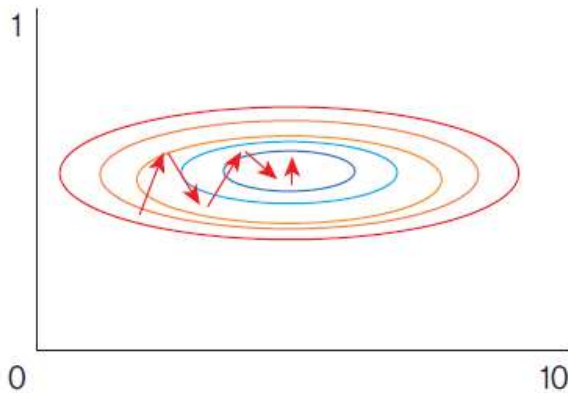
```
y_train_one_hot = to_categorical(y_train, num_classes = 10) #, dtype = "int32")
y_train_one_hot.shape
y_train_one_hot[:10]
```

```
array([[0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]])
```

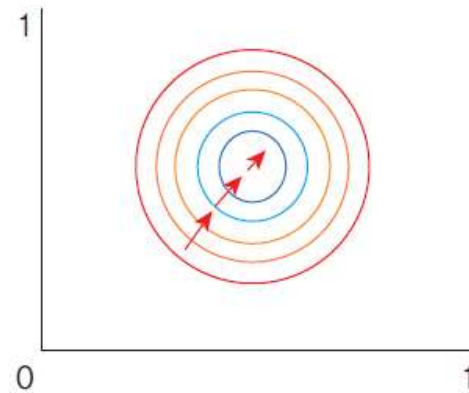


# 데이터 정규화

- 신경망은 일련의 선형 조합과 비선형 활성화 함수를 통해 이러한 차이가 나는 입력을 결합하여 학습하게 되므로 각 입력과 관련된 매개변수도 서로 다른 범위를 가지면서 학습된다.



특징값의 범위가  
다른 경우



특징값이  $[0, 1]$ 에서  
움직이는 경우

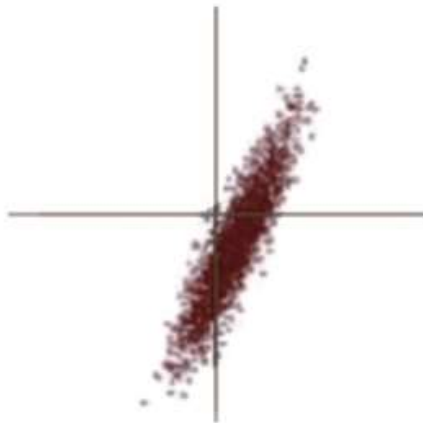
그림 8-11 데이터 정규화의 영향



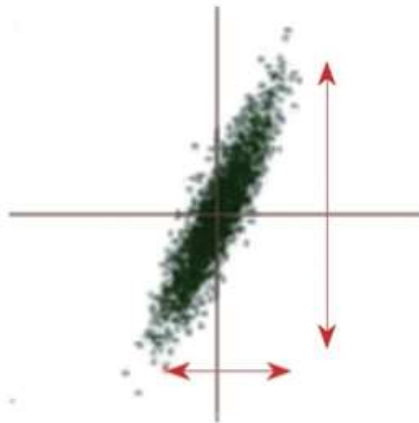
# 데이터 정규화 (Normalization)

- 부동 소수점 숫자 정밀도와 관련된 문제를 피하기 위해, 입력 값이 대략 -1.0에서 1.0 범위에 있도록 하는 것이 좋다

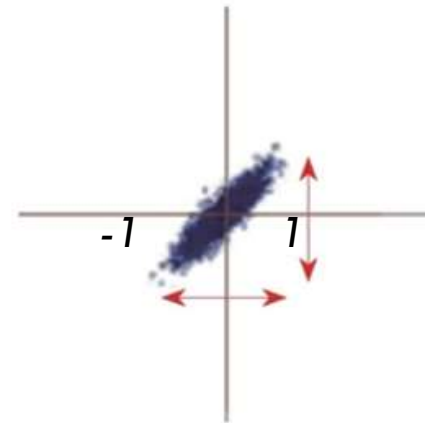
원래의 데이터



원점이 데이터의 중심이 되도록 함



정규화된 데이터

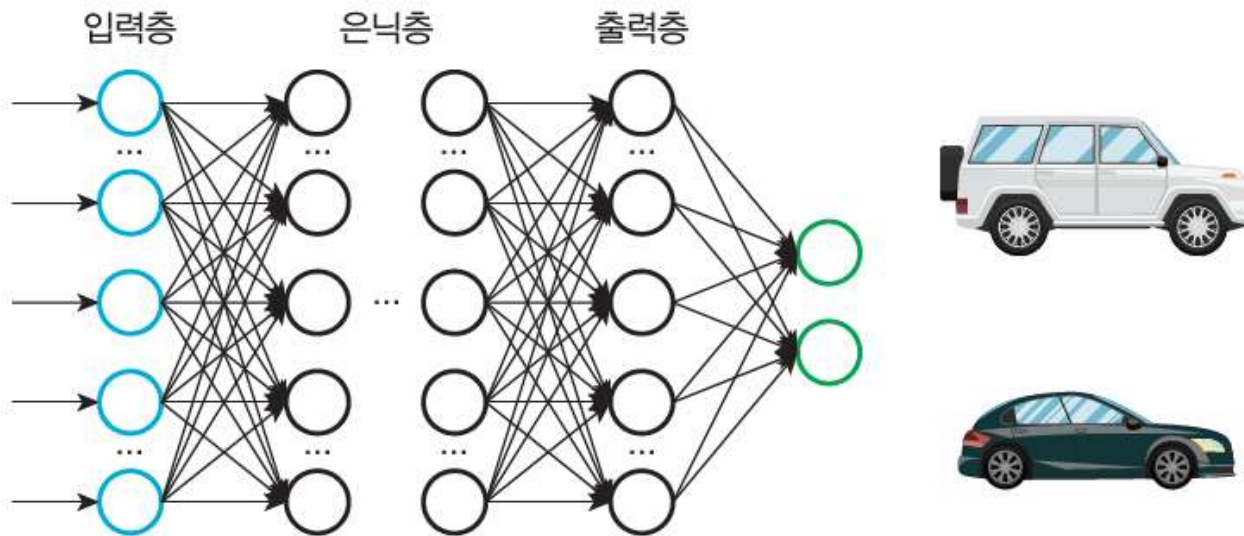


$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$



# 예제

- 사람의 나이, 성별, 연간 수입을 기준으로, 선호하는 자동차의 타입(세단 아니면 SUV)을 예측할 신경망을 만들고 싶다고 가정하자





# 훈련 데이터 정규화

범주형 데이터

[0]	30	male	3800	SUV
[1]	36	female	4200	SEDAN
[2]	52	male	4000	SUV
[3]	42	female	4400	SEDAN

정규화 필요

[0]	-1.23	-1.0	-1.34	(1.0 0.0)
[1]	-0.49	1.0	0.45	(0.0 1.0)
[2]	1.48	-1.0	-0.45	(1.0 0.0)
[3]	0.25	1.0	1.34	(0.0 1.0)



# sklearn의 데이터 정규화 방법 - MinMaxScaler

$$X\_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))$$
$$X\_scaled = X\_std * (max - min) + min$$

```
from sklearn.preprocessing import MinMaxScaler
```

```
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
```

```
scaler = MinMaxScaler()
```

```
scaler.fit(data) # 최대값과 최소값을 알아낸다.
```

```
print(scaler.transform(data)) # 데이터를 변환한다.
```

```
[[0.  0. ]  
 [0.25 0.25]  
 [0.5  0.5 ]  
 [1.  1.  ]
```





# sklearn의 데이터 정규화 방법 - StandardScaler

```
# StandardScaler()
from sklearn.preprocessing import StandardScaler
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
# data = [[-1, 2, 7], [-0.5, 6, 3], [0, 10, -2], [1, 18, 6]]

scaler = StandardScaler()
scaler.fit(data)          # 평균=0, 분산=std 로 데이터 변경
print(scaler.transform(data))
```

```
[[ -1.18321596 -1.18321596]
 [-0.50709255 -0.50709255]
 [ 0.16903085  0.16903085]
 [ 1.52127766  1.52127766]]
```



# 케라스의 데이터 정규화 방법 - **Normalization**

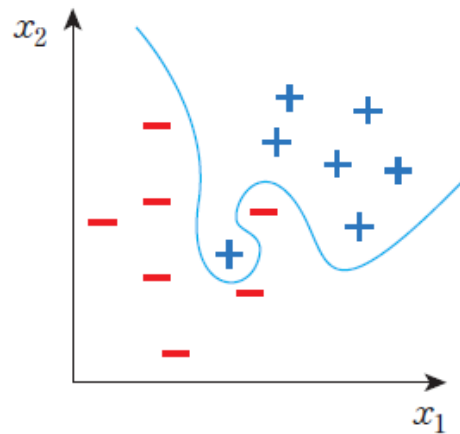
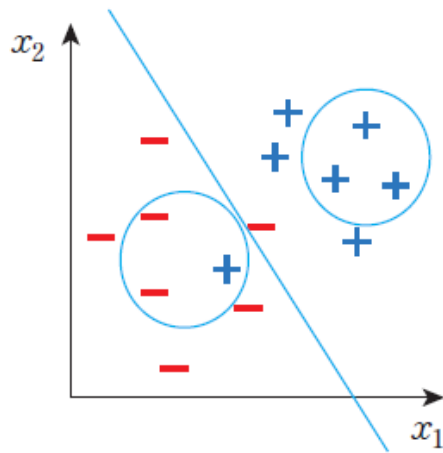
- 데이터 정규화가 필요하면 **Normalization** 레이어를 중간에 넣으면 된다.

```
>>> adapt_data = np.array([[1.], [2.], [3.], [4.], [5.]], dtype=np.float32)
>>> input_data = np.array([[1.], [2.], [3.]], dtype=np.float32)
>>> layer = Normalization()
>>> layer.adapt(adapt_data)
>>> layer(input_data)
<tf.Tensor: shape=(3, 1), dtype=float32, numpy=
array([[ -1.4142135 ],
       [ -0.70710677 ],
       [  0.          ]], dtype=float32)>
```



# 과잉 적합과 과소 적합

- 과잉 적합(over fitting)은 지나치게 훈련 데이터에 특화돼 실제 적용 시 좋지 못한 결과가 나오는 것을 말한다.

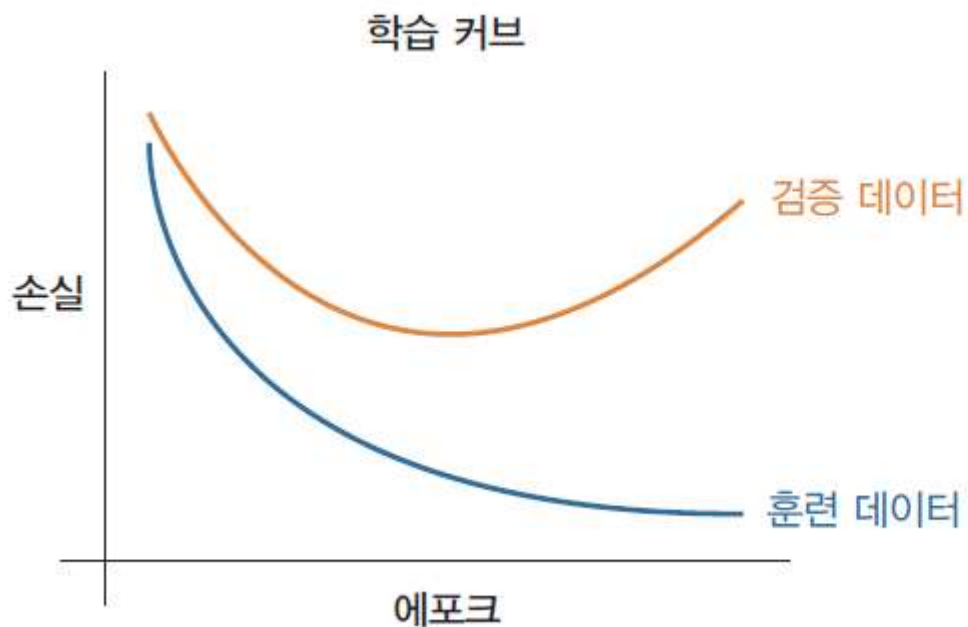


일반화에 실패!

그림 8-13 과잉 적합의 예



# 과잉 적합 (overfitting)을 아는 방법



훈련 데이터의 손실값은 계속 감소하지만 검증 데이터의 손실값은 오히려 증가하고 있네요!





# 과잉 적합의 예

- 영화 리뷰를 분류하는 문제를 생각해보자. IMDB 사이트는 영화에 대한 리뷰가 올라가 있는 사이트이다.
- 영화 리뷰를 입력하면 리뷰가 긍정적인지 부정적인지를 파악하는 신경망을 구현해보자

"I love this movie.  
I've seen it many times  
and it's still awesome."



"This movie is bad.  
I don't like it at all.  
It's terrible."





# IMDB 리뷰가 긍정적인지 부정적인지를 파악

```
import numpy as numpy
import tensorflow as tf
import matplotlib.pyplot as plt

# 데이터 다운로드
(train_data, train_labels), (test_data, test_labels) = \
    tf.keras.datasets.imdb.load_data( num_words=1000)
# imdb.load_data()의 인자로 num_words를 사용하면 이 데이터에서 등장 빈도 순위로
# 몇 등까지의 단어를 사용할 것인지를 의미

# 원-핫 인코딩으로 변환하는 함수
def one_hot_sequences(sequences, dimension=1000):
    results = numpy.zeros((len(sequences), dimension))
    for i, word_index in enumerate(sequences):
        results[i, word_index] = 1.
    return results

train_data = one_hot_sequences(train_data)
test_data = one_hot_sequences(test_data)
```



# 신경망 모델 구축

```
model = tf.keras.Sequential()  
model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(1000,)))  
model.add(tf.keras.layers.Dense(16, activation='relu'))  
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='adam',  
              metrics=['accuracy'])
```

# 신경망 훈련, 검증 데이터 전달

```
history = model.fit(train_data,  
                    train_labels,  
                    epochs=20,  
                    batch_size=512,  
                    validation_data=(test_data, test_labels),  
                    verbose=2)
```



# 훈련 데이터의 손실값과 검증 데이터의 손실값을 그래프에 출력

```
history_dict = history.history
```

```
loss_values = history_dict['loss']
```

```
val_loss_values = history_dict['val_loss']
```

```
acc = history_dict['accuracy']
```

```
epochs = range(1, len(acc) + 1)
```

```
# 훈련 데이터 손실값
```

```
# 검증 데이터 손실값
```

```
# 정확도
```

```
# 에포크 수
```

```
plt.plot(history.history['loss'])
```

```
plt.plot(history.history['val_loss'])
```

```
plt.title('Loss Plot')
```

```
plt.ylabel('loss')
```

```
plt.xlabel('epochs')
```

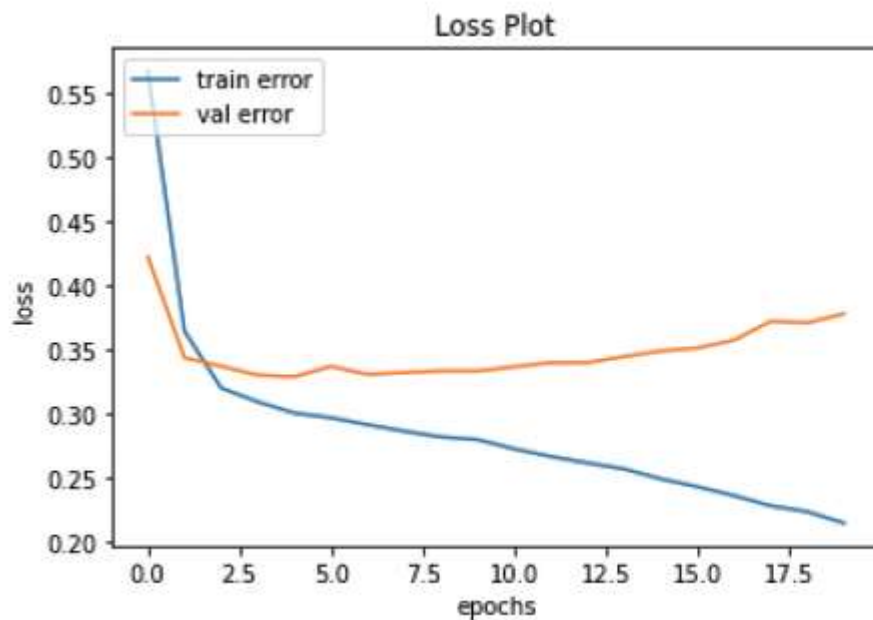
```
plt.legend(['train error', 'val error'], loc='upper left')
```

```
plt.show()
```





# 실행 결과



훈련 데이터의 손실값은 줄어  
들지만 검증 데이터의 손실값은  
오히려 증가하네요!





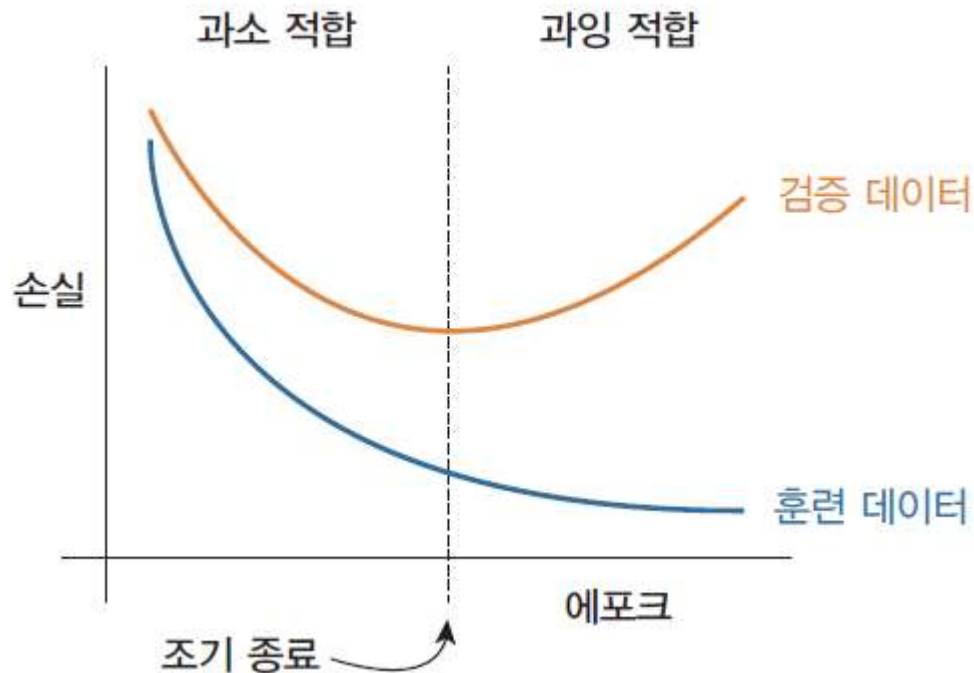
# 과잉 적합 방지 전략

- 조기 종료: 검증 손실이 증가하면 훈련을 조기에 종료한다.
- 가중치 규제 방법: 가중치의 절대값을 제한한다.
- 데이터 증강 방법: 데이터를 많이 만든다.
- 드롭아웃 방법: 몇 개의 뉴런을 쉬게 한다.



## 조기종료

- 검증 손실이 더 이상 감소하지 않는 것처럼 보일 때마다 훈련을 중단할 수 있다. 이런 식으로 훈련을 중단하는 것을 조기 종료(Early Stopping)라고 한다.





# 가중치 규제

- 가중치의 값이 너무 크면, 판단 경계선이 복잡해지고 과잉 적합이 일어난다는 사실을 발견하였다

$$Loss = Cost + \lambda \sum |W|$$

L1 규제

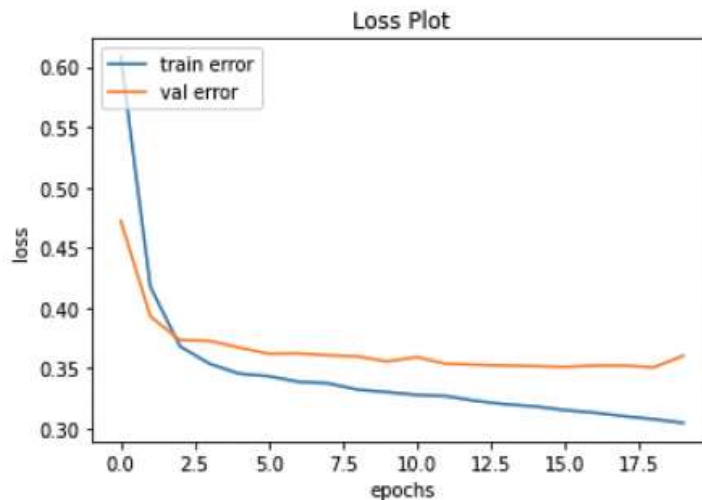
$$Loss = Cost + \lambda \sum W^2$$

L2 규제



# 신경망 모델 구축

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(16,
    kernel_regularizer=tf.keras.regularizers.l2(0.001),
    activation='relu', input_shape=(1000,)))
model.add(tf.keras.layers.Dense(16,
    kernel_regularizer=tf.keras.regularizers.l2(0.001), activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```



아직도 과잉 적합이지만 규제를  
적용한 모델이 덜 과잉 적합됨을  
알 수 있습니다.





# 드롭아웃 (dropout)

- 드롭아웃은 몇 개의 노드들을 학습 과정에서 랜덤하게 제외하는 것이다.

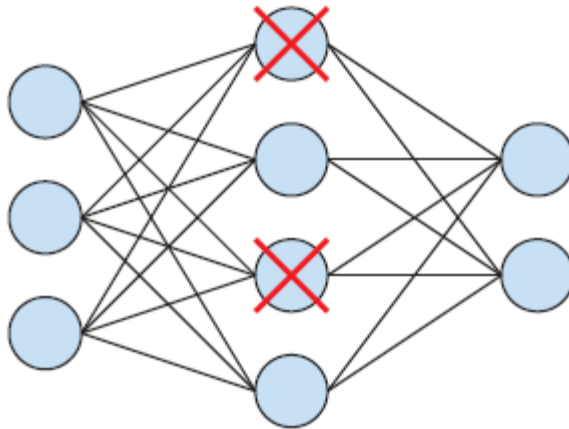


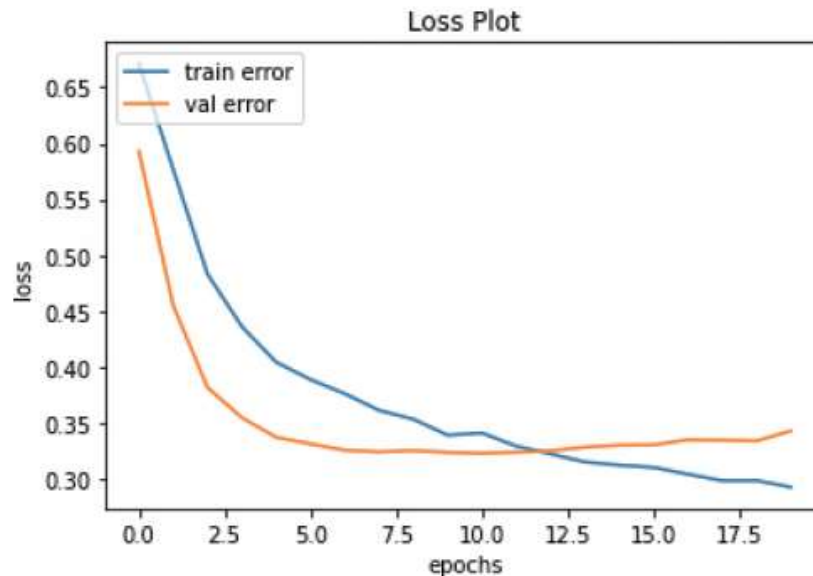
그림 8-14 드롭아웃



# Dropout

# 신경망 모델 구축

```
model = tf.keras.Sequential()  
model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(10000,)))  
model.add(tf.keras.layers.Dropout(0.5))  
model.add(tf.keras.layers.Dense(16, activation='relu'))  
model.add(tf.keras.layers.Dropout(0.5))  
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```





# 데이터 증강 방법 (*data augmentation*)

- 데이터 증강(data augmentation) 방법은 소량의 훈련 데이터에서 많은 훈련 데이터를 뽑아내는 방법이다.

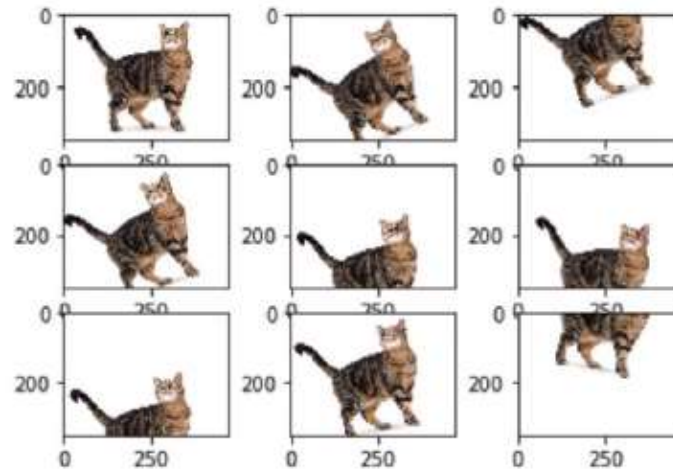


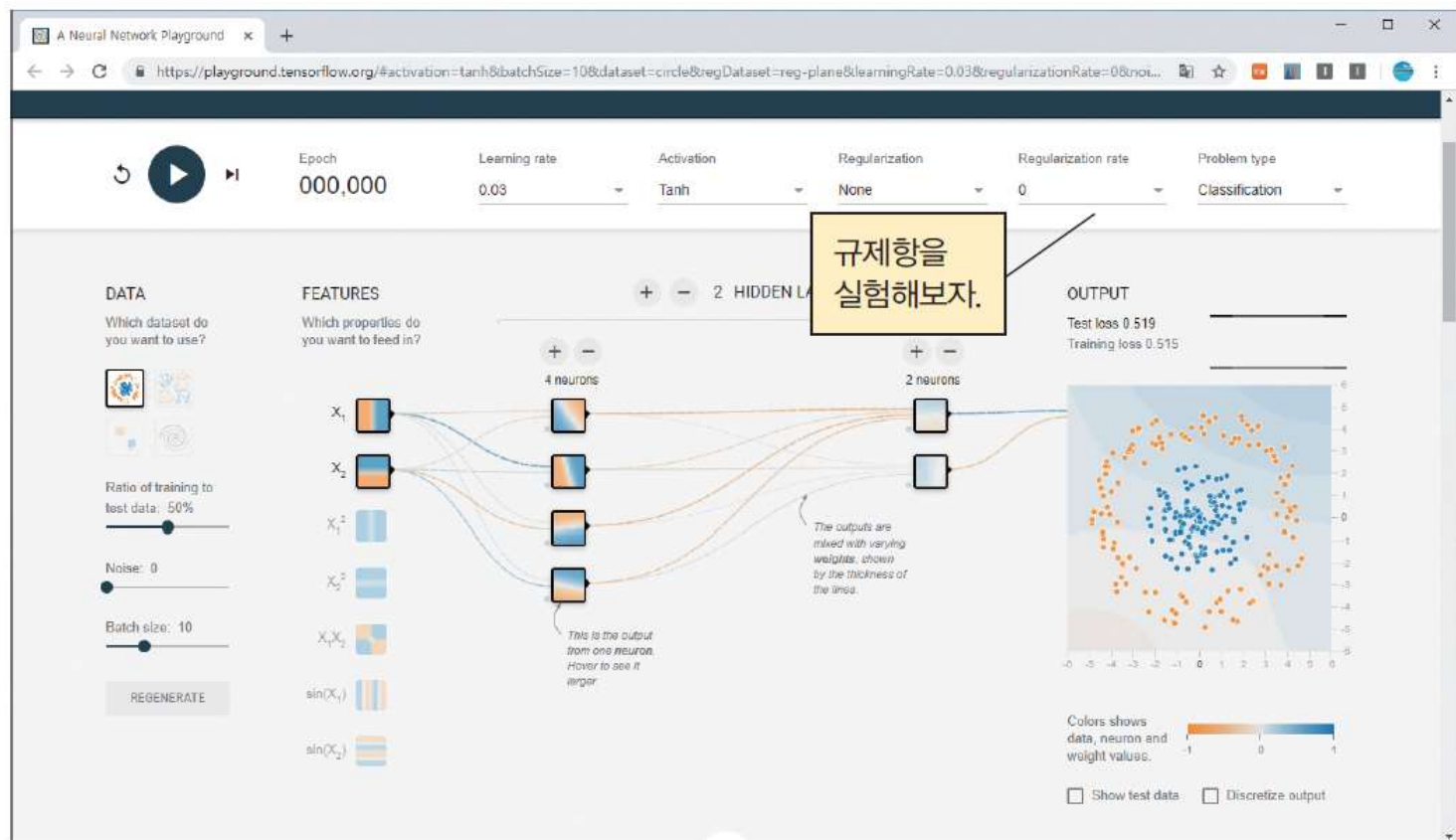
그림 8-15 데이터 증강 방법





# Lab: 배치 크기, 학습률, 규제항

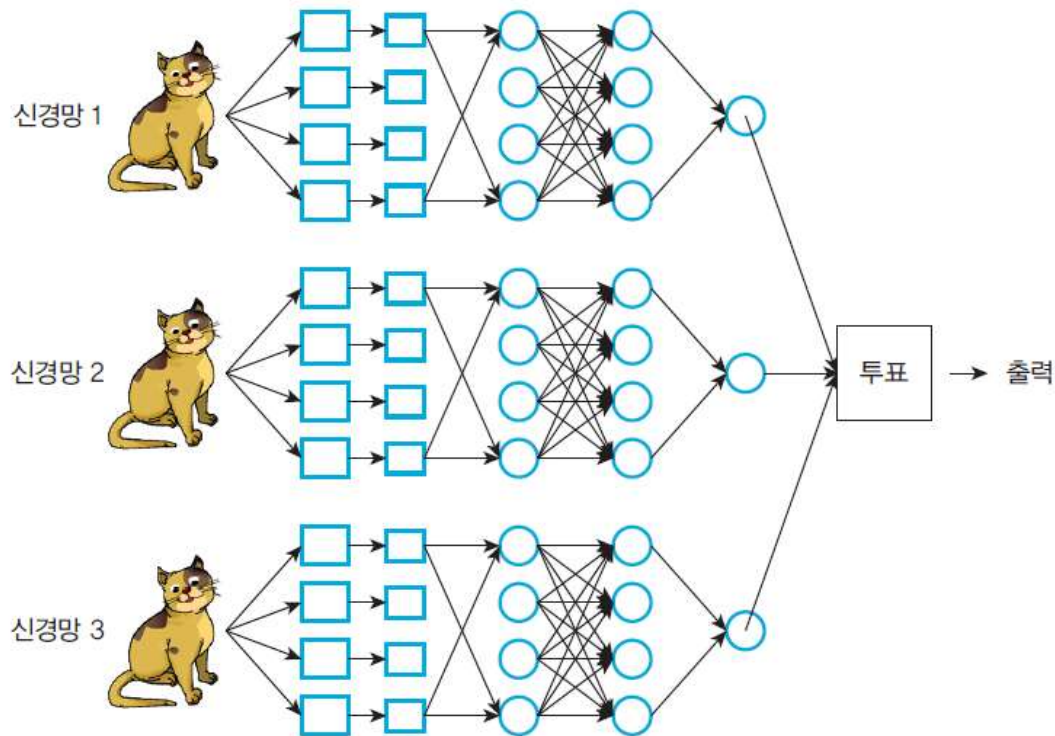
- <https://playground.tensorflow.org>





# 앙상블

- 앙상블(ensemble)은 여러 전문가를 동시에 훈련시키는 것과 같다. 이 방법은 동일한 딥러닝 신경망을  $N$ 개를 만드는 것이다.
- 각 신경망을 독립적으로 학습시킨 후에 마지막에 합치는 것이다.





## 예제: *MNIST* 필기체 숫자 인식

- 우리는 7장에서 MNIST 숫자를 MLP로 인식해본 경험이 있다. 동일한 데이터 세트에 대하여 이번에는 심층 신경망을 사용해보자. 얼마나 정확도가 증가할까?

```
import matplotlib.pyplot as plt
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```



# 예제: MNIST 필기체 숫자 인식

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

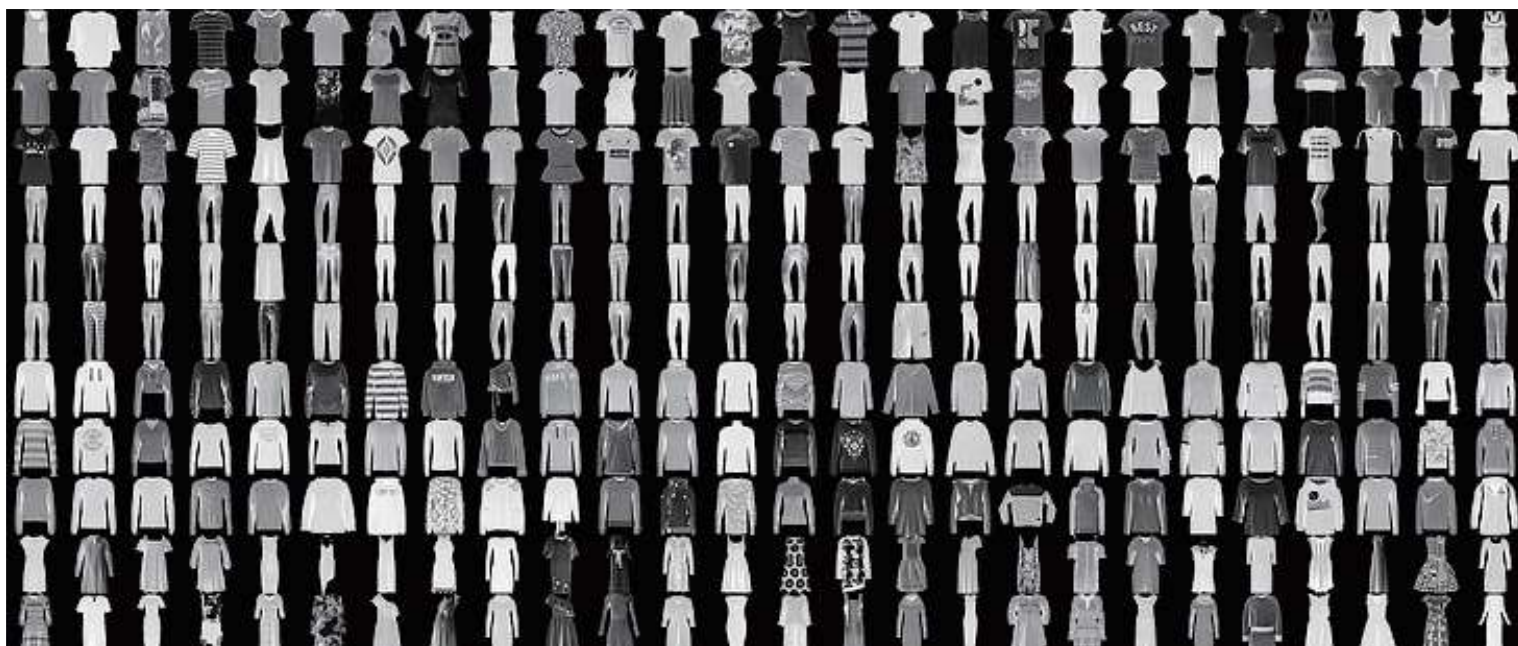
```
model.fit(x_train, y_train, epochs=5)  
model.evaluate(x_test, y_test)
```

```
Epoch 1/5  
60000/60000 [=====] - 7s 116us/sample - loss: 0.2205 - acc: 0.9348  
Epoch 2/5  
60000/60000 [=====] - 7s 110us/sample - loss: 0.0969 - acc: 0.9700  
Epoch 3/5  
60000/60000 [=====] - 7s 109us/sample - loss: 0.0678 - acc: 0.9785  
Epoch 4/5  
60000/60000 [=====] - 6s 108us/sample - loss: 0.0529 - acc: 0.9834  
Epoch 5/5  
60000/60000 [=====] - 7s 108us/sample - loss: 0.0428 - acc: 0.9859  
10000/10000 [=====] - 0s 43us/sample - loss: 0.0645 - acc: 0.9795
```



# 예제: 패션 아이템 분류

- 이번 절에서는 텐서플로우 튜토리얼에 나오는 패션 아이템을 심층 신경망으로 분류하는 코드를 살펴보자.  
([https://www.tensorflow.org/tutorials/keras/basic\\_classification](https://www.tensorflow.org/tutorials/keras/basic_classification))





# 데이터 세트

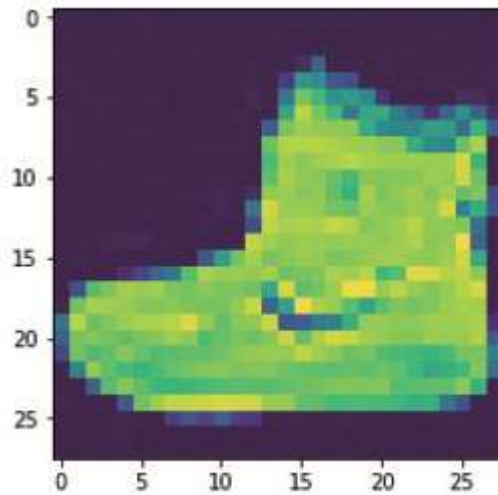
- 이미지는 28x28 크기이고 픽셀 값은 0과 255 사이의 값이다. 레이블 (label)은 0에서 9까지의 정수로서 패션 아이템의 범주를 나타낸다.

레이블	범주
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot



# 하나의 데이터

- `plt.imshow(train_images[0])`





## 예제: 패션 아이템 분류

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models

fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

plt.imshow(train_images[0])

train_images = train_images / 255.0
test_images = test_images / 255.0
```





## 예제: 패션 아이템 분류

```
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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

model.fit(train_images, train_labels, epochs=5)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('정확도:', test_acc)
```

```
10000/10000 [=====] - 0s 32us/sample - loss: 0.3560 -
acc: 0.8701
정확도: 0.8701
```



# 예제: 타이타닉 생존자 예측하기



생존자를 예측해봅시다. 어떤 부류의 사람들의 생존률이 높았을까요?  
우리는 어떤 속성을 이용하여 이것을 예측할 수 있을까요?



Second



Third



Crew





# 라이브러리 적재

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
```



# 학습 데이터 다운로드

# 데이터 세트를 읽어들인다.

```
train = pd.read_csv("train.csv", sep=',')
```

```
test = pd.read_csv("test.csv", sep=',')
```

# 필요없는 컬럼을 삭제한다.

```
train.drop(['SibSp', 'Parch', 'Ticket', 'Embarked', 'Name',\n            'Cabin', 'PassengerId', 'Fare', 'Age'], inplace=True, axis=1)
```

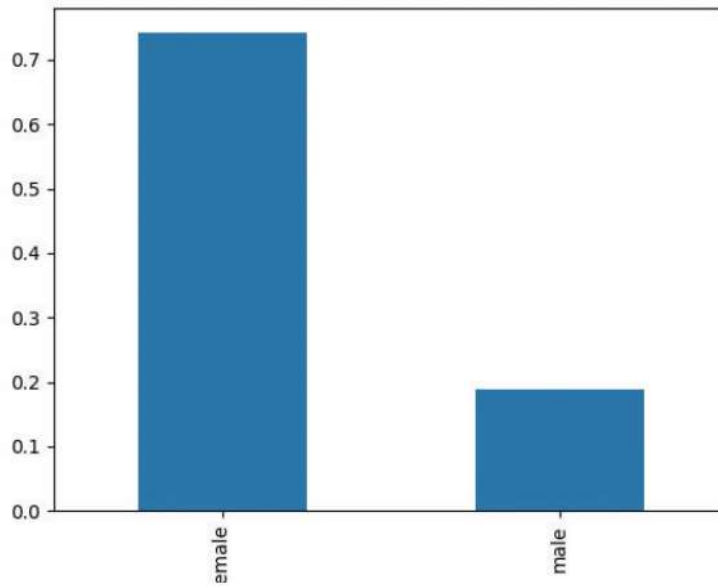
# 결손치가 있는 데이터 행은 삭제한다.

```
train.dropna(inplace=True)
```



# 시각화

```
>>> df = train.groupby('Sex').mean()["Survived"]  
>>> df.plot(kind='bar')  
>>> plt.show()
```



여성의 생존률이 높은  
것을 알 수 있네요!





# 하스 데이터 정제

```
>>> train.drop(['SibSp', 'Parch', 'Ticket', 'Embarked', 'Name',\n                'Cabin', 'PassengerId', 'Fare', 'Age'], inplace=True, axis=1)
```

```
>>> train.head()
```

	Survived	Pclass	Sex
0	0	3	male
1	1	1	female
2	1	3	female
3	1	1	female
4	0	3	male



# 범주형 데이터를 정수로 변환

```
# 기호를 수치로 변환한다.  
for ix in train.index:  
    if train.loc[ix, 'Sex']=="male":  
        train.loc[ix, 'Sex']=1  
    else:  
        train.loc[ix, 'Sex']=0
```

```
# 2차원 배열을 1차원 배열로 평탄화한다.  
target = np.ravel(train.Survived)
```

```
# 생존여부를 학습 데이터에서 삭제한다.  
train.drop(['Survived'], inplace=True, axis=1)
```

```
train = train.astype(float) # 최근 소스에서는 float 형태로 형변환하여야
```

컴퓨터는 숫자만 처리할 수 있다. 딥러닝은 0부터 1 사이의 실수만 처리 가능

교과서 소스에 추가



# 케라스 모델 구축

# 케라스 모델을 생성한다.

```
model = tf.keras.models.Sequential()  
model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(2,)))  
model.add(tf.keras.layers.Dense(8, activation='relu'))  
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

# 케라스 모델을 컴파일한다.

```
model.compile(loss='binary_crossentropy',  
              optimizer='adam',  
              metrics=['accuracy'])
```

# 케라스 모델을 학습시킨다.

```
model.fit(train, target, epochs=30, batch_size=1, verbose=1)
```





# 실행결과 : 모델로 test data에 대한 예측

...

Epoch 29/30

891/891 [=====] - 1s 753us/sample - loss: 0.4591 - acc: 0.7677

Epoch 30/30

891/891 [=====] - 1s 753us/sample - loss: 0.4547 - acc: 0.7789

약 78% 정확도

- [DIY] test data에 대한 생/사 예측은?



# Summary

- DNN(deep neural networks)에서 사용하는 학습 알고리즘이다. DNN은 MLP(다층 퍼셉트론)에서 은닉층의 개수를 증가시킨 것이다.
- 예전의 MLP에서는 인간이 영상의 특징을 추출하여서 신경망에 제공하였다. DNN에서는 특징 추출 자체도 학습으로 수행할 수 있다.
- 그래디언트 소실 문제란 은닉층이 신경망에 많이 추가되면 손실 함수의 그래디언트가 0에 가까워지고 따라서 학습이 되지 않는 현상이다. 시그모이드 함수가 범인이었다. 그래디언트 소실 문제를 해결하기 위하여 DNN에서는 활성화 함수로 **ReLU 함수**를 많이 사용한다.
- 교차 엔트로피(cross entropy)를 손실 함수로 많이 사용한다. 교차 엔트로피는 2개의 확률분포 간의 거리를 측정한 것이다.
- **가중치의 초기값은 작은 난수로 결정**되어야 한다. 가우시안 분포의 작은 난수를 사용하는 것이 좋다.
- 온라인 학습과 배치 학습의 중간에 있는 방법이 **미니 배치(mini batch)**이다. 이 방법에서는 훈련 데이터를 작은 배치들로 분리시켜서 하나의 배치가 끝날 때마다 학습을 수행하는 방법이다.



# Q & A

