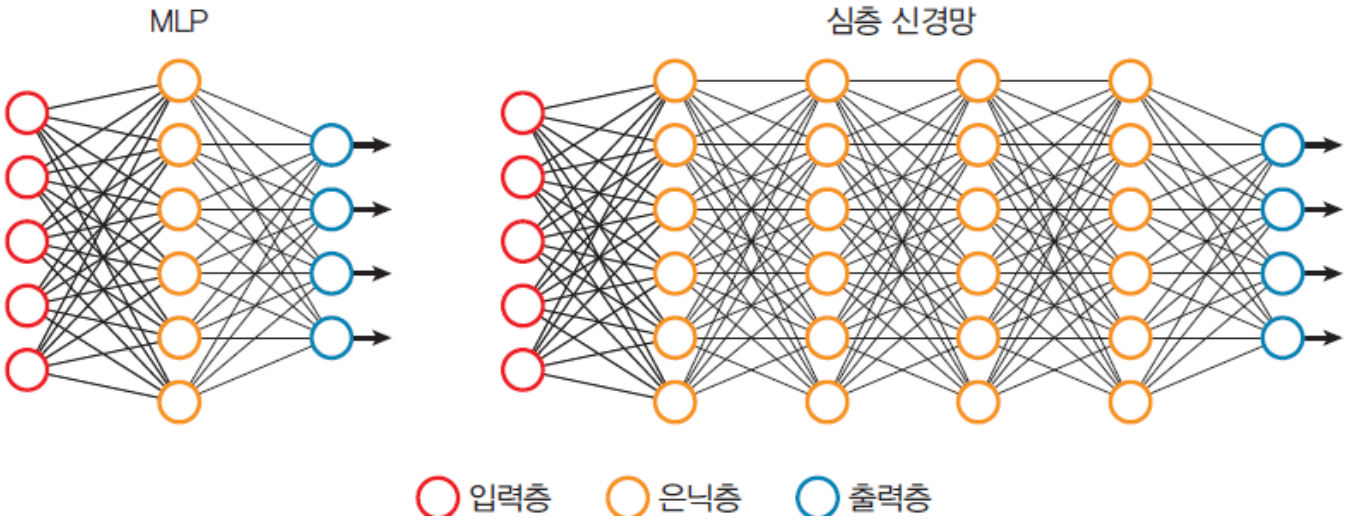
심층신경망

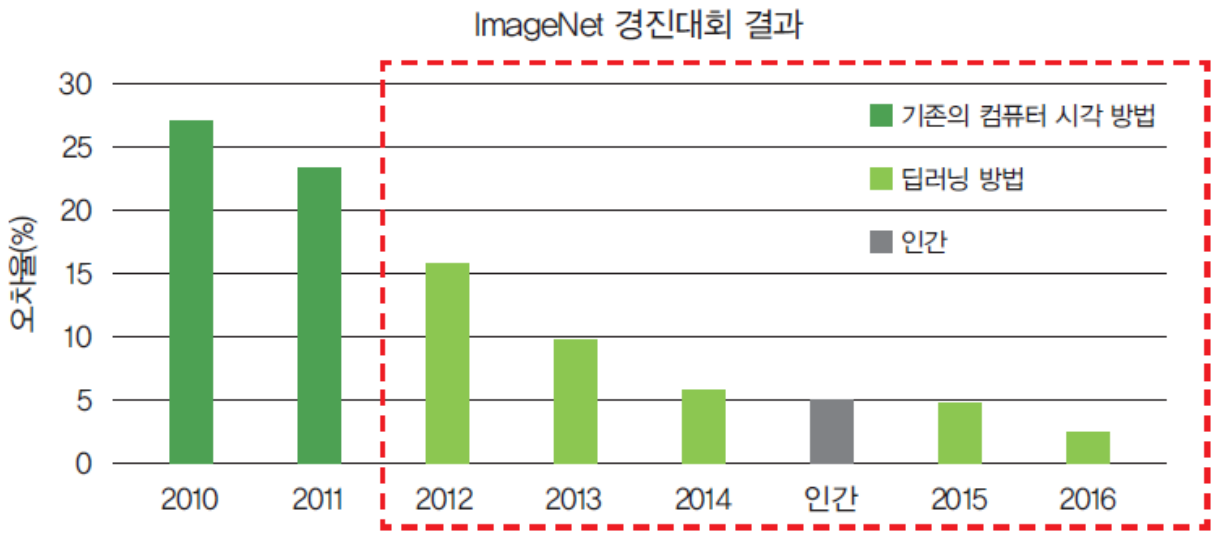
심층신경망

- 심층 신경망(DNN: Deep Neural Networks) = MLP + 은닉층 개수를 증가
- 은닉층을 하나만 사용하는 것이 아니고 여러 개를 사용
- 컴퓨터 시각, 음성 인식, 자연어 처리, 소셜 네트워크 필터링, 기계 번역 등에 적용되어서 인간 전문가에 필적하는 결과를 얻고 있다.



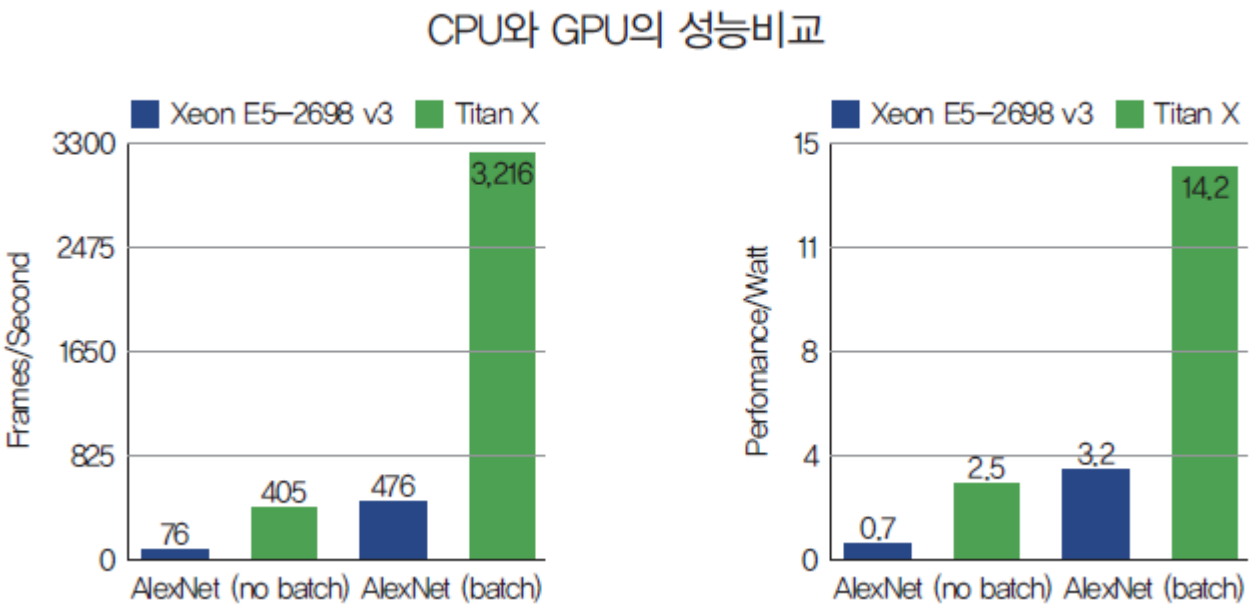
MLP의 문제점 해결

- 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
- 2012년, AlexNet이 다른 머신러닝 방법들을 큰 차이로 물리치고 ImageNet 경진대회에서 우승



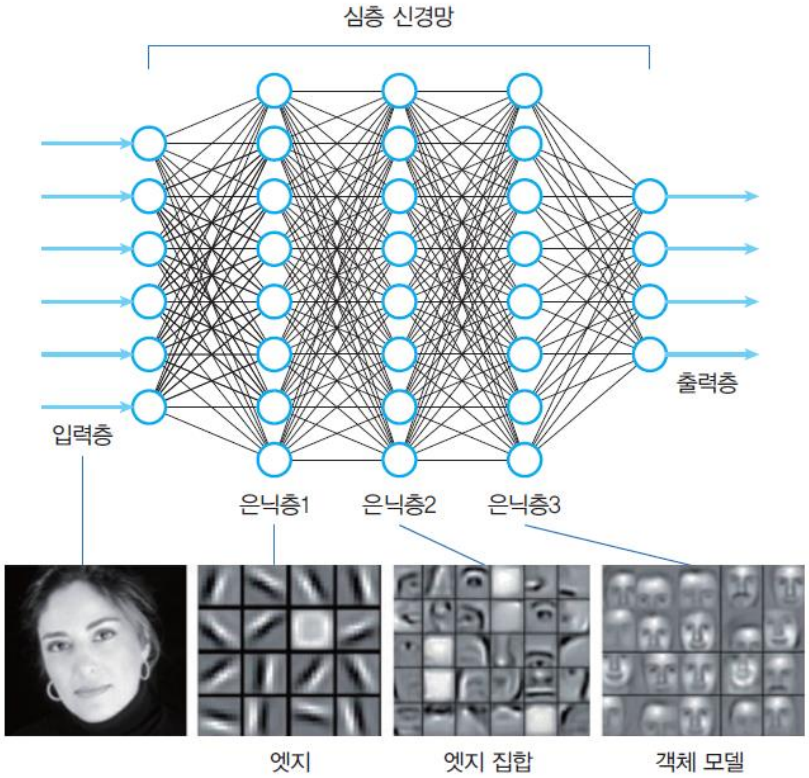
GPU의 도움

- DNN의 학습 속도는 상당히 느리고 계산 집약적이기 때문에 학습에 시간과 자원이 많이 소모.
- 최근 GPU(Graphic Processor) 기술이 엄청나게 발전하면서 GPU가 제공하는 데이터 처리 기능을 딥러닝에 사용



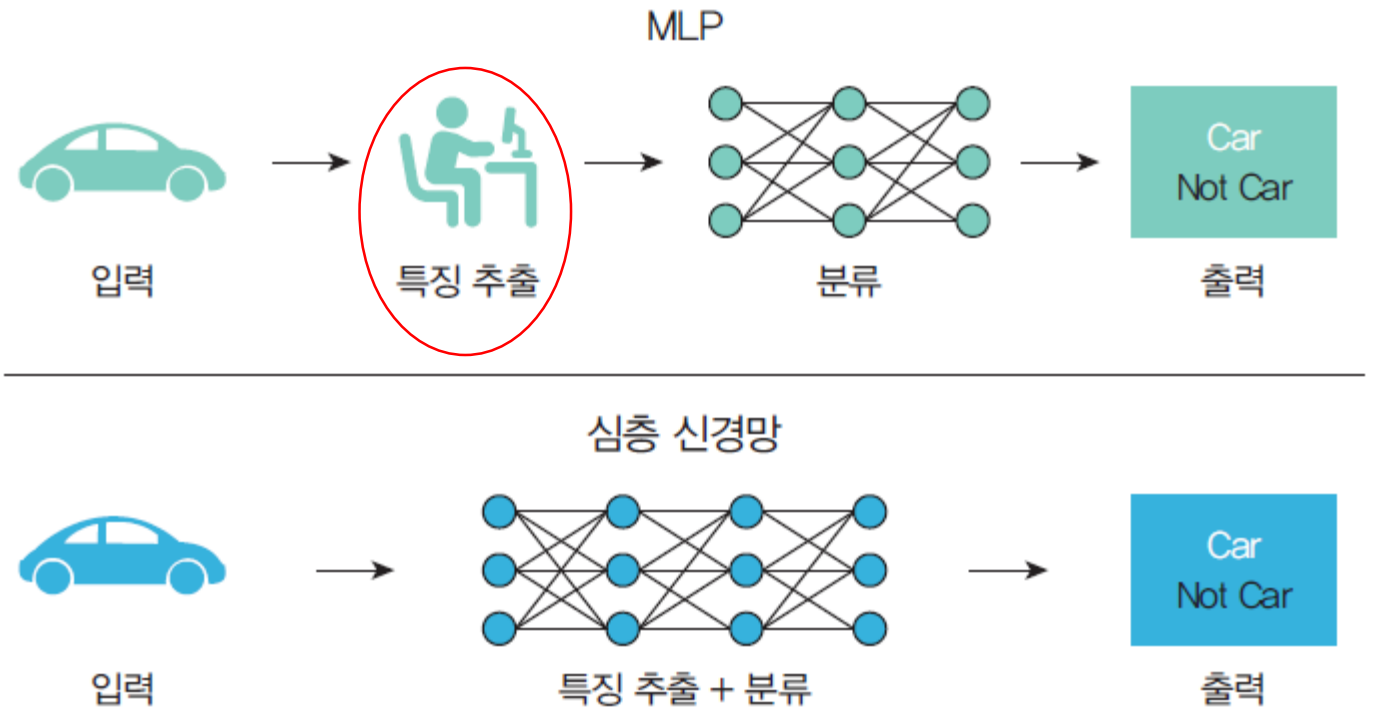
은닉층의 역할

- 한 개의 은닉층으로는 충분한 특징을 추출할 수 없다.
- DNN에서는 여러 개의 은닉층 중에서 앞단은 경계선(에지)과 같은 저급 특징들을 추출하고 뒷단은 코너와 같은 고급 특징들을 추출한다.



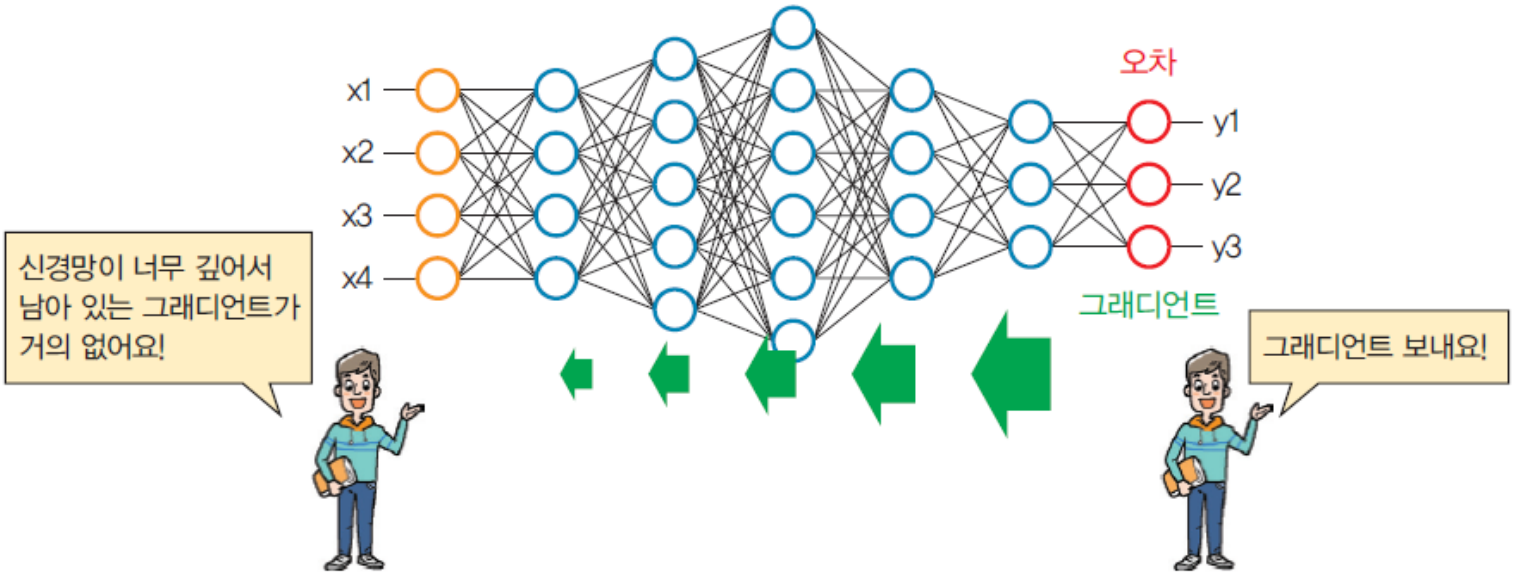
MLP vs DNN

- MLP에서는 인간이 영상의 특징을 추출. 그러나 DNN에서는 학습으로 수행.



그래디언트 소실 문제

- 심층 신경망에서 그래디언트가 전달되다가 점점 0에 가까워지는 현상



그래디언트 소실 문제

- 원인은 시그모이드 활성화 함수
- 시그모이드 함수의 특성상, 아주 큰 양수나 음수가 들어오면 출력이 포화되어서 기울기가 거의 0이 된다. 그래디언트가 너무 작아지면 학습이 효과적으로 수행되지 못한다. 즉 가중치와 바이어스 값들이 업데이트 되지 못한다.

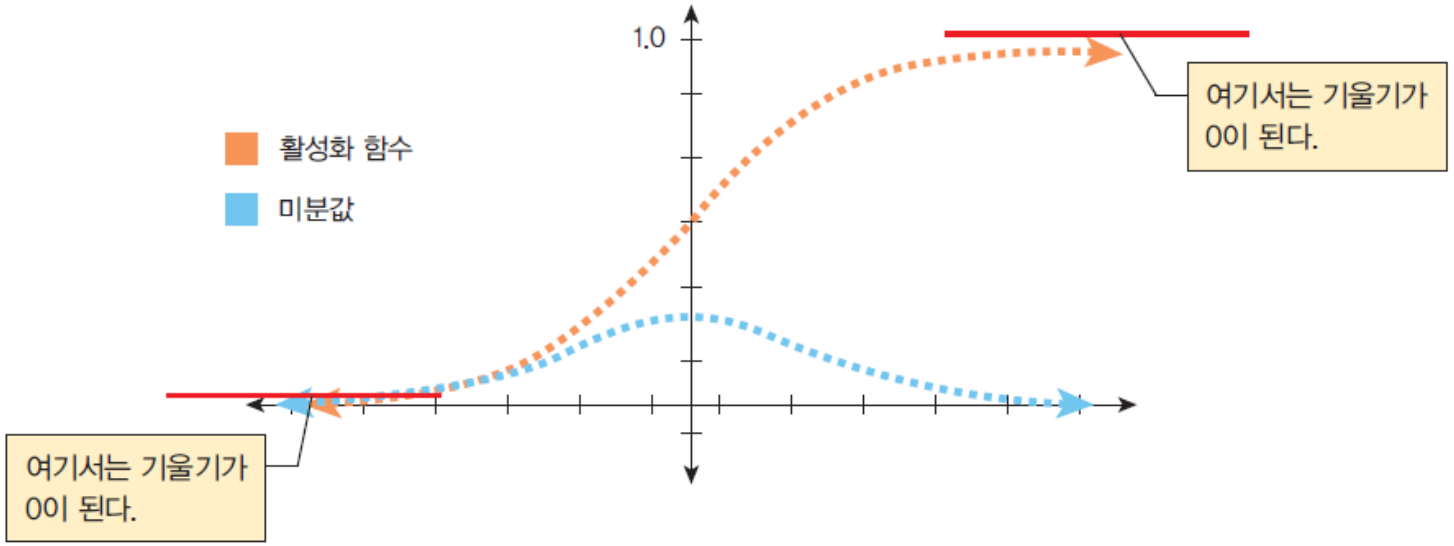
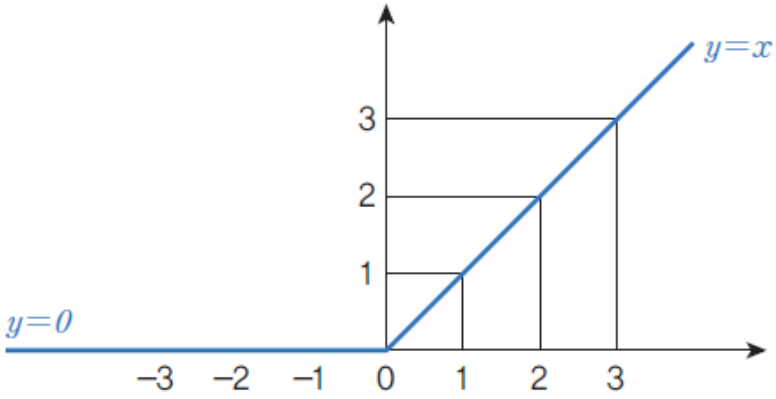


그림 8-5 시그모이드 함수와 그래디언트 소실

새로운 활성화 함수

- ReLU 함수 사용
- 장점 : 계산이 간단. 빠르게 계산. 그래디언트 소실문제가 완화된다.

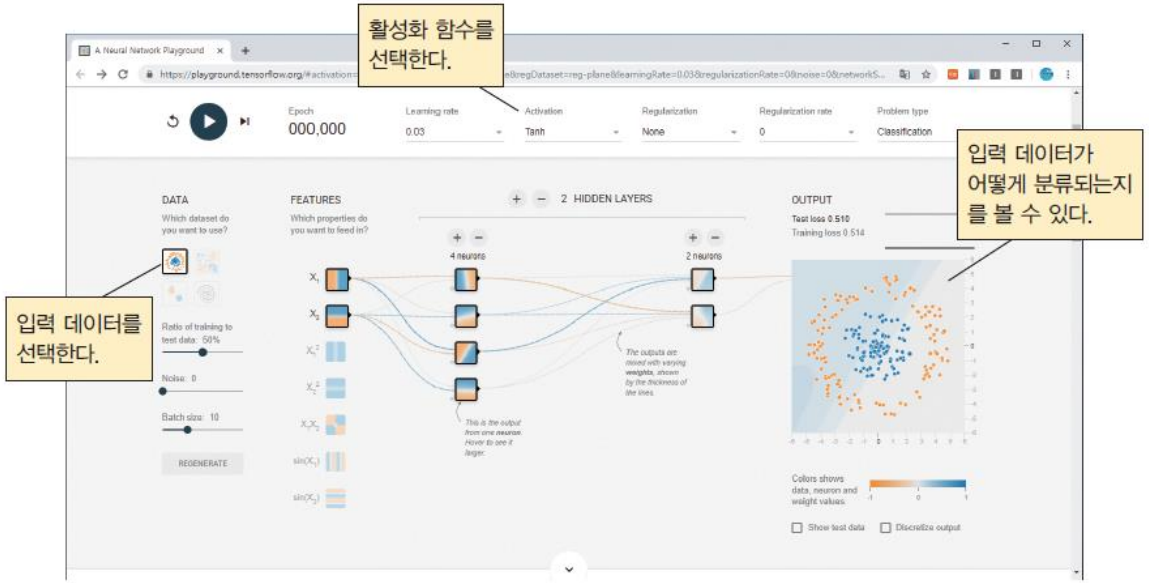


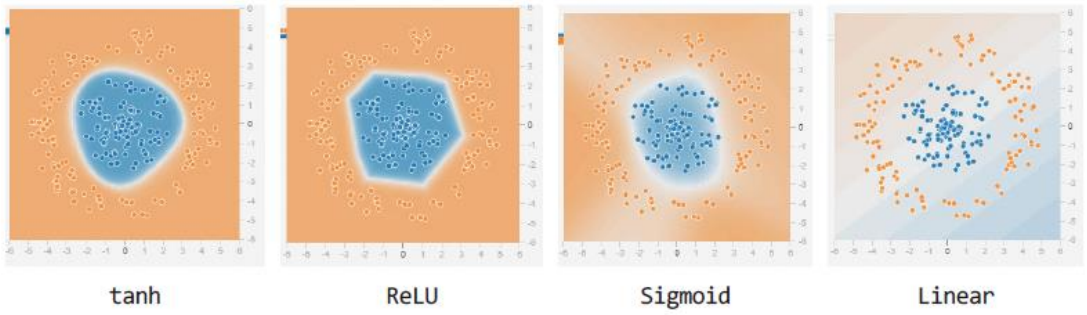
$$f(x)=\begin{cases} x & (x>0) \\ 0 & (x\leq 0) \end{cases}$$

그림 8-7 ReLU 함수

Lab: 활성화 함수 실험

- https://playground.tensorflow.org





손실 함수 선택 문제

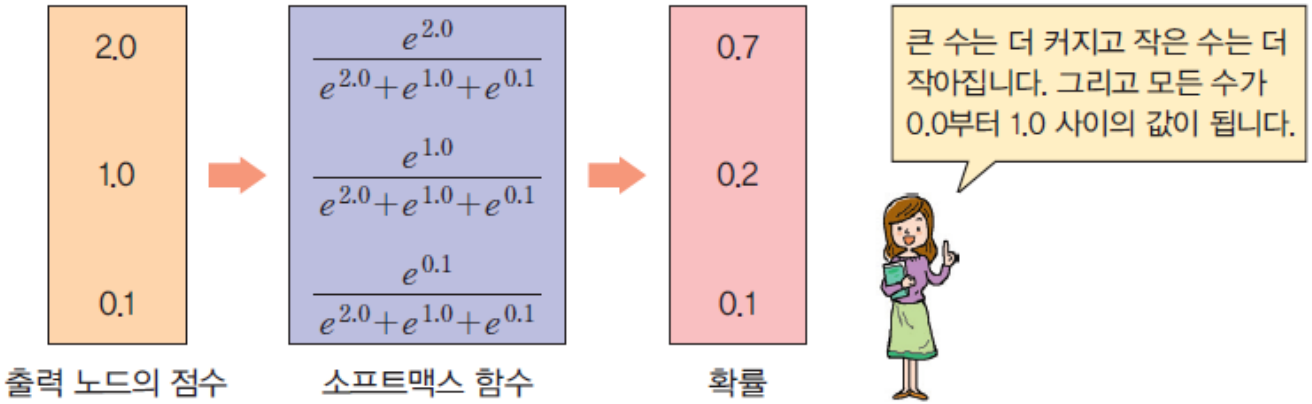
- 회귀 문제 – 평균 제곱 오차(Mean Squared Error: MSE) 사용

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

- 분류 문제 – 교차 엔트로피 함수 사용 – 뒤에서 소개

소프트맥스(softmax) 활성화 함수

- 전체 값을 합하면 1.0이 되는 확률값을 사용하는 함수

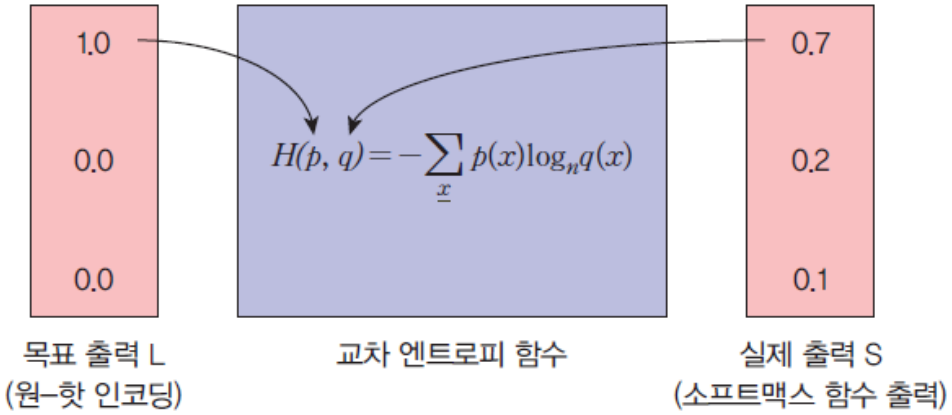


교차 엔트로피 손실 함수

- 2개의 확률 분포 p, q 간의 거리를 측정한 것(p는 목표 출력, q는 실제 출력)
- 목표 출력의 확률 분포는 원-핫 인코딩(one-hot encoding)을 사용

$$H(p, q) = -\sum_{x} p(x) \log_n q(x)$$

- 실제 출력이 얼마나 목표값에 가까운지를 알 수 있다.



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

완벽하게 일치한다면

- 목표 출력과 실제 출력이 완벽하게 일치한다면 교차 엔트로피는 0

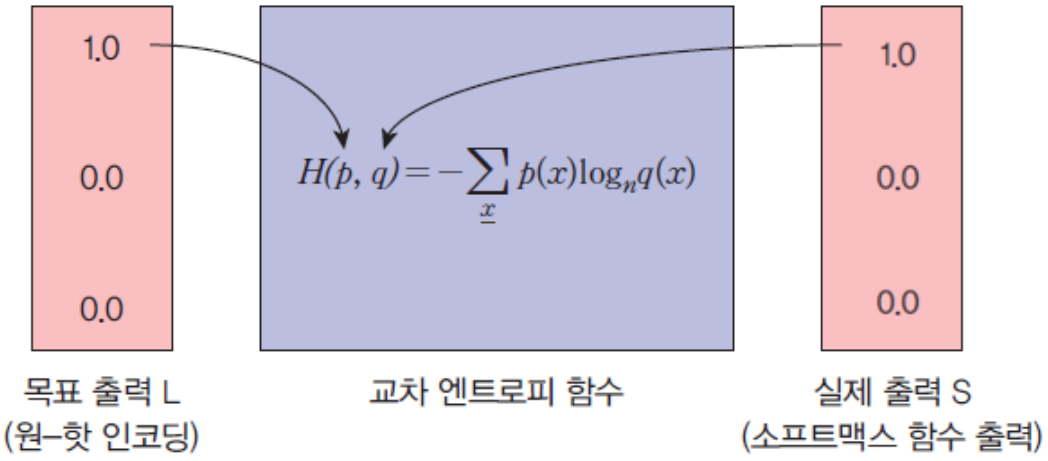


그림 8-9 교차 엔트로피 함수

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

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

실제출력은 소프트 맥스 함수,
목표 출력은 원-핫 인코딩사용

$$H(p, q) = -\sum_{x} p(x) \log_n q(x)$$ p=목표 출력, q=실제 출력

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

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

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

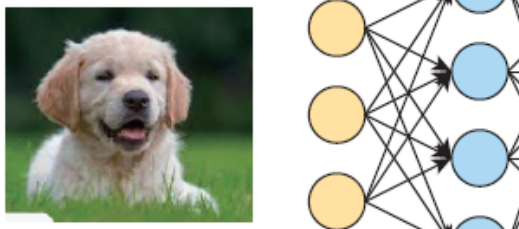
▷ 따라서 3개 샘플의 평균 교차 엔트로피 오류는 (0.51 + 0.51 + 1.20) / 3 = 0.74가 된다.

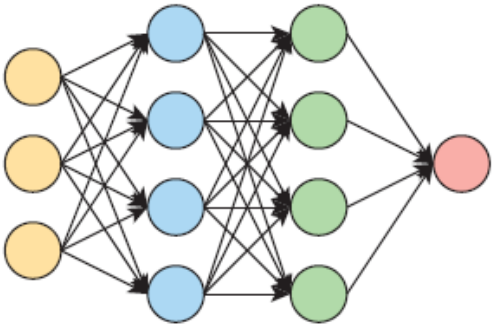
- 목표 값에 해당하는 실제 출력이 얼마나 목표값에 가까운지를 알 수 있다.

케라스에서 손실함수

1. BinaryCrossentropy

- 이진 분류 문제를 해결하는 데 사용되는 손실 함수
- 예측값이 실제 레이블(0 또는 1)에서 얼마나 멀리 떨어져 있는지 측정





1 → 강아지
0 → 강아지 아님

실제 계산예

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

- n: 샘플의 개수이다.
- y: 샘플의 실제 레이블, 0 또는 1만 가능하다.
- $\hat{y}$: 샘플의 예측값, 0에서 1 사이의 값, 신경망이 예측하는 값이다.

실제 레이블 y	1	0	0	1
예측 값 ŷ	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 : $BCE4 = -(1 \cdot \log(0.9) + (1-1) \cdot \log(1-0.9))$

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

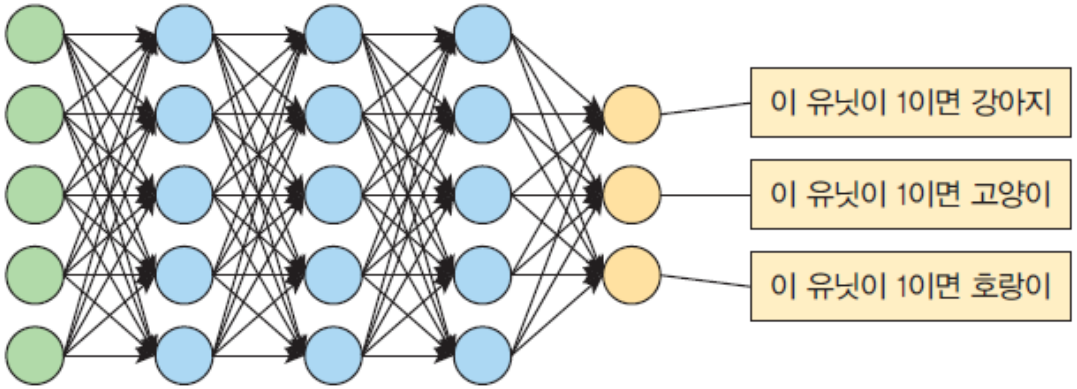
```
y_true = [  [1],   [0],   [0],   [1]]
y_pred = [[0.8], [0.3], [0.5], [0.9]]
bce = tf.keras.losses.BinaryCrossentropy()
print(bce(y_true, y_pred).numpy())
```

```
0.3445814
```

케라스에서 손실함수

2. CategoricalCrossentropy

- 다중 분류 문제에서 사용
- 정답 레이블은 원-핫 인코딩으로 제공(예. 입력 이미지를 “강아지(1,0, 0)”, “고양이(0, 1, 0)”, “호랑이(0, 0, 1)” 중의 하나로 분류)



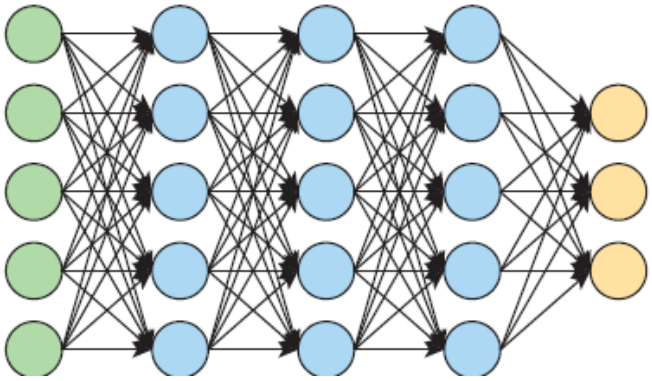

```
y_true = [[0.0, 1.0, 0.0], [0.0, 0.0, 1.0], [1.0, 0.0, 0.0]] # 고양이, 호랑이, 강아지
y_pred = [[0.6, 0.3, 0.1], [0.3, 0.6, 0.1], [0.1, 0.7, 0.2]]
cce = tf.keras.losses.CategoricalCrossentropy ()
print(cce(y_true, y_pred).numpy ())
```

```
1.936381
```

케라스에서 손실함수

3. SparseCategoricalCrossentropy

- 마찬가지로 다중 분류 문제에서 사용
- 정답 레이블이 원-핫 인코딩이 아니고 정수로 제공(예. 정답 레이블을 0(강아지), 1(고양이), 2(호랑이)로 표시)



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

- 가중치를 0으로 시작하면 → 오차 역전파가 제대로 되지 않는다.
- 가중치를 모두 같은 값으로 두면 → 역전파되는 오차가 동일하게 되어 모든 가중치가 동일하게 변경된다. 모든 노드가 같은 일을 하게 되는 셈.
- 너무 큰 가중치는 → 그래디언트 폭발을 발생하여 학습이 수렴하지 않고 발산.

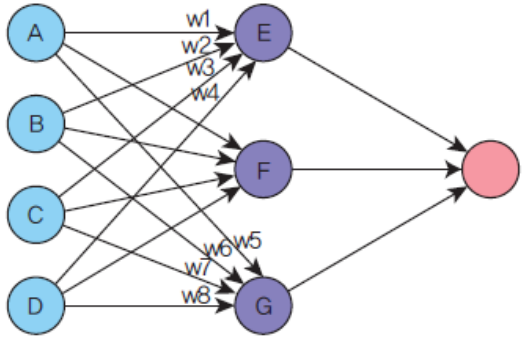


그림 8-10 가중치가 동일할 때의 문제점

- 그래서, 가중치의 초기값은 작은 난수로 해야 한다.

케라스에서 가중치 초기화 방법

- 이니셜라이즈 이용. kernel_initializer, bias_initialize

```
from tensorflow.keras import layers
from tensorflow.keras import initializers

layer = layers.Dense(
    units=64,
    kernel_initializer=initializers.RandomNormal(stddev=0.01),
    bias_initializer=initializers.Zeros()
)
```

가중치 초기화

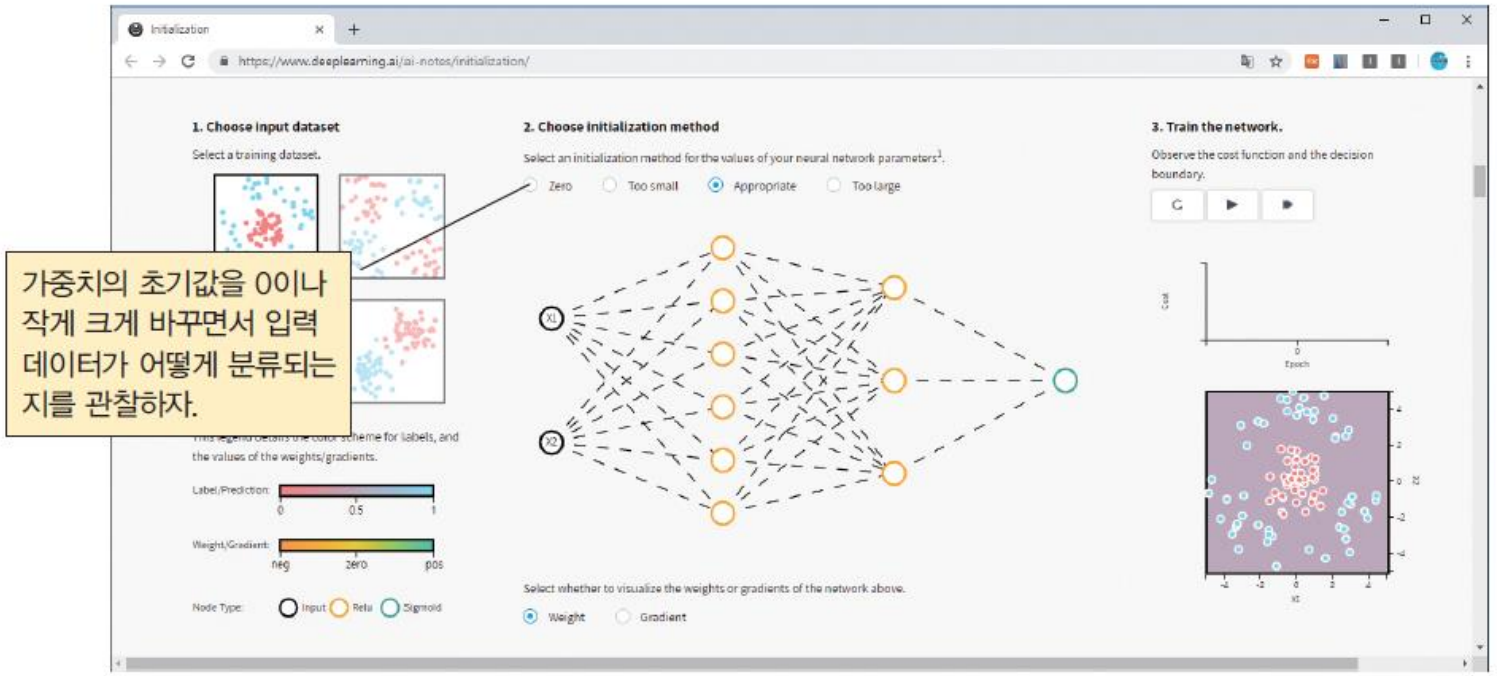
바이어스 초기화

- 문자열로 내장 이니셜라이저 사용도 가능

```
layer = layers.Dense(
    units=64,
    kernel_initializer='random_normal',
    bias_initializer='zeros'
)
```

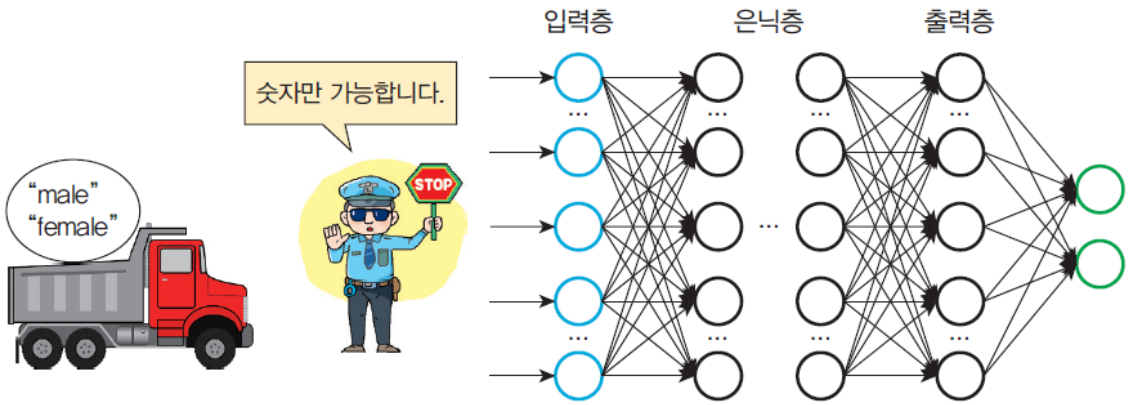
Lab: 가중치 초기화 실험

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



범주형 데이터 처리

- 신경망은 숫자만 받을 수 있다. 문자열은 숫자로 바꾸어야 한다.



- 'male'과 'female'처럼 간단한 경우에는 직접 코딩하여 변경

```
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(카테고리형) 변수 : 레이블 값을 가지는 변수(예.‘red’,’green’,’blue’)
- 머신러닝 알고리즘에서는 입력 및 출력 변수가 모두 숫자여야 한다.

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

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']]
```

- 단점. 바뀐 0, 1, 2값이 어떤 순서가 있는 것으로 오해할 수도 → 이 문제를 해결하려면 원-핫 인코딩

원-핫 인코딩(sklearn 사용)

- 단 하나의 값만 1이고 나머지는 모두 0인 인코딩 -> 아주 많이 사용
- Sklearn 라이브러리의 OneHotEncoder 클래스를 사용

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(케라스 사용)

- 케라스의 to_categorical() 호출 사용

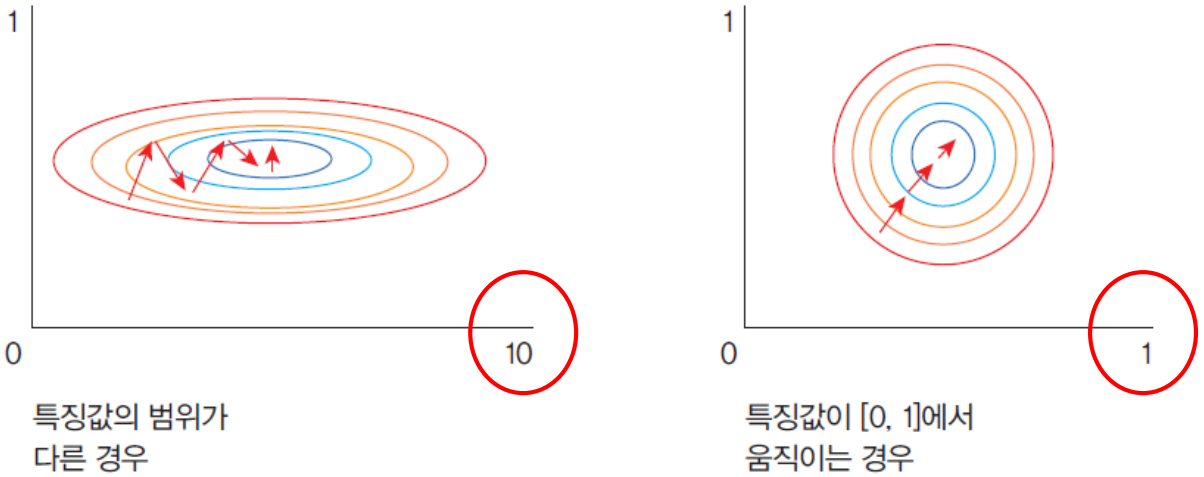
```
class_vector =[2, 6, 6, 1]

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]]
```

데이터 정규화

- 입력 x1과 x2의 서로 다른 경우(예. X1은 0 ~ 10.0, x2는 0 ~ 1.0)에는 입력과 관계되는 매개 변수도 서로 다른 범위를 가지면서 학습되기 때문에 발산하기 쉽다.



특징값의 범위가 다른 경우

특징값이 [0, 1]에서 움직이는 경우

그림 8-11 데이터 정규화의 영향

- 데이터 정규화(data normalization) : 모든 입력을 같은 범위의 값으로 만드는 것. 평균이 0이 되도록 값의 범위를 대략 -1에서 1사이로 제한.

데이터 정규화

- **정규화 값 x ′ = (x – 평균) / 표준편차**

표준편차 = sqrt(sum((값-평균)**2) / 개수)

예. 데이터 (30, 36, 52, 42) 일 때

평균 = (30 + 36 + 52 + 42) / 4= 160 /4 = 40

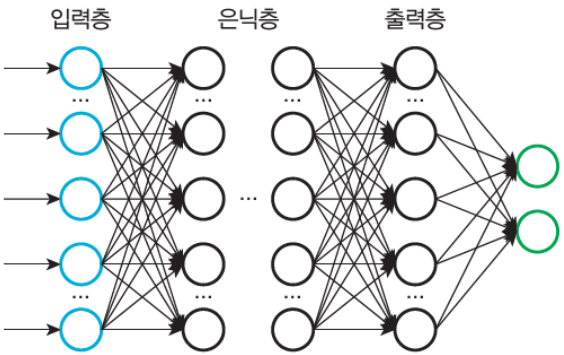
표준편차 = sqrt(((30-40)**2 + (36-40)**2 + (52-40)**2 + (42-40)**) / 4)

= sqrt((100 + 16 + 144 + 4) / 4) = sqrt(66.0) = 8.12

30세의 정규화 값 = (30 - 40.0) / 8.12 = -1.23

예제

- 사람의 나이, 성별, 연간 수입을 기준으로, 선호하는 자동차의 타입(세단 아니면 SUV)을 예측할 신경망을 만드는 경우.



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

정규화 필요

원핫인코딩

[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 클래스 사용하여 0과 1사이에 있도록 변환

```
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.  ]]
```

케라스의 데이터 정규화 방법

- Normalization 레이어를 중간에 넣어 정규화.
- adapt() 메서드로 평균과 분산을 계산

```
>>> adapt_data = np.array([[1.], [2.], [3.], [4.], [5.]], dtype=np.float32)
>>> input_data = np.array([[1.], [2.], [3.]], 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)>
```

p.320 예제를 정규화, 원핫 인코딩 해보기

과잉 적합과 과소 적합

- 과잉 적합(over fitting)
  - 지나치게 훈련 데이터에 특화되어 실제 적용시 좋지 못한 결과가 나오는 것
  - 신경망이 너무 복잡(매개변수, 은닉층, 뉴런이 많을 때)하면 발생
  - 교과서의 예제만 완벽하게 풀고 다양한 문제를 푸는 과정이 없는 경우에 해당

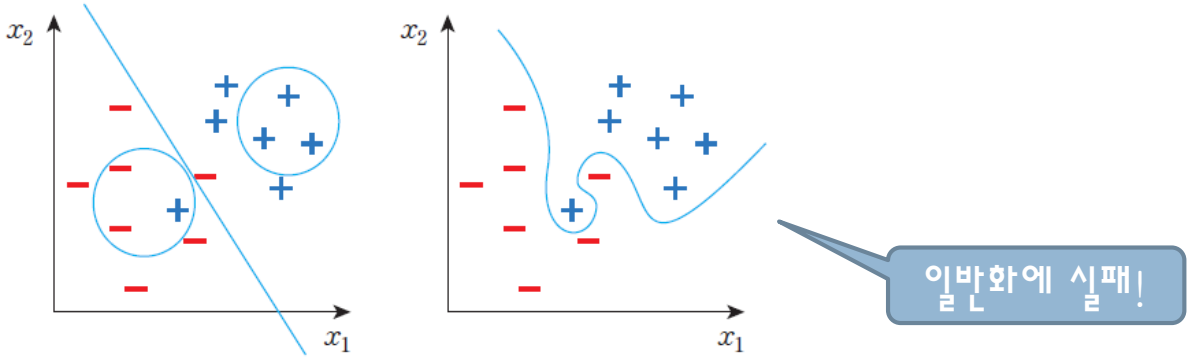
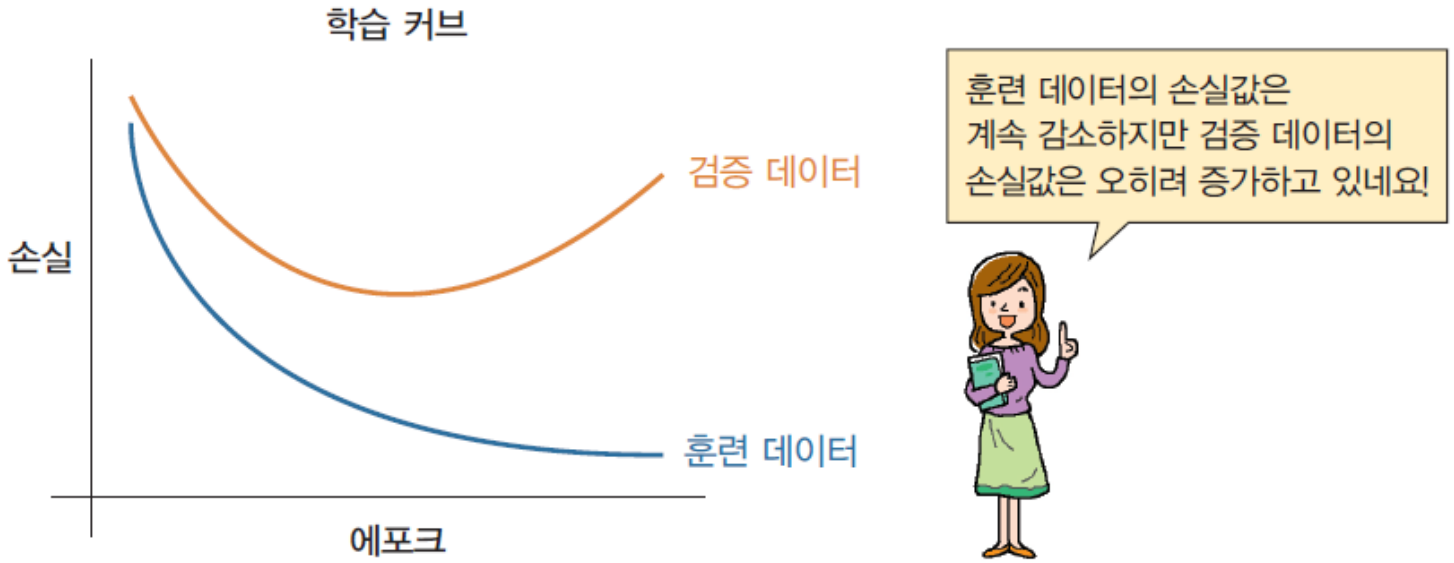


그림 8-13 과잉 적합의 예

- 과소 적합(under fitting)
  - 충분히 훈련되지 않거나 신경망 모델이 너무 단순할 때 발생

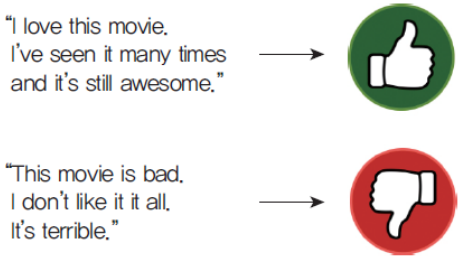
과잉 적합을 아는 방법

- 훈련이 계속되어도 검증 데이터의 손실 값이 감소하지 않는다.

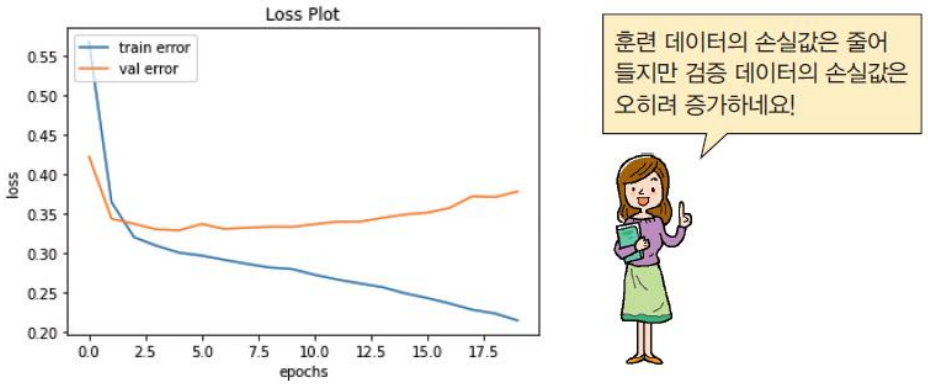


과잉 적합의 예

- IMDB : 영화에 대한 리뷰가 올라가 있는 사이트
- 영화 리뷰를 입력하면 리뷰가 긍정적(1)인지 부정적(0)인지를 파악하는 신경망을 구현해보자



- 영화 리뷰는 미리 전처리되어 정수 시퀀스로 제공되며 리뷰당 10,000 차원 벡터의 멀티-핫 인코딩으로 변환하여 입력으로 사용



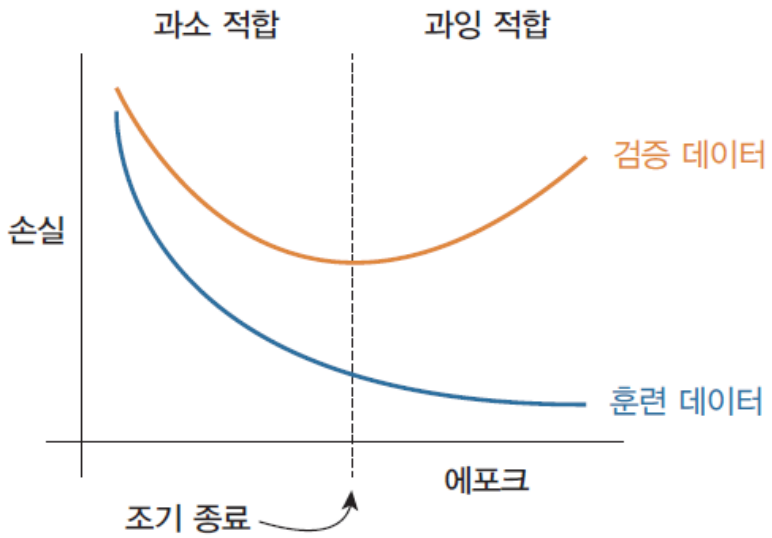
과잉 적합 방지 전략

- 가장 좋은 방법 → 많은 훈련 데이터 사용
- 훈련 데이터를 많이 확보할 수 없는 경우에는 데이터 증강(data augmentation)이나 규제(regularization) 기법을 사용

- 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.

1. 조기종료

- 검증 손실이 더 이상 감소하지 않는 것처럼 보일 때마다 훈련을 중단



2. 가중치 규제

- 가중치의 값이 너무 크면 과잉 적합이 일어난다는 사실에 근거하여 신경망의 손실함수에 가중치가 커지면 불리하도록 비용을 추가한다.

- L1 규제 : 가중치의 절대값에 비례하는 비용을 손실 함수에 추가

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

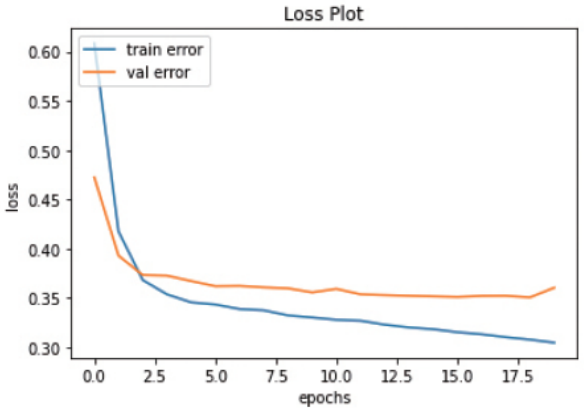
- L2 규제 : 가중치의 제곱에 비례하는 비용을 손실함수에 추가

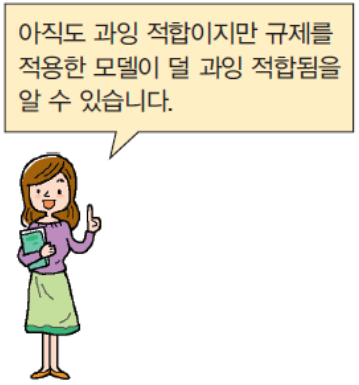
$$Loss = Cost + \lambda \sum W^2$$

2. 가중치 규제

- p.325의 과잉적합을 가중치 규제를 적용하여 해소된 결과를 그래프로 확인할 수 있다.

```
# 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'))
```





3. 드롭 아웃

- 가장 효과적이고 널리 사용하는 기법
- 학습과정에서 몇 개의 노드를 쉬게하여 다른 노드가 그 역할을 대신하는 것
  → 동일한 특징을 검출하는 전문가가 많이 생기는 효과를 발생
- 훈련 단계에서만 사용. 테스트 단계에서는 모든 노드를 사용
- 케라스에서 Dropout 레이어를 이용해서 추가

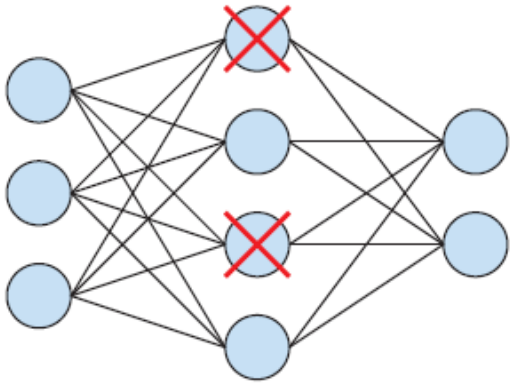


그림 8-14 드롭아웃

4. 데이터 증강 방법

- 데이터 증강(data augmentation) : 소량의 훈련 데이터에서 많은 훈련 데이터를 뽑아내는 방법
- 이미지를 좌우로 확대하거나 회전시켜서 변형된 이미지 생성

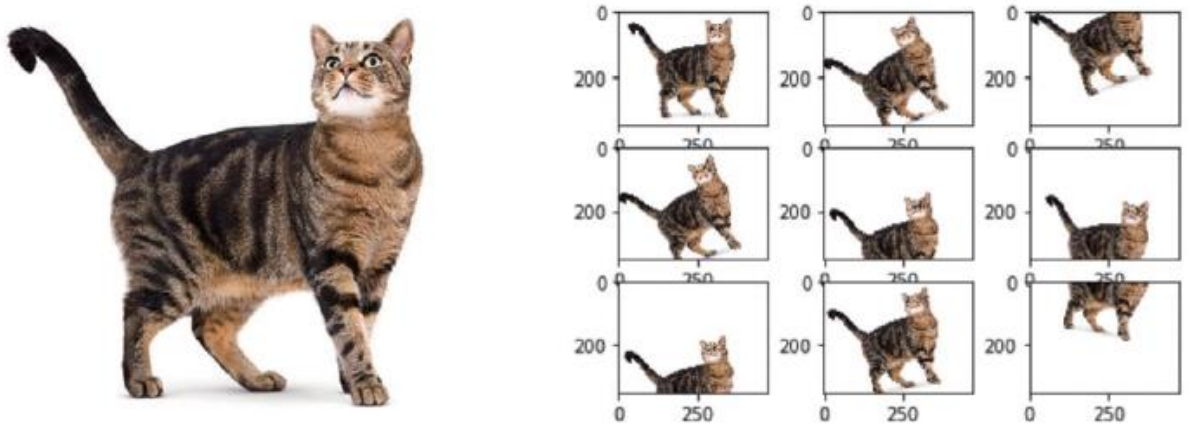
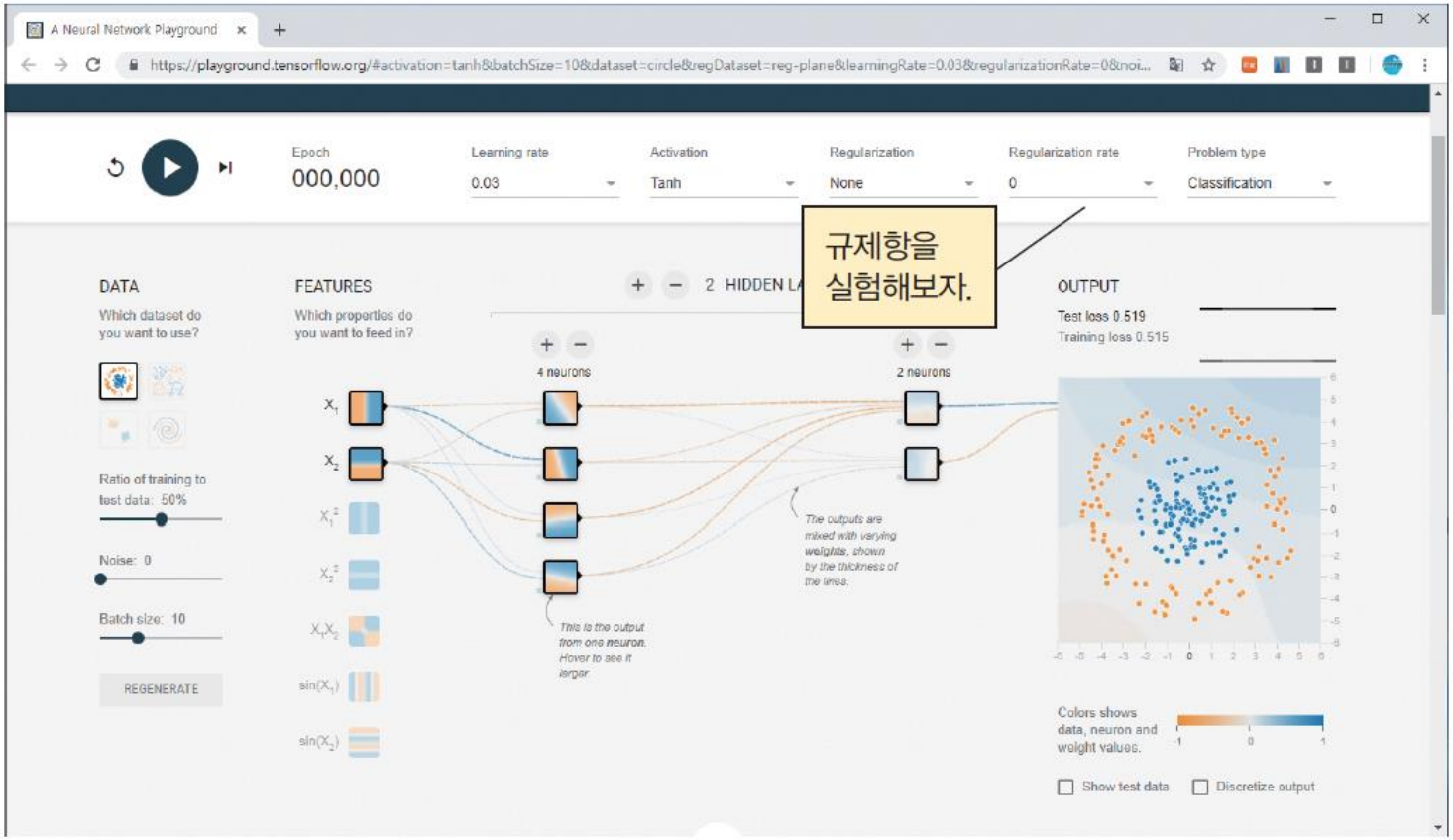


그림 8-15 데이터 증강 방법

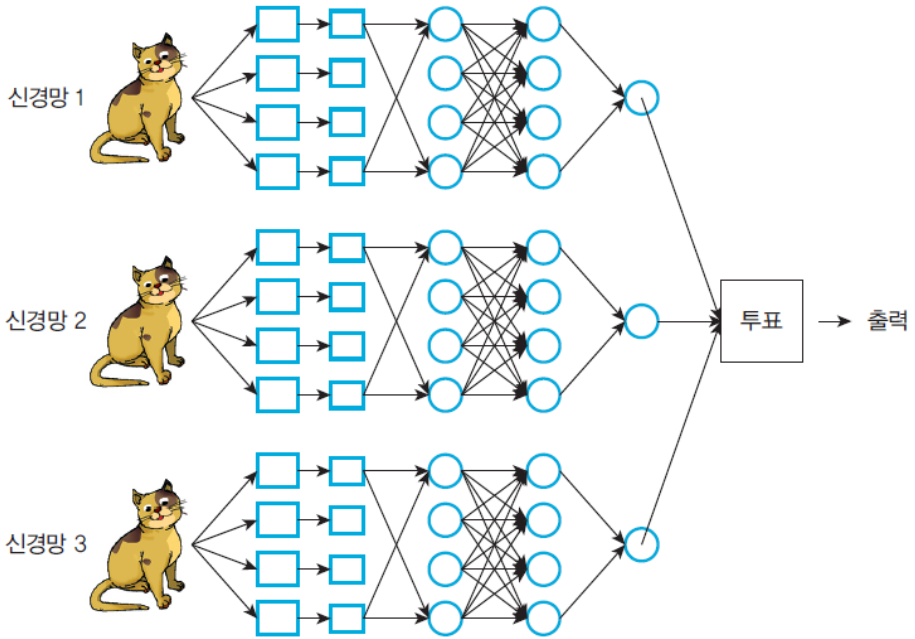
Lab: 배치 크기, 학습률, 규제항

- https://playground.tensorflow.org



앙상블(ensemble)

- 여러 전문가를 동시에 훈련시키는 것
- 동일한 딥러닝 신경망을 N개를 만들어 각 신경망을 독립적으로 학습시킨 후에 마지막에 합치는 것
- 약 2-5% 정도의 성능 향상



예제: MNIST 필기체 숫자 인식

- 7장에서 MNIST 숫자를 MLP로 인식해본 경험이 있다.(p.266)
- 동일한 데이터 세트에 대하여 이번에는 심층 신경망을 사용해보자. 얼마나 정확도가 증가할까?

예제: 패션 아이템 분류

- 패션 MNIST 데이터셋 사용
- 70,000개의 이미지 중에서 60,000개는 훈련용, 10,000개는 테스트용
- 이미지 크기 : 28x28. 픽셀 값 : 0 ~ 255 사이
- 10개 범주

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

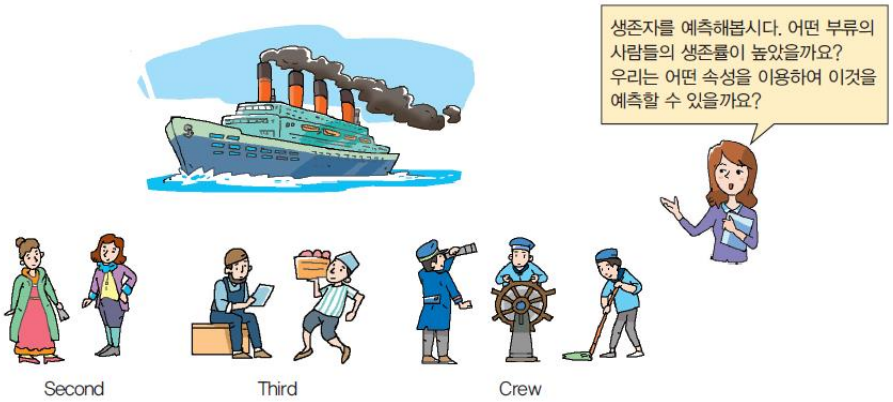
예제: 패션 아이템 분류

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

예제: 패션 아이템 분류

예제: 타이타닉 생존자 예측하기

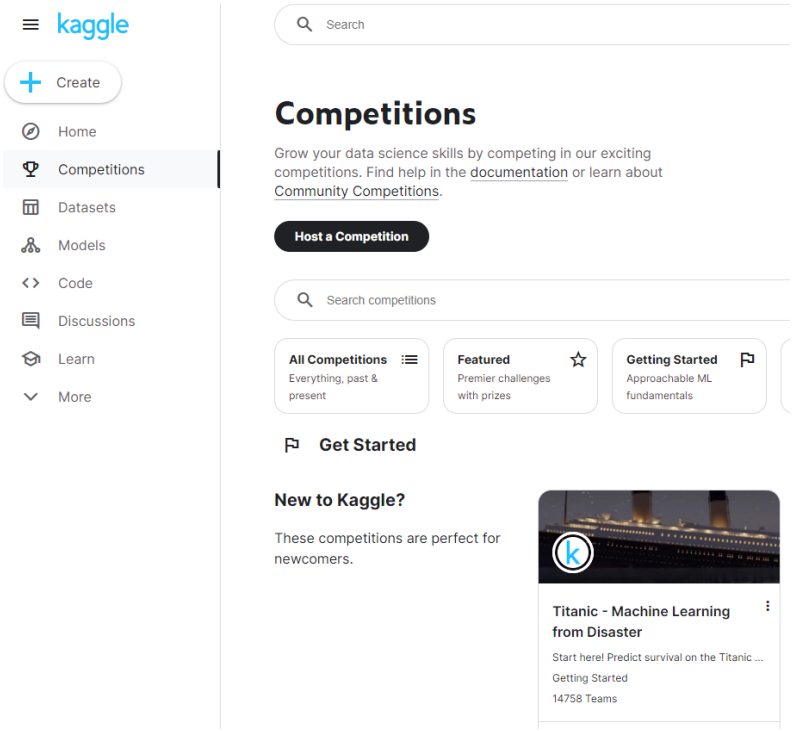
- PassengerId : 각 승객의 고유 번호
- Survived : 생존 여부(종속 변수)
  - 0 = 사망
  - 1 = 생존
- Pclass : 객실 등급 - 승객의 사회적, 경제적 지위
  - 1위 = 상위
  - 2위 = 중간
  - 세 번째 = 낮음
- Name : 이름
- Sex : 성별
- Age : 나이
- SibSp : 동반한 Sibling(형제자매)와 Spouse(배우자)의 수
- Parch : 동반한 Parent(부모) Child(자식)의 수
- Ticket : 티켓의 고유넘버
- Fare : 티켓의 요금
- Cabin : 객실 번호
- Embarked : 승선한 항
  - C = 쉘부르
  - Q = 퀸스타운
  - S = 사우샘프턴



예제: 타이타닉 생존자 예측하기

예제: 타이타닉 생존자 예측하기

- 학습(예측) 정확도를 높이기 위한 노력이 핵심 - Kaggle Competition
  - ① 속성(열) 선택을 추가하고 다양한 조합으로 도전
  - ② 학습모델 디자인을 변경



비교: 패션 아이템 분류, 타이타닉 생존자 예측하기

- **공통점** – 입력 데이터가 여러개 값
- **차이점** – 출력 개수

1. 패션 아이템
   - 10개 중에 하나를 최종 선택
   - 마지막 출력 : model.add(layers.Dense(10, activation='softmax'))
   - 손실함수 loss = 'sparse_categorical_crossentropy',

2. 타이타닉
   - 1개(생존/사망)
   - 마지막 출력 : model.add(layers.Dense(1, activation='sigmoid'))
   - 손실함수 loss = 'binary_crossentropy'