Deep Learning 1

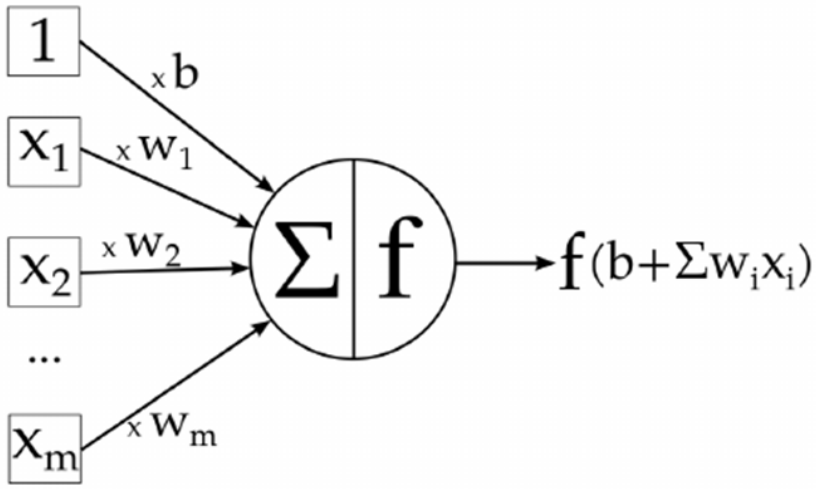
ICT Innovation Square
김 보 연

Perceptron &
Multi layered Perceptron

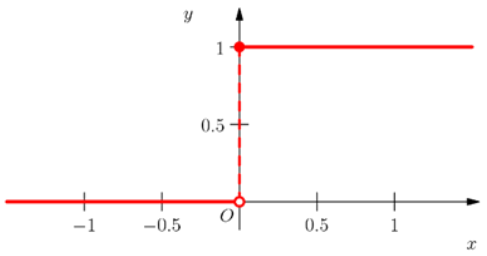
최초의 인공신경망: 퍼셉트론(Perceptron)

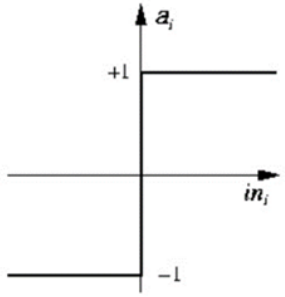
- 1958, 프랭크 로센블라트(Frank Rosenblatt)
  - Perceptron (SLP: Single Layer Perceptron)
    - ✓ 입력층과 출력층으로만 구성됨 (1층의 신경망)
    - ✓ 입력층 뉴런과 출력층 뉴런 사이 연결에 가중치(weight)있음
      - 입력층 뉴런의 작용: 입력을 pass

- Single neuron



- Activation function: step function





$$f_\Phi(\mathbf{x}) = \begin{cases} 1 & \text{if } b + \mathbf{w} \cdot \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

퍼셉트론 학습 알고리즘: 델타 룰

- Perceptron 학습 규칙: 델타 룰(delta rule)
  - Cost function(=loss function):
    - ✓ 에러가 최소일 때, 최소값을 가지는 함수

$$E = \sum_j \frac{1}{2}(t_j - y_j)^2$$ (t_i : 뉴런 j의 target value, y_i : 뉴런 j의 현재 출력)

뉴런 j에 대해 ⇒ $E_j = \frac{1}{2}(t_j - y_j)^2$

  - 학습 = 입력 뉴런 i와 출력 뉴런 j 사이의 가중치 W_{ji}조정 + b_j 조정 $y = Wx_j + b$

$$W_{ji}(t+1) = W_{ji}(t) - \eta \Delta W_{ji}(t)$$
$$= W_{ji}(t) - \eta \frac{\partial E}{\partial W(t)}$$
$$= W_{ji}(t) + \eta (t_j - y_j) x_i$$
$$= W_{ji}(t) + \eta \cdot error \cdot x_i$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial \left(\frac{1}{2}(t_j - y_j)^2\right)}{\partial w_{ji}} = \frac{\partial \left(\frac{1}{2}(t_j - y_j)^2\right)}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}}$$
$$= -(t_j - y_j) \frac{\partial y_j}{\partial w_{ji}} = -(t_j - y_j) x_i,$$

$$\frac{\partial x_i w_{ji}}{\partial w_{ji}} = x_i,$$

퍼셉트론 학습 알고리즘: 델타 룰

- Perceptron 학습 규칙: 델타 룰(delta rule)
  - Cost function(=loss function):
    - ✓ 에러가 최소일 때, 최소값을 가지는 함수

$$E = \sum_j \frac{1}{2}(t_j - y_j)^2$$

(t_i : 뉴런 j의 target value, y_i : 뉴런 j의 현재 출력)

뉴런 j에 대해 →

$$E_j = \frac{1}{2}(t_j - y_j)^2$$

  - 학습 = 입력 뉴런 i와 출력 뉴런 j 사이의 가중치 W_{ji}조정 + b_j 조정

$$b_j(t+1) = b_j(t) - \eta \Delta b_j(t)$$

$$= b_j(t) - \eta \frac{\partial E}{\partial b(t)}$$

$$= b_j(t) + \eta (t_j - y_j) \cdot 1$$

$$= b_j(t) + \eta \cdot error$$

$$y = W x_j + b$$

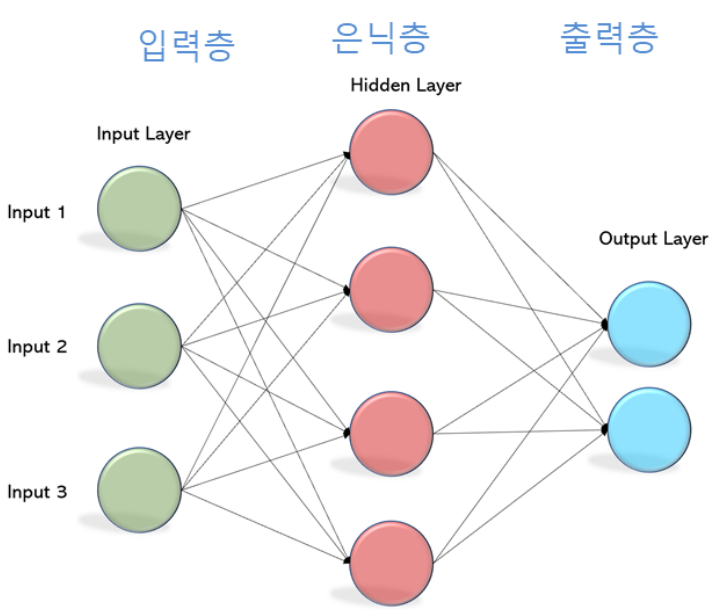
$$\frac{\partial E}{\partial b_j} = \frac{\partial \left(\frac{1}{2}(t_j - y_j)^2\right)}{\partial b_j} = \frac{\partial \left(\frac{1}{2}(t_j - y_j)^2\right)}{\partial y_j} \frac{\partial y_j}{\partial b_j}$$

$$= -(t_j - y_j) \frac{\partial y_j}{\partial b_j} = -(t_j - y_j) \cdot 1$$

다층 퍼셉트론

- 구조: 은닉층(hidden-layer)을 가지는 퍼셉트론
- 활성화 함수: sigmoid 함수
- 학습 알고리즘: 역전파 알고리즘

MLP 구조

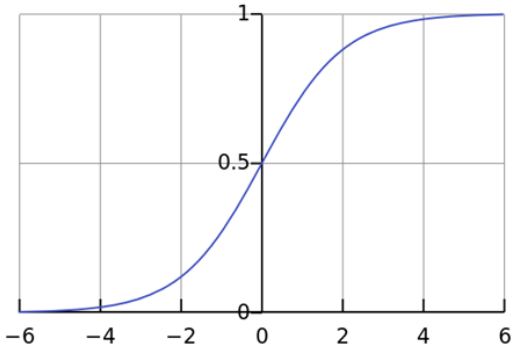


Transformation

$$f(s) = \frac{1}{1+e^{-s}}$$

$$s = \sum w \cdot x$$

$$A = \frac{1}{1+e^{-x}}$$

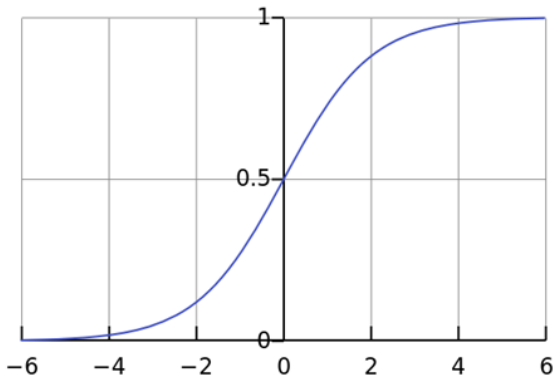


MLP

- 활성화 함수(Activation functions)

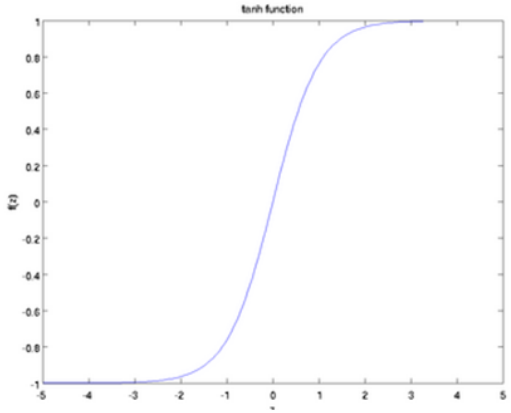
[1] Sigmoid function (0~1)

$$A = \frac{1}{1+e^{-x}}$$



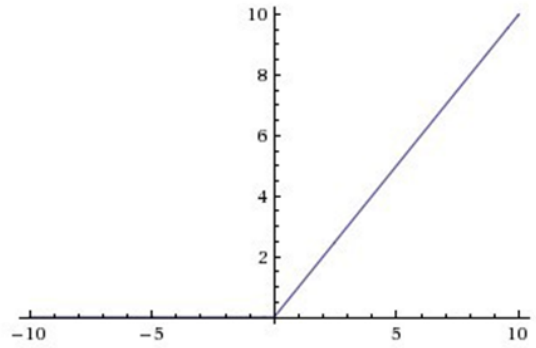
[2] tanh function (-1, 1)

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



[3] ReLU function (0, +)

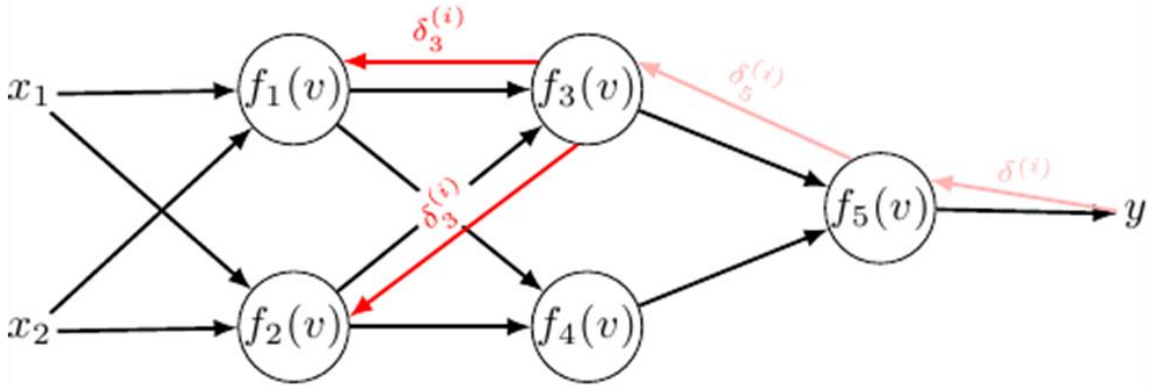
$$A(x) = \max(0,x)$$



MLP

- 학습 규칙: (오류) 역전파 학습(EBP, Error Back Propagation)

$$\delta_3^{(i)} = \delta_5 \cdot g'(r_3) \cdot w_{3,5}$$



$$\delta_j = \begin{cases} \sigma'(\zeta_j)(t_j - y_j) & \text{if j is an output unit} \\ \sigma'(\zeta_j)\sum_k \delta_k w_{jk} & \text{if j is a hidden unit} \end{cases}$$

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

$$\frac{\partial \sigma}{\partial z}(z) = \sigma(z)(1-\sigma(z))$$

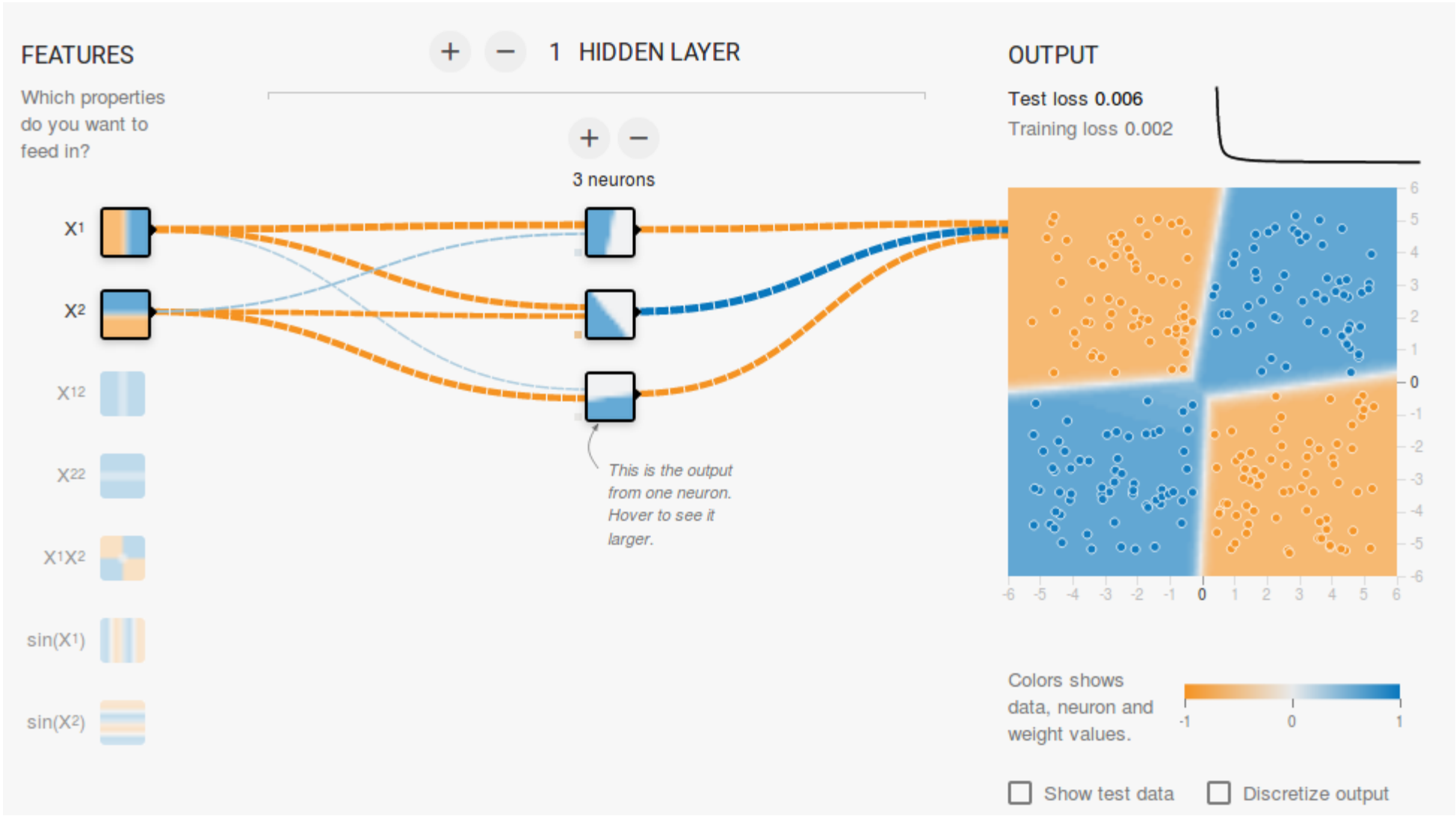
$$\begin{aligned} W_{ji}(t+1) &= W_{ji}(t) + \eta \Delta W_{ji}(t) \\ &= W_{ji}(t) - \eta \frac{\partial E}{\partial W(t)} \\ &= W_{ji}(t) + \eta(t_j - y_j)\delta(\zeta) \cdot x_i \\ &= W_{ji}(t) + \eta \cdot error\ (1-s) \cdot s \cdot x_i \end{aligned}$$

오류 역전파 학습 알고리즘

- 역전파 학습 알고리즘
  1. 모든 층에 있는 가중치를 임의의 수로 초기화한다
  2. Feed-forward :
     - 학습 데이터(레이블 된)를 입력층에서 입력받아 은닉층을 통해 출력층까지 feed-forward
  3. 에러 역전파 및 가중치 조정:
     - 출력층에서의 에러를 사용하여, 출력층과 바로 앞의 은닉층 사이의 가중치 조정
     - 앞 단계의 은닉층으로 이동하면서 입력층에 도달할 때까지 가중치 수정 반복
       - 출력층의 오류를 역으로 전파함
  4. 사전에 목표한 정확도에 도달할 때까지 단계 2~5를 반복함

은닉층의 수와 문제 유형의 관계

http://playground.tensorflow.org/



딥러닝의 발전 요인

1. 혁신적인 알고리즘

① 경사감소 소멸을 줄일 수 있는 활성화 함수의 사용: ReLU

② 컨볼루션 신경망(CNN, Convolution neural network)

③ 심층신뢰망 (DBN: Deep Belief Network)

④ 규제화: 드롭아웃(Dropout) 알고리즘

- ✓ 과적합(overfitting) 문제를 해결

2. 하드웨어의 발전

- GPU의 사용, 행렬과 벡터 연산이 빠르고 효율적

신경망의 레이어 구성 방법

(1) 레이어 구성 방법 1 Sequential API

```
model = keras.Sequential()
# Input Layer & hidden layer 1
model.add(Dense(units=3, activation='                , input_dim=2))
# hidden layer 2
model.add(Dense(units=2, activation='                ))
# Output Layer
model.add(Dense(units=1, activation='                ))
```

신경망의 레이어 구성 방법

(2) 레이어 구성 방법 2

Sequential API

```
model2 = keras.Sequential()
# Input Layer
model2.add( keras.Input( shape=(2,) ) )
# hidden layer 1
model2.add(Dense(units=3, activation=          ))
# hidden layer 2
model2.add(Dense(units=2, activation=          ))
# Output Layer
model2.add(Dense(units=1, activation=          ))
```

신경망의 레이어 구성 방법

(3) 레이어 구성 방법 3

Sequential API

```
model3 = keras.Sequential(
    [
        layers.Dense(3, activation='                ', input_dim=2 ),
        layers.Dense(2, activation='                '),
        layers.Dense(1, activation='                ')
    ]
)
```

신경망의 레이어 구성 방법

(4) 레이어 구성 방법 4

Functional API

```
inputs = keras.Input(shape=(2,))
dense1 = layers.Dense(3, activation=          )(inputs)
dense2 = layers.Dense(2, activation=          )(dense1)
outputs = layers.Dense(1,activation=          )(dense2)

model4 = keras.Model(inputs=inputs, outputs=outputs, name="xor_model")
```

딥러닝에서 사용하는 문제 유형별 신경망 모델 및 Loss function

- 문제 유형별 노드 수/loss function(손실함수)

문제 유형	회귀(regression)	이진 분류 (binary classification)	다중 분류 (multi-class classification)
출력 노드 수	1	1	각 class 당 1개 One-hot encoding
출력 노드의 activation function	Linear (아무것도 적지 않음)	Sigmoid	Softmax
Loss function	Mean Squared Error(MSE)	Cross-Entropy	Cross-Entropy
Keras function name	mean_squared_error	binary_crossentropy	categorial_crossentropy

혹은 sparse_categorical_crossentropy

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

$$E = -\sum_k t_k \log y_k$$

손실함수의 종류와 설명서: https://www.tensorflow.org/api_docs/python/tf/keras/losses/

딥러닝에서 사용하는 고급 경사 하강법(Gradient Decent)

- 딥러닝에서 사용하는 고급 경사 하강법

고급경사하강법	개요	효과
확률적 경사 하강법 (Stochastic Gradient Decent)	랜덤하게 추출한 일부데이터를 사용해 더 빨리, 자주 업데이트를 하게하는 방법	속도 개선
모멘텀 (Momentum)	이전의 gradient를 사용, 관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선
네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 gradient를 계산. 불필요한 이동을 줄이는 효과	정확도 개선
아다그라드 (Adagrad)	변수의 업데이트가 많으면 학습률을 적게하여 이동 보폭을 줄이는 방법	보폭 크기 개선
알엠에스프롭 (RMSProp)	Adagrad의 보폭 민감도를 보완한 방법	보폭 크기 개선
아담 (Adam)	모멘텀과 알엠에스프롭 방법을 결합한 방법	정확도와 보폭 크기 개선

옵티마이저 설명서: https://www.tensorflow.org/api_docs/python/tf/keras/losses

회귀 문제의 출력층 노드 구현

activation 없음

```
Dense(1)
```

One-hot encoding + categorical_crossentropy

```
from sklearn.preprocessing import LabelEncoder

e = LabelEncoder().fit(y_str)
y_num = e.transform(y_str)
print(e.classes_)
print(y_num)
```

```
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
from tensorflow.keras.utils import to_categorical

y = to_categorical(y_num, num_classes=3)
print(y.shape, y[0])
```

```
(150, 3) [1. 0. 0.]
```

```
model.add( Dense(units=3, activation='softmax') )
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

sparse_categorical_crossentropy

```
y_train
```

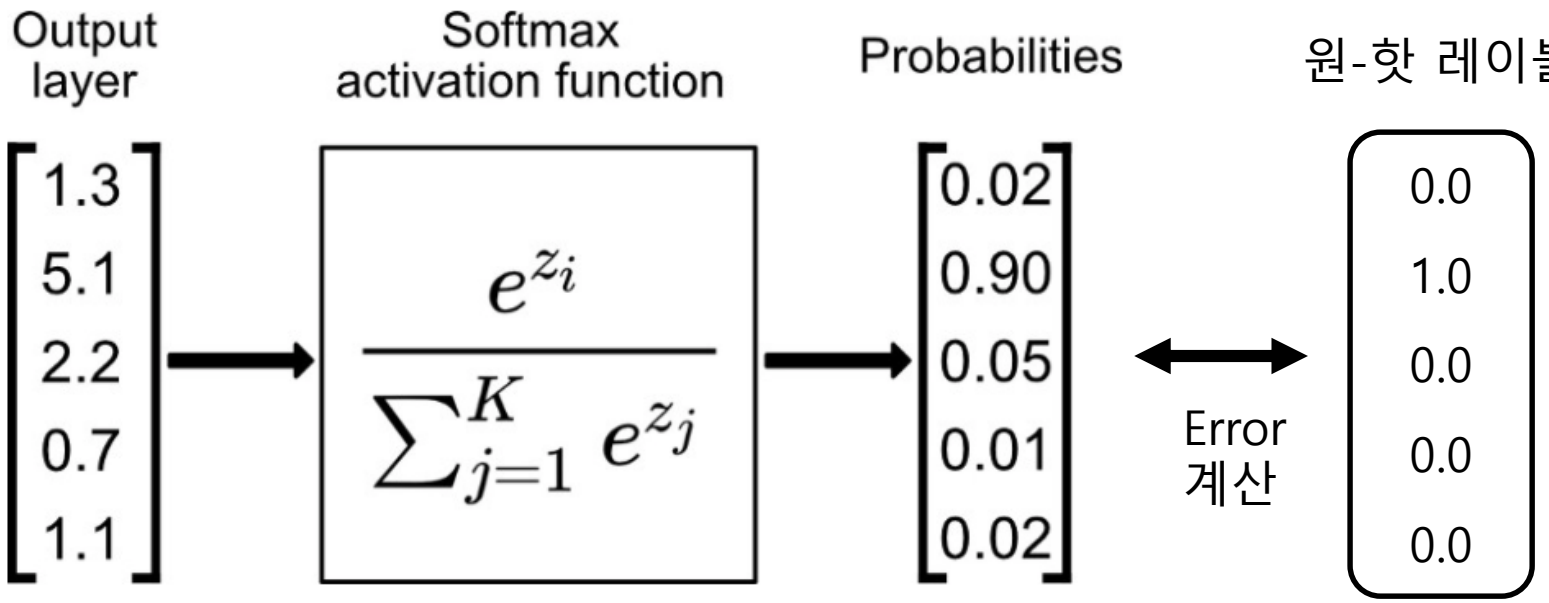
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

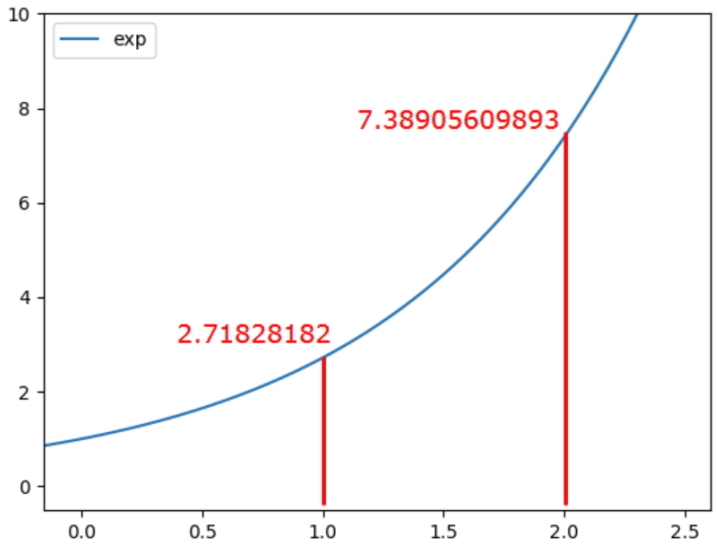
```
model.add( Dense(units=10, activation='softmax') )
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
```

다중 분류 모델의 출력층에서의 분류 클래스 구하기

```
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
```

softmax





총합이 항상 1

과적합(over fitting)

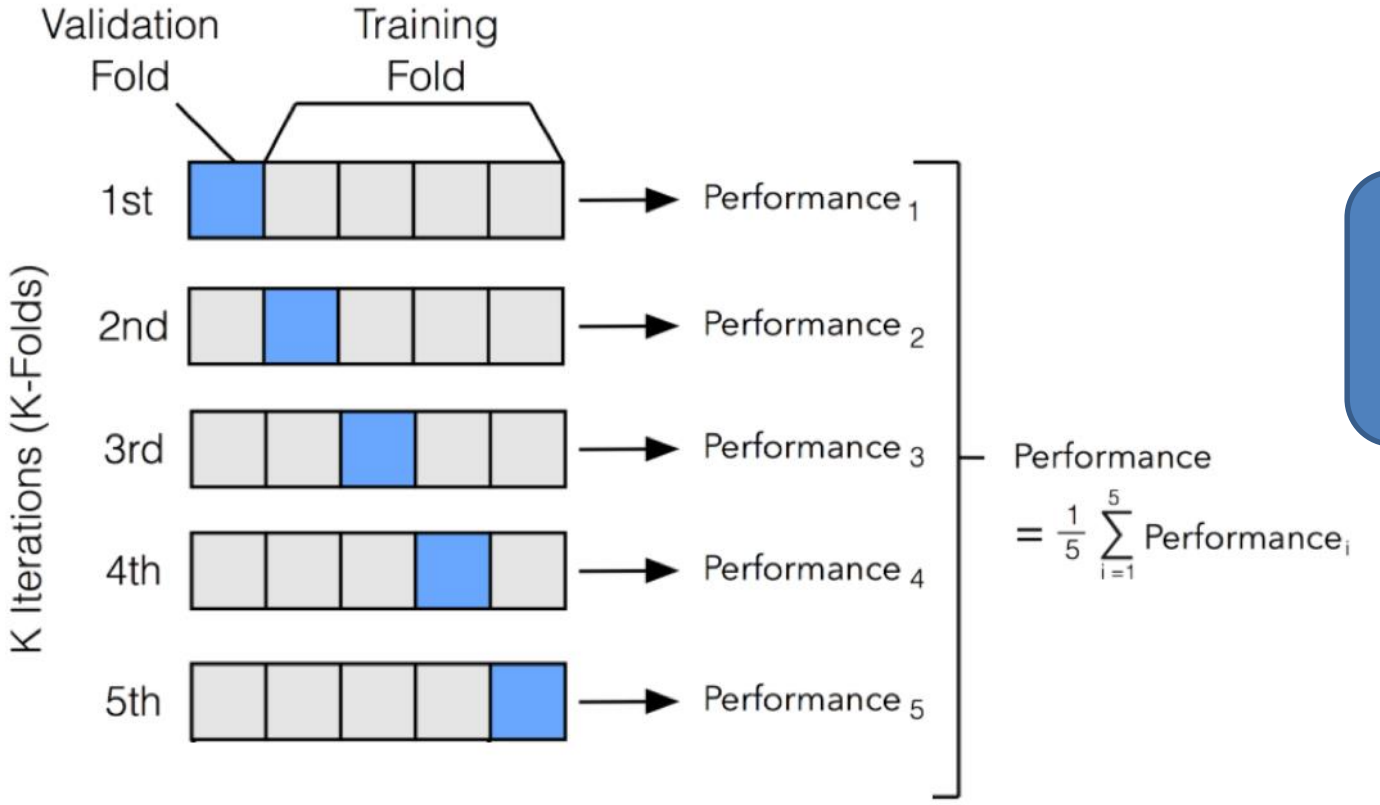
- 과적합이란?
  - 신경망 모델이 학습 데이터 세트에 대해서는 일정 수준 이상의 예측 정확도를 보이지만,

    새로운 데이터 세트에 적용하면 성능이 낮은 현상
  - 모델이 주어진 학습 세트에 대해서만 최적화된 것을 말함

- 과적합시 모델의 성능은?
  - K-fold cross validation

K-fold cross validation(k-겹 교차 검증)

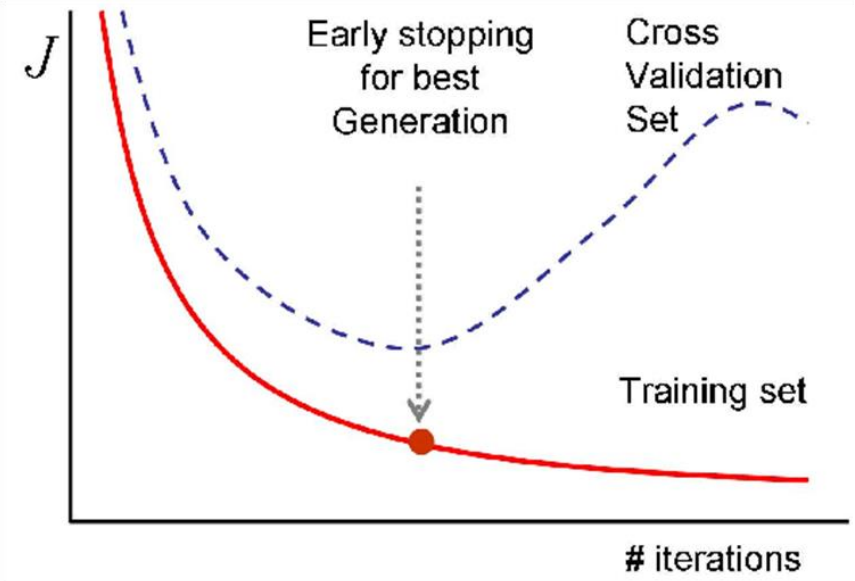
- 학습 데이터가 부족할 때, 과적합시 모델의 성능 예측에 사용
- 방법
  - 데이터 세트를 k-블록으로 나누어,
  - 한 블록씩 번갈아 가며 테스트 세트로 사용하고, 나머지 k-1 블록을 학습 세트로 사용하여 학습. 테스트 셋에 대한 평균 성능을 사용하는 방법
-



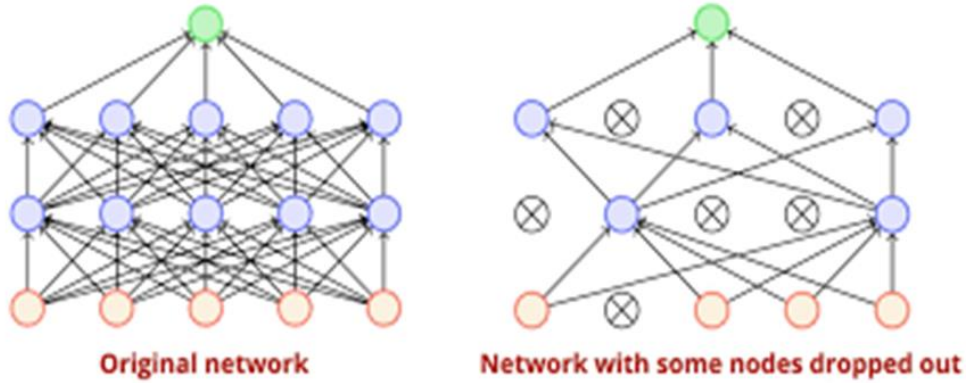
실제 상황에서의 성능과 비슷하게 나옴.

과적합(over fitting)

- 과적합이란?



- 과적합을 피하는 방법 EarlyStopping
  - ① 학습과 동시에 테스트를 병행하여 진행
  - ② 규제화(Regulization): Drop-out
    - ✓ 학습 중에 일부의 노드를 제거



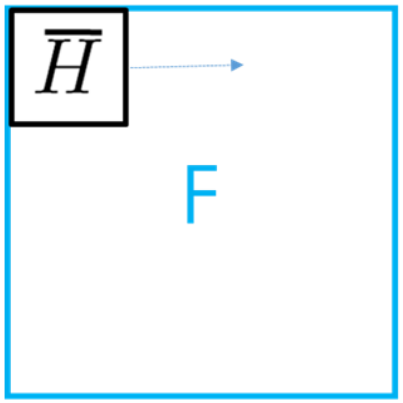
컨볼루션 신경망

- CNN 관련 주요 용어
  1. Convolution(합성곱)
  2. 커널(kernel)/필터(filter)
  3. 스트라이드(stride)
  4. 피처 맵(feature map)
  5. 패딩(padding)
  6. 채널(channel)
  7. 엑티베이션 맵(activation map)
  8. 풀링 레이어(pooling layer)
  9. 다층 퍼셉트론 (fully connected network)

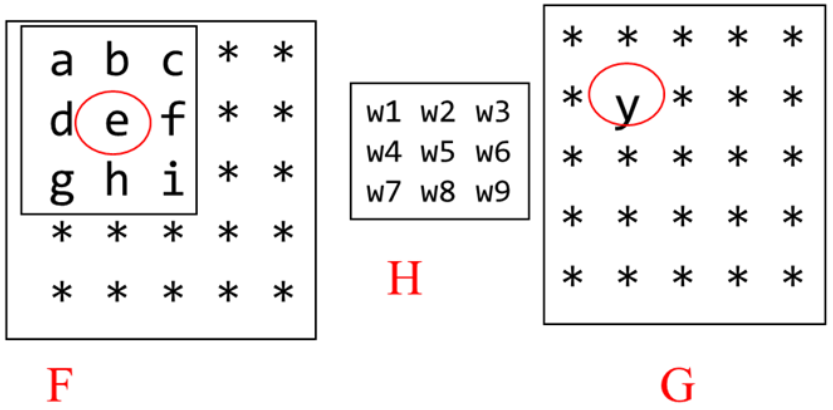
컨볼루션 신경망

1. 컨볼루션(Convolution) = 합성곱
   - 하나의 함수와 또 다른 함수가 겹치는 구간의 면적

- 영상 처리(image processing)에서의 컨볼루션



$$G = H * F$$



$$\begin{aligned} y \\ &= a \cdot w1 + b \cdot w2 + c \cdot w3 \\ &+ d \cdot w4 + e \cdot w5 + f \cdot w6 \\ &+ g \cdot w7 + h \cdot w8 + i \cdot w9 \end{aligned}$$

컨볼루션 신경망

2. 커널(Filter = Kernel)

- (이미지의) 특징을 찾아내기 위한 공용 파라메터
- CNN의 학습의 대상은 필터 파라메터

3. 스트라이드(stride)

- 필터가 이동하는 간격

4. 피처 맵(feature map)

- 원본 이미지 크기 $m \times m$
- 필터(filter, kernel) 크기 $n \times n$
- 필터가 이동하는 보폭 크기(stride) s
- ➔ 특성 지도(feature map)의 크기 $\frac{(m-n)}{s} + 1$

컨볼루션 신경망

5. 패딩(Padding)

- Feature map의 크기가 줄어들지 않도록 입력 데이터의 외곽에 지정된 픽셀만큼 특정값으로 채워 넣는 것
- 보통은 0으로 채움
- 다음의 크기만큼 상하좌우에 padding (K: 필터 크기)

$$Zero\ Padding = \frac{(K-1)}{2}$$

6. 채널(Channel)

- 1종류의 데이터
- 컬러 사진은 RGB 3개 성분의 3종의 이미지 ➔ 3 채널
  - ✓ 높이 39픽셀, 폭 31픽셀인 컬러 사진 ➔ (39, 31, 3)
- 흑백 사진 ➔ 1채널
  - ✓ 높이 39픽셀, 폭 31픽셀인 흑백 사진 ➔ (39, 31, 1)

컨볼루션 신경망

7. 활성화 및 활성화 맵(Activation map)

- 활성화 함수:
  - ✓ reLU(rectified Linear Unit) 함수
- 활성화 맵:
  - ✓ Feature map을 활성화한 결과인 2차원 맵

8. 풀링 (pooling)

- 합성곱(convolution)의 결과의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용
- 종류
  - ✓ 최대 풀링(Max pooling)
  - ✓ 평균 풀링(Average pooling)
  - ✓ 확률적 풀링(stochastic pooling)
- Pooling size는 정사각형
- 입력 데이터의 크기는 pooling size의 배수 관계

컨볼루션 신경망

9. Fully connected neural network

= Flatten layer + Softmax layer

- 마지막 풀링이 끝난 후 NxN 크기의 2D결과를 1차원 벡터로 표현(flatten layer) 후, 다층 신경망(softmax layer)을 구성
- 다층 퍼셉트론: 오류역전파 학습 (경사 하강법)

ILSVRC - classification

- https://www.youtube.com/watch?v=5-C_sTUW-Ts
- https://www.youtube.com/watch?v=xVhD2OBqoyg

전이 학습

- 전이 학습(transfer learning)
  - 전이 학습이란, 이미 학습된 모델을 재사용하여, 다른 영역의 데이터를 추가로 학습시키는 것
  - ImageNet 데이터를 통해 점, 선, 색, 모양 등의 이미지를 인식하는 기본적인 능력이 학습되어 있으므로, 여기에 이미지를 추가로 학습 시키는 방식

- 전이 학습이 적용되는 모델들
  - XCeption, VGG16, Inception V3
    - ✓ TensorFlow, Keras, Pytorch에서 인터페이스 제공
  - VGG16, VGG19, Inception V3, XCeption, ResNet50
    - ✓ GitHub 에 제공

전이 학습 적용 방법론

1. Feature extraction
   - 최종 레이어를 제외한 pre-trained model(VGG 또는 Inception V3 모델 등) 을 다른 task에서 feature를 추출하기 위해 사용
2. Fine-tuning
   - 합성곱 신경망이 앞쪽의 layer는 일반적인 feature를 추출하고, 뒤쪽 layer일수록 주어진 task에 특화된 feature를 추출하는 특징을 활용
   - 특정 layer는 weight를 고정시키고 나머지는 재학습되는 동안 fine tuning(미세 조정)되도록함