3일차 인공신경망 및 CNN 모델

강의자료: https://github.com/nahyungsun/tutorial

0. 인공 신경망

신경망

- 기계 학습 역사에서 가장 오래된 기계학습 모델, 현재 가장 다양한 형태를 가짐
- 1950년대 퍼셉트론 → 1980년대 다층 퍼셉트론
- 딥러닝의 기초가 됨

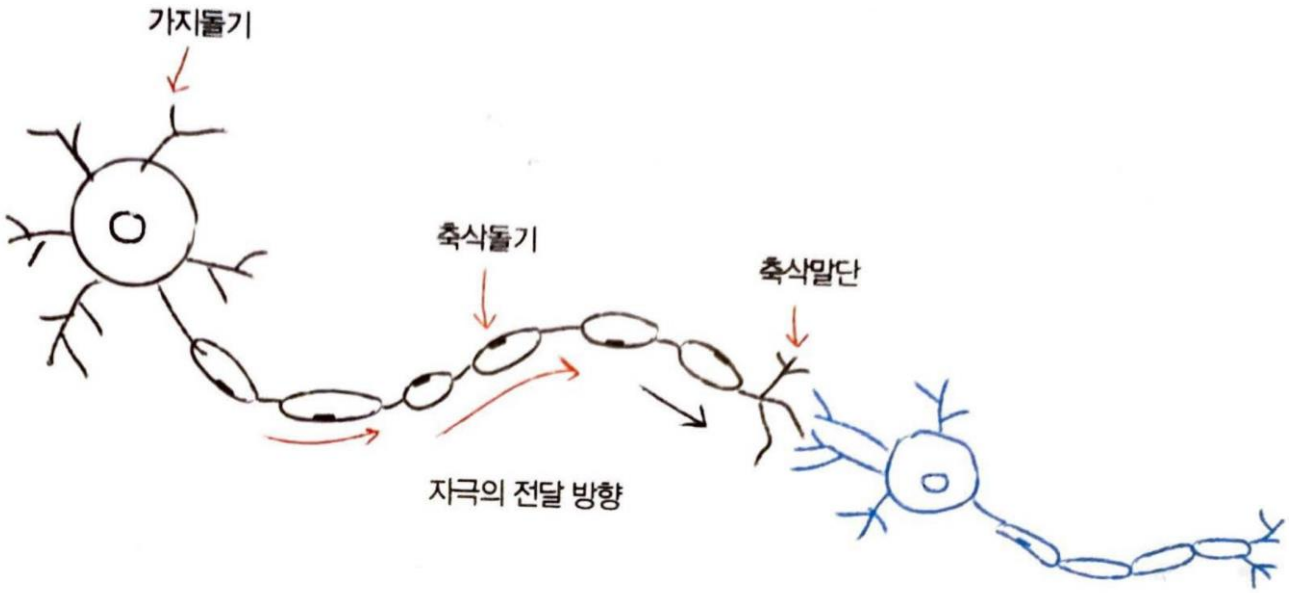
신경망의 발전

- 컴퓨터 과학과 의학의 시너지
- 컴퓨터 과학: 계산 능력 발전
- 의학: 두뇌의 정보 처리 방식 연구(뉴런의 동작 이해 등)

0. 인공 신경망

뉴런의 동작 원리

- 가지돌기에서 신호를 받아들임
- 신호가 축삭돌기를 지나 축삭말단으로 전달됨
- 축삭돌기를 지나는 동안 신호가 약해지거나, 강하게 전달되기도 함
- 축삭말단까지 전달된 신호는 다른 뉴런의 가지돌기로 전달



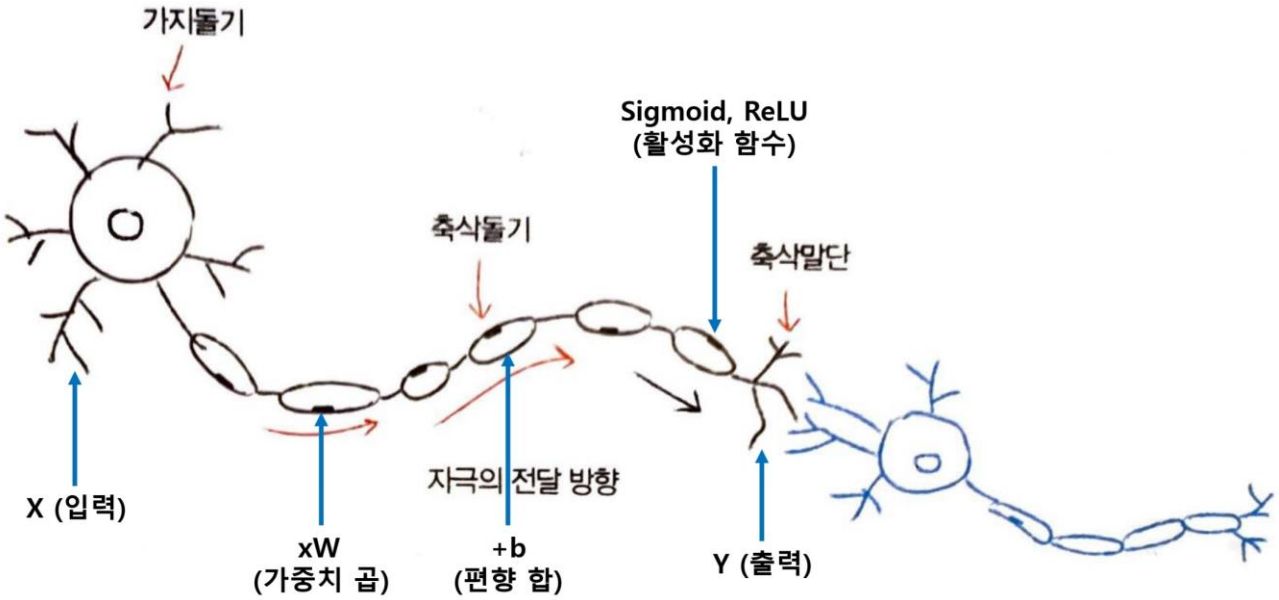
0. 인공 신경망

뉴런의 기본 동작

- 입력 신호, 즉 입력 값 X에 가중치(W)를 곱하고 편향(b)을 더함
- 그 후 활성화 함수를 거쳐 결과 값 Y를 만들어 냄

학습

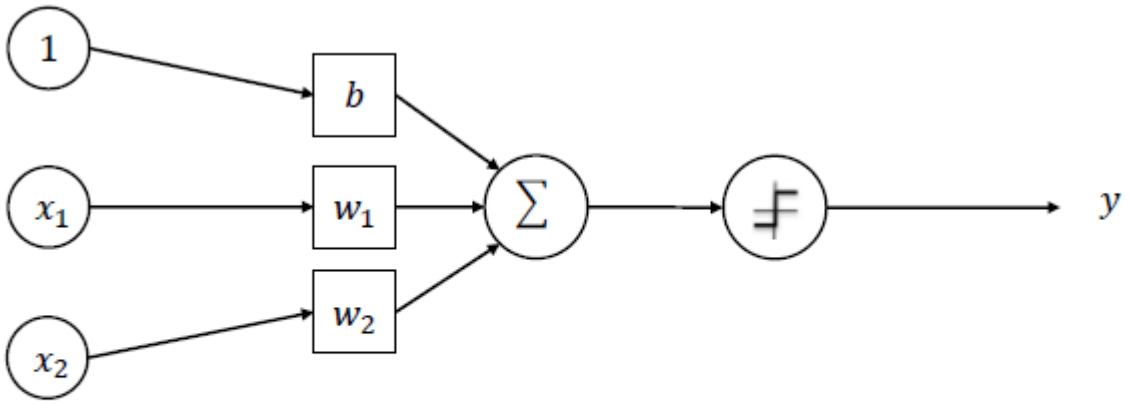
- 원하는 Y 값을 만들기 위해 가중치와 편향을 변경해가면서 적절한 값을 찾아내는 과정



0. 인공 신경망

인공신경망 구조

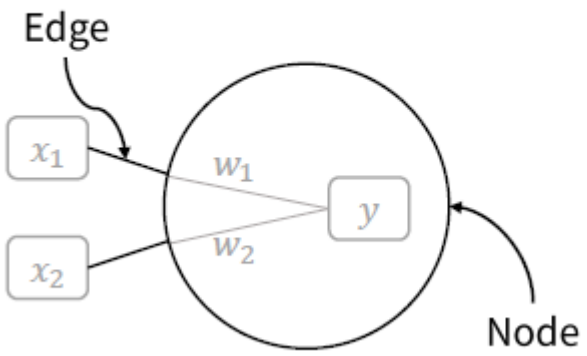
– 인공 뉴런인 퍼셉트론을 기본 단위로 하며, 이를 조합해 복잡한 구조를 이룸



0. 인공 신경망

인공신경망 표현

- 노드(Node)와 간선(Edge)을 이용해 표현
- 노드: 단일 뉴런 연산
- 간선: 뉴런의 연결성



편향을 잊지 말자

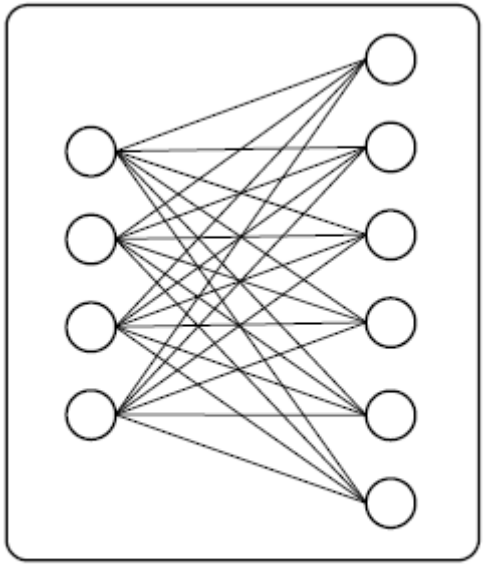
$$y = a(w_1x_1 + w_2x_2 + b)$$

$$y = a(\boldsymbol{w}^T\boldsymbol{x} + b)$$

0. 인공 신경망

전결합 계층

- Fully-Connected Layer
- 두 계층 간의 연결이 모두 되어 있는 계층으로 고전적인 신경 네트워크 아키텍처

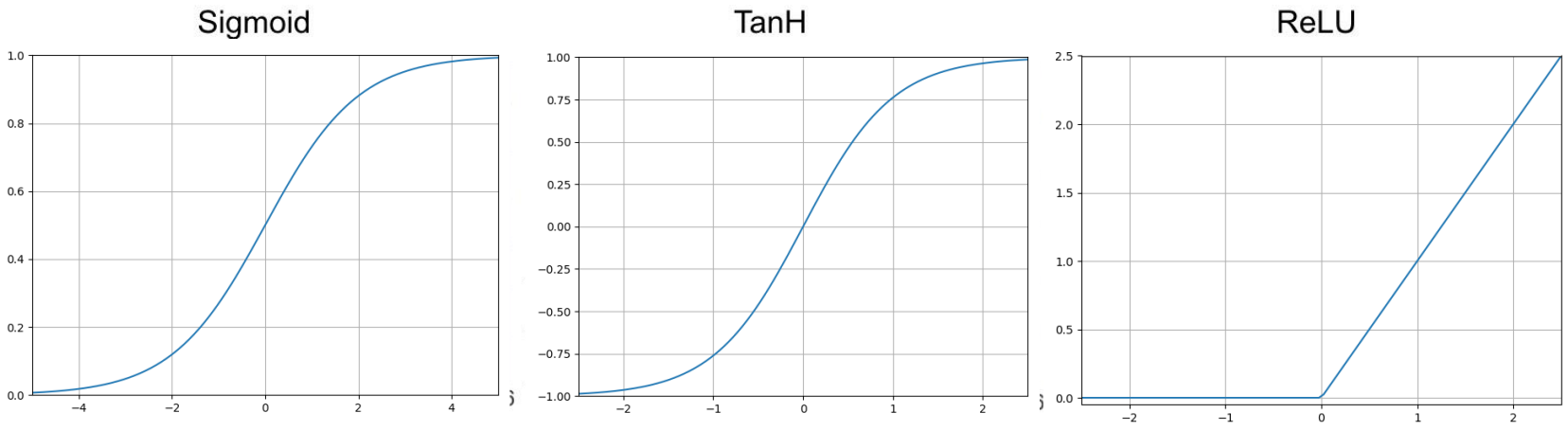


FC Layer의 그래프 표현

0. 인공 신경망

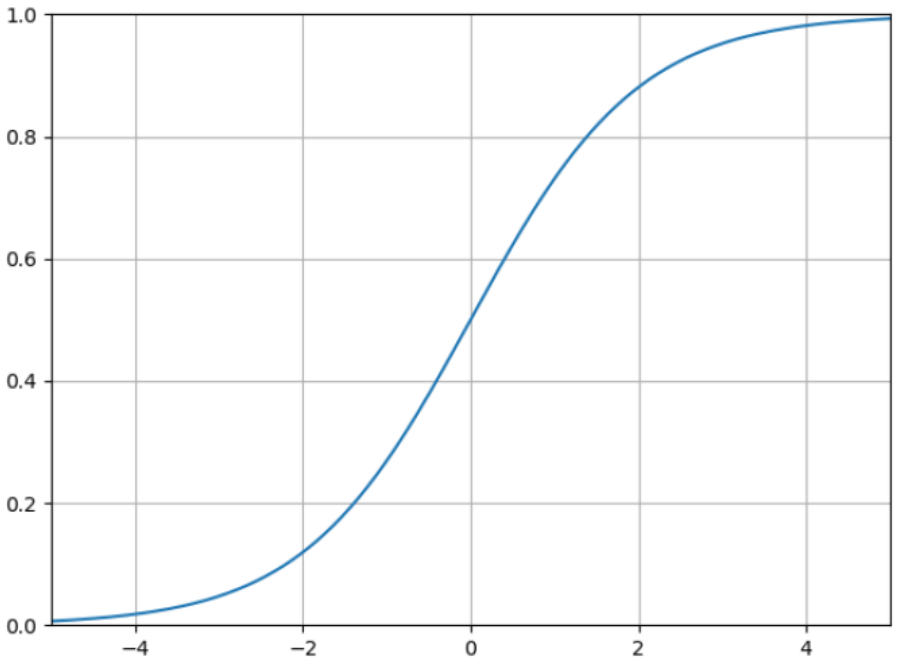
활성화 함수

- 인공신경망을 통과해온 값을 최종적으로 어떤 값으로 만들지 결정
- 일반적으로 비선형 함수



0. 인공 신경망

Sigmoid

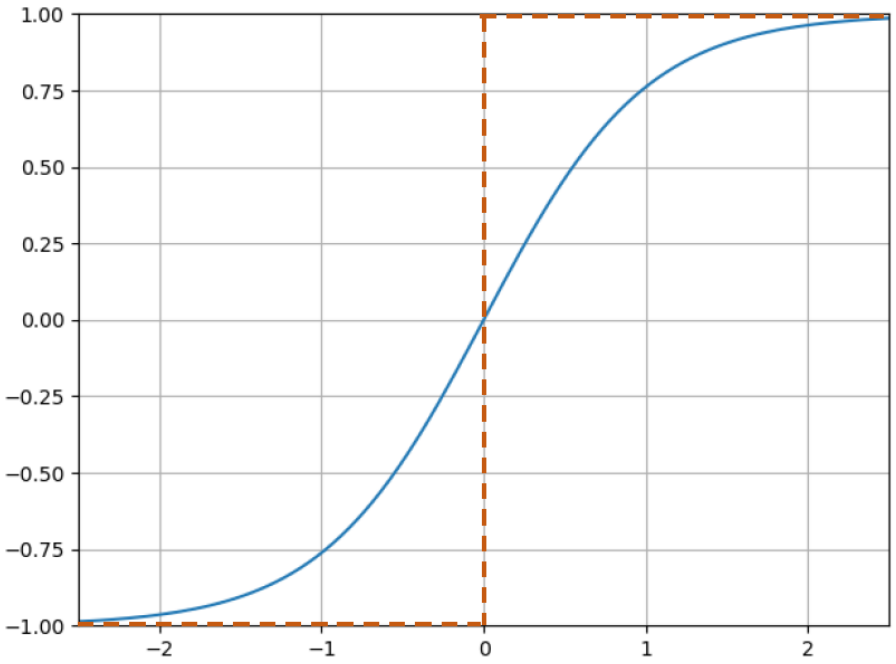


Sigmoid 함수

- 값이 작아질 수록 0, 커질 수록 1에 수렴
- 0~1사이의 실수 값으로 출력이 정의됨
- 확률을 표현할 수 있음
- 입력값이 0에 가까울 수록 출력이 크게 변함

0. 인공 신경망

Tanh

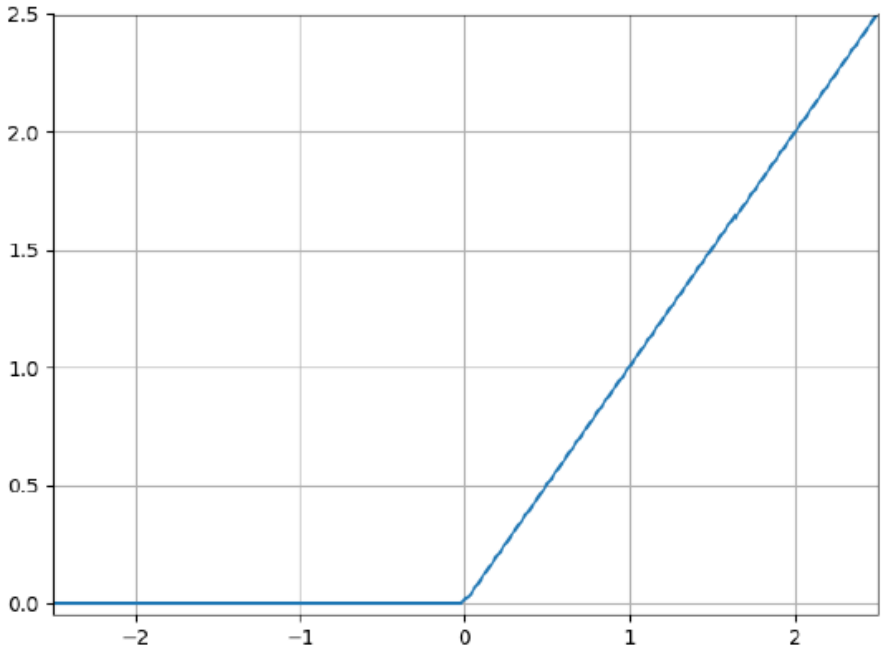


Tanh (Hyperbolic tangent) 함수

- 값이 작아질 수록 -1, 커질 수록 1에 수렴
- -1~1사이의 실수 값으로 출력이 정의됨
- 입력값이 0에 가까울 수록 출력이 크게 변함

0. 인공 신경망

ReLU

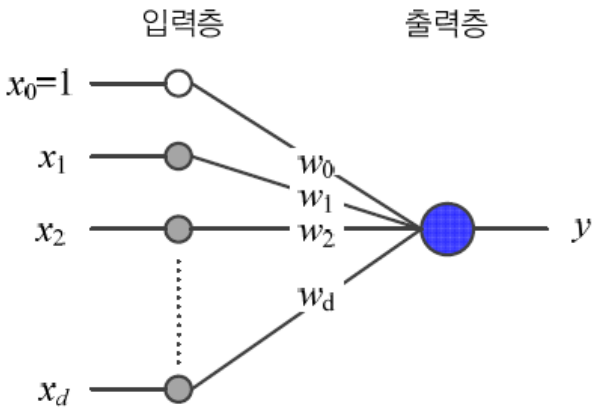


ReLU (Rectified linear unit) 함수

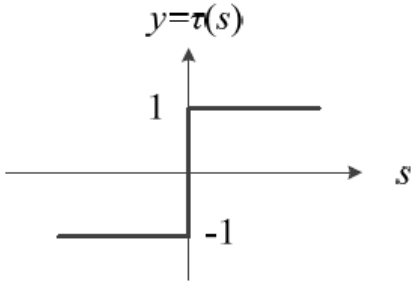
- 0보다 작은 값을 0으로 강제하는 함수
- 딥러닝에서 가장 많이 사용됨
- 미분 값이 일정해 학습에 용이함
- 구현이 단순해 빠른 연산이 가능

0. 인공 신경망

- 단층 퍼셉트론의 구조
  - 입력층과 출력층을 가짐
  - 입력층의 i번째 노드는 입력 벡터 x= $(x_1, x_2, \ldots, x_d)$의 요소 x_i를 담당
  - i번째 입력층 노드와 출력층을 연결하는 간선은 가중치 w_i를 가짐
  - 출력층은 하나의 노드



(a) 퍼셉트론의 구조



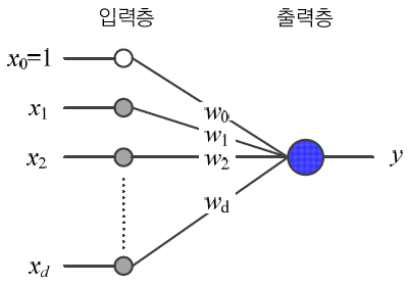
(b) 계단함수를 활성함수 $\tau(s)$로 이용함

0. 인공 신경망

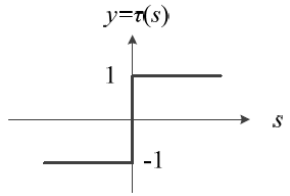
- 퍼셉트론의 동작
  - 해당하는 입력값과 가중치를 곱한 결과를 모두 더해 s를 구하고, 활성함수를 적용
  - 활성함수로 계단함수를 사용하므로 최종 출력 y는 +1 또는 -1 즉, 선형 분류기

$$y = \tau(s)$$

$$\text{이때 } s = w_0 + \sum_{i=1}^{d} w_i x_i, \qquad \tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$$



(a) 퍼셉트론의 구조



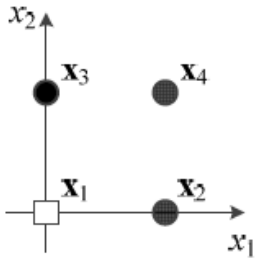
(b) 계단함수를 활성함수 $\tau(s)$로 이용함

S=X0*W0+ X1*W1+ X2*W2+… +Xd*Wd
Y=활성화함수(S)

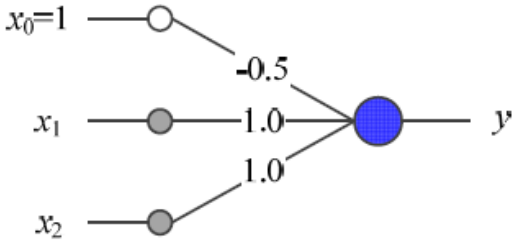
0. 인공 신경망

- 예제
  - OR 분류

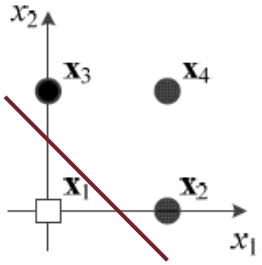
$$\mathbf{x}_1 = \begin{pmatrix}0\\0\end{pmatrix},\ y_1 = -1,\ \ \mathbf{x}_2 = \begin{pmatrix}1\\0\end{pmatrix},\ y_2 = 1,\ \ \mathbf{x}_3 = \begin{pmatrix}0\\1\end{pmatrix},\ y_3 = 1,\ \ \mathbf{x}_4 = \begin{pmatrix}1\\1\end{pmatrix},\ y_4 = 1$$



(a) 훈련집합



(b) 퍼셉트론

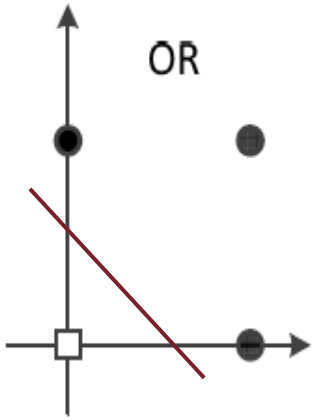


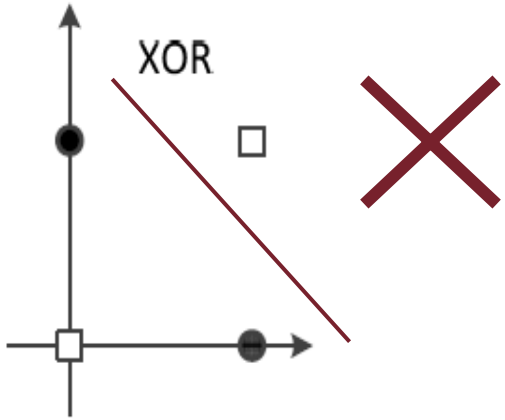
(a) 훈련집합

$$\mathbf{x}_1\text{: } s = -0.5 + 0 * 1.0 + 0 * 1.0 = -0.5, \quad \tau(-0.5) = -1$$
$$\mathbf{x}_2\text{: } s = -0.5 + 1 * 1.0 + 0 * 1.0 = 0.5, \quad \tau(0.5) = 1$$
$$\mathbf{x}_3\text{: } s = -0.5 + 0 * 1.0 + 1 * 1.0 = 0.5, \quad \tau(0.5) = 1$$
$$\mathbf{x}_4\text{: } s = -0.5 + 1 * 1.0 + 1 * 1.0 = 1.5, \quad \tau(1.5) = 1$$

0. 인공 신경망

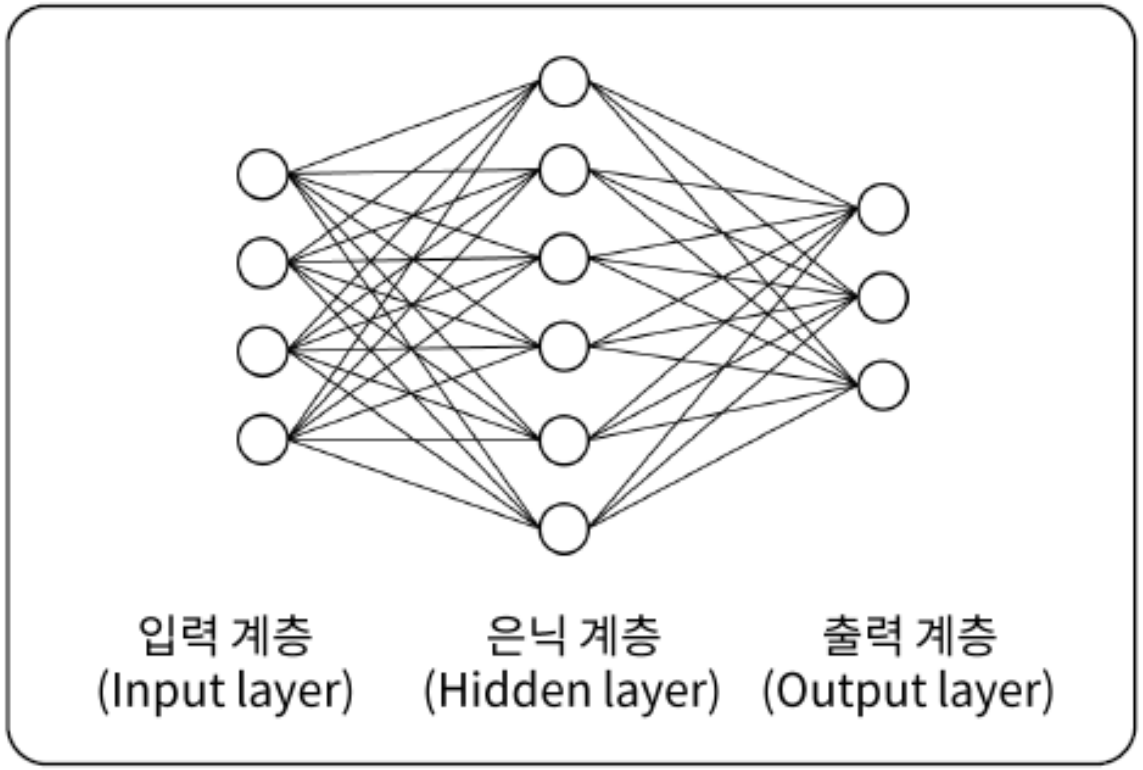
- 퍼셉트론의 한계
  - 선형 분류가 불가능한 상황에서는 해결 할 수 없음





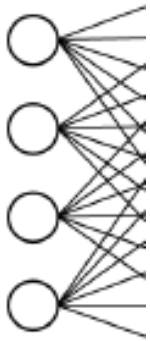
0. 인공 신경망

- 다층 퍼셉트론
  - 입력 층(Input Layer), 은닉 층(Hidden Layer), 출력 층(Output layer)로 이루어진 신경망
  - 각 계층은 전결합층으로 이루어져 있음



0. 인공 신경망

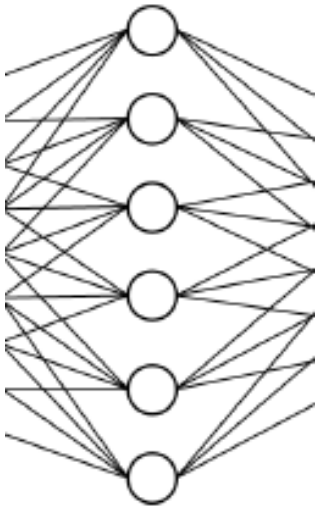
- 입력 층(Input Layer)
  - 아무런 연산이 일어나지 않고 신경망의 입력 값을 받아 다음 계층으로 전달하는 역할
  - 입력 층의 노드 수는 입력 벡터의 길이와 동일



입력 계층
(Input layer)

0. 인공 신경망

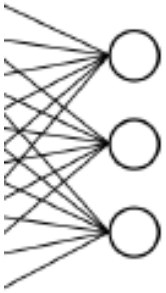
- 은닉 층(Hidden Layer)
  - 입력 층과 연결된 전결합 계층
  - 복잡한 문제를 해결할 수 있게 하는 핵심 계층



은닉 계층
(Hidden layer)

0. 인공 신경망

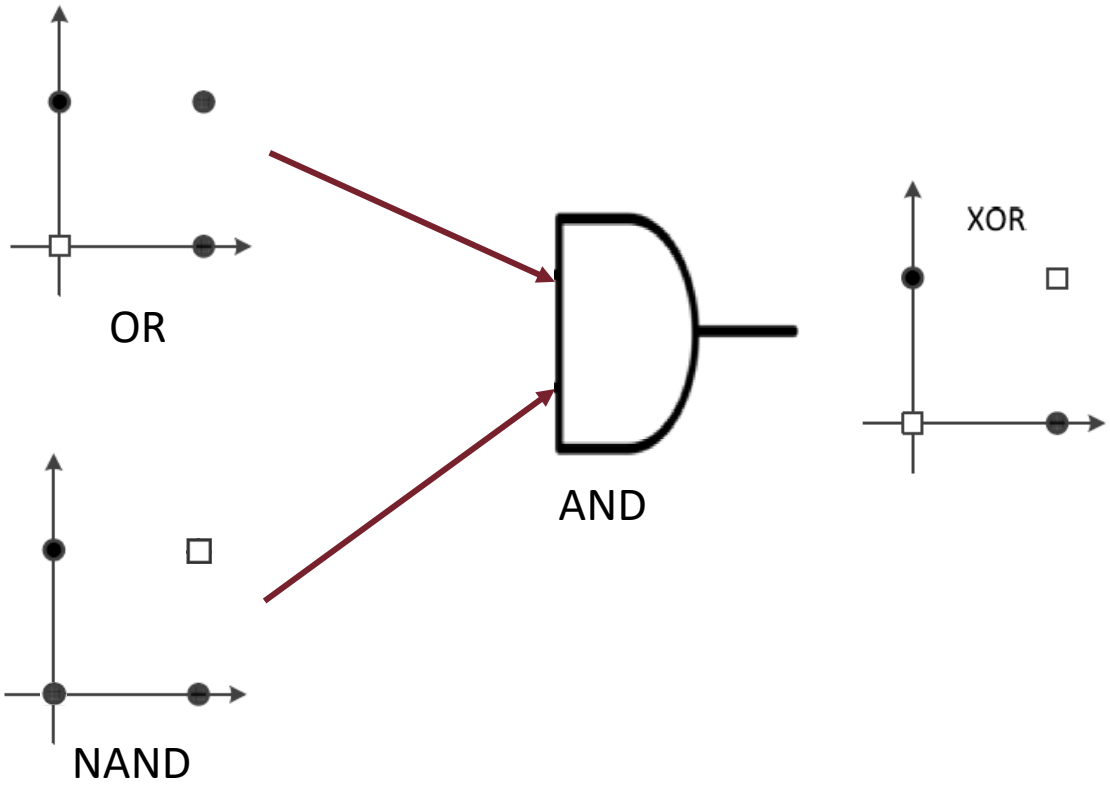
- 출력 층(Output Layer)
  - 은닉 층 다음에 오는 전결합 계층
  - 신경망 외부로 출력 값을 전달 함
  - 출력 층의 노드 수는 출력 벡터의 길이와 동일



출력 계층
(Output layer)

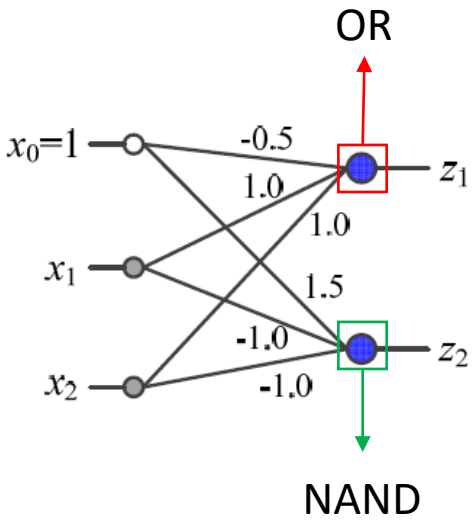
0. 인공 신경망

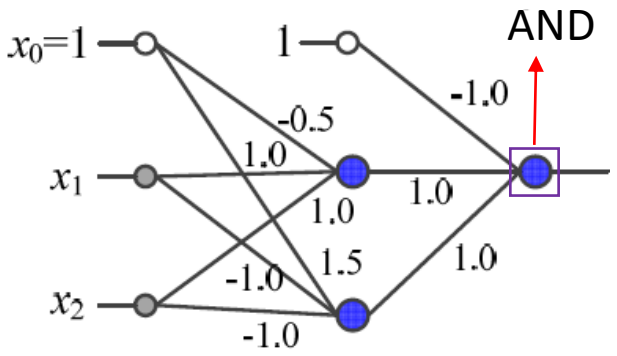
- 다층 퍼셉트론
  - XOR 분류



0. 인공 신경망

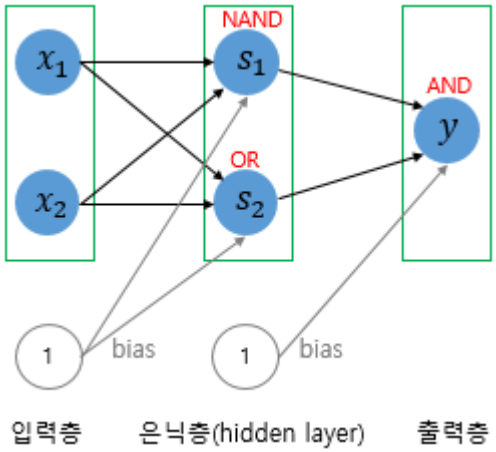
- 다층 퍼셉트론
  - XOR 분류
    - OR퍼셉트론과 NAND퍼셉트론을 병렬로 연결
    - 결과를 AND 퍼셉트론에 순차 연결

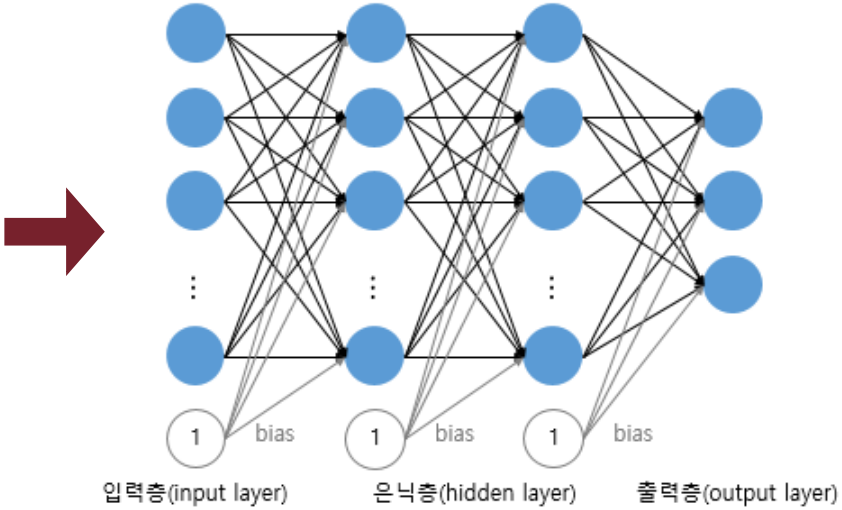




0. 인공 신경망

- 다층 퍼셉트론
  - 은닉층을 더 쌓아 심층 신경망(Deep Neural Network)을 만들어 더 복잡한 문제도 해결 가능

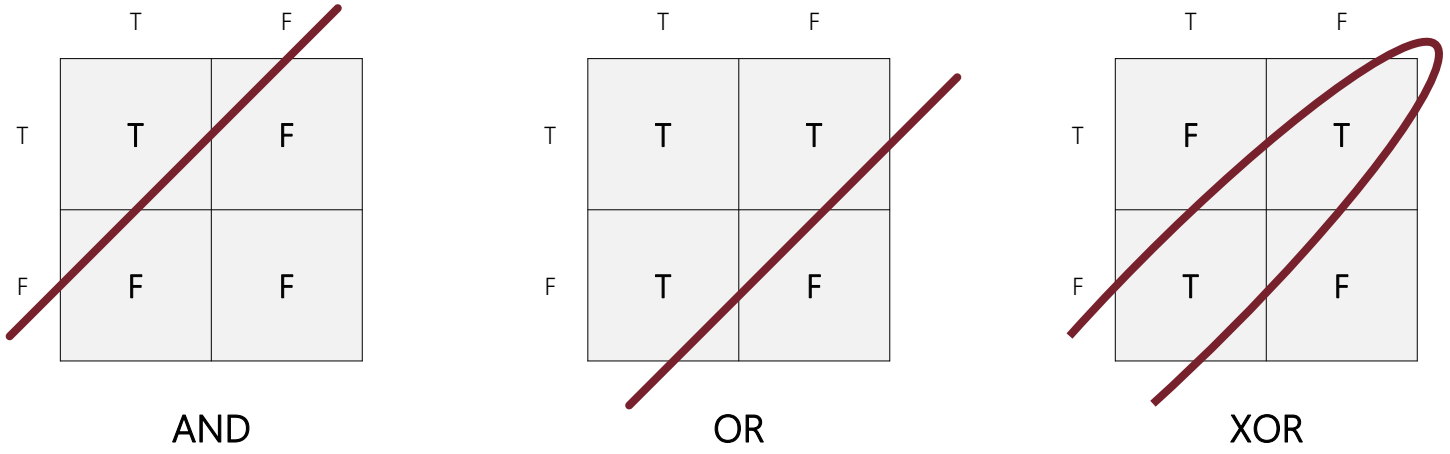




0. 인공 신경망

딥러닝

- 머신러닝의 선형적(Linear)인 성질 때문에 비선형적 문제를 해결할 수 없음
- 인간의 뇌 신경과 유사한 구조인 인공신경망을 이용하여 학습함
- 인공신경망이 다층으로 쌓인 것이 딥러닝(Deep-Learning)



0. 인공 신경망

XOR 실습

- 필요한 라이브러리 임포트

```
import torch
device = 'cuda' if torch.cuda.is_available() else 'cpu'
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

- XOR 문제에 해당되는 입력과 출력 정의

```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
```

- 1개의 뉴런을 가지는 단층 퍼셉트론 구현

```
linear = nn.Linear(2, 1, bias=True)
sigmoid = nn.Sigmoid()
model = nn.Sequential(linear, sigmoid).to(device)
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)
```

0. 인공 신경망

XOR 실습

- 200번 부터 epoch 비용이 감소되지 않는 것을 확인할 수 있다.
- 단층 퍼셉트론은 XOR 문제를 풀 수 없기 때문!

```
#10,001번의 에포크 수행. 0번 에포크부터 2,000번 에포크까지.
for step in range(2000):
    optimizer.zero_grad()
    hypothesis = model(X)

    # 비용 함수
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 100 == 0: # 100번째 에포크마다 비용 출력
        print(step, cost.item())
```

```
0 0.7666423320770264
100 0.6931473612785339
200 0.6931471824645996
300 0.6931471824645996
400 0.6931471824645996
500 0.6931471824645996
600 0.6931471824645996
700 0.6931471824645996
800 0.6931471824645996
900 0.6931471824645996
1000 0.6931471824645996
1100 0.6931471824645996
1200 0.6931471824645996
1300 0.6931471824645996
1400 0.6931471824645996
1500 0.6931471824645996
1600 0.6931471824645996
1700 0.6931471824645996
1800 0.6931471824645996
1900 0.6931471824645996
```

0. 인공 신경망

XOR 실습

- 결과를 확인해 봐도 XOR 문제를 풀 수 없는 것을 확인할 수 있다.

```
with torch.no_grad():
    hypothesis = model(X)
    predicted = (hypothesis > 0.5).float()
    accuracy = (predicted == Y).float().mean()
    print('모델의 출력값(Hypothesis): ', hypothesis.detach().cpu().numpy())
    print('모델의 예측값(Predicted): ', predicted.detach().cpu().numpy())
    print('실제값(Y): ', Y.cpu().numpy())
    print('정확도(Accuracy): ', accuracy.item())
```

```
모델의 출력값(Hypothesis):  [[0.5]
 [0.5]
 [0.5]
 [0.5]]
모델의 예측값(Predicted):  [[0.]
 [0.]
 [0.]
 [0.]]
실제값(Y):  [[0.]
 [1.]
 [1.]
 [0.]]
정확도(Accuracy):  0.5
```

0. 인공 신경망

XOR 실습

- XOR 문제를 해결하기 위해 다층 퍼셉트론을 시도
- 필요한 라이브러리 임포트
- XOR 문제에 해당되는 입력과 출력 정의

```
import torch
import torch.nn as nn
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

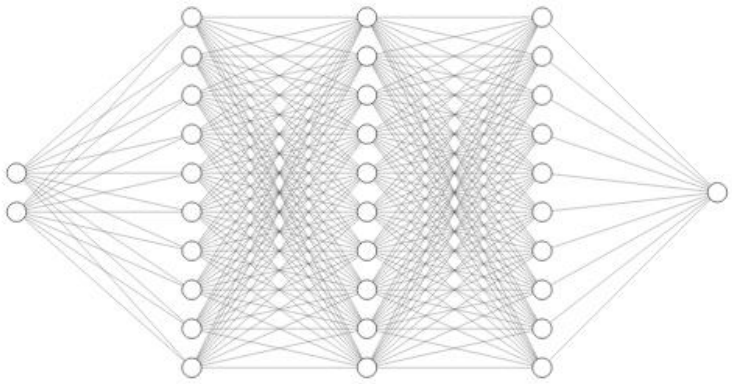
```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
```

0. 인공 신경망

XOR 실습

- 다층 퍼셉트론 설계
- 입력층, 은닉층1, 은닉층2, 은닉층3, 출력층을 가지는 은닉층이 3개인 인공 신경망
- 밑의 그림은 설계한 인공 신경망을 시각화한 것

```
model = nn.Sequential(
          nn.Linear(2, 10, bias=True), # input_layer = 2, hidden_layer1 = 10
          nn.Sigmoid(),
          nn.Linear(10, 10, bias=True), # hidden_layer1 = 10, hidden_layer2 = 10
          nn.Sigmoid(),
          nn.Linear(10, 10, bias=True), # hidden_layer2 = 10, hidden_layer3 = 10
          nn.Sigmoid(),
          nn.Linear(10, 1, bias=True), # hidden_layer3 = 10, output_layer = 1
          nn.Sigmoid()
          ).to(device)
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)  # modified learning rate from 0.1 to 1
```



0. 인공 신경망

XOR 실습

- 이전과 다르게 epoch 가 진행될 때마다 비용이 줄어드는 것을 확인할 수 있다.

```
for epoch in range(10001):
    optimizer.zero_grad()
    # forward 연산
    hypothesis = model(X)

    # 비용 함수
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    # 100의 배수에 해당되는 에포크마다 비용을 출력
    if epoch % 1000 == 0:
        print(epoch, cost.item())
```

```
0 0.00011297257151454687
1000 9.459738794248551e-05
2000 8.12893922557123e-05
3000 7.120047666830942e-05
4000 6.32575829513371e-05
5000 5.6864570069592446e-05
6000 5.166375922271982e-05
7000 4.731238732347265e-05
8000 4.358691876404919e-05
9000 4.041282954858616e-05
10000 3.7670895835611756e-05
```

0. 인공 신경망

XOR 실습

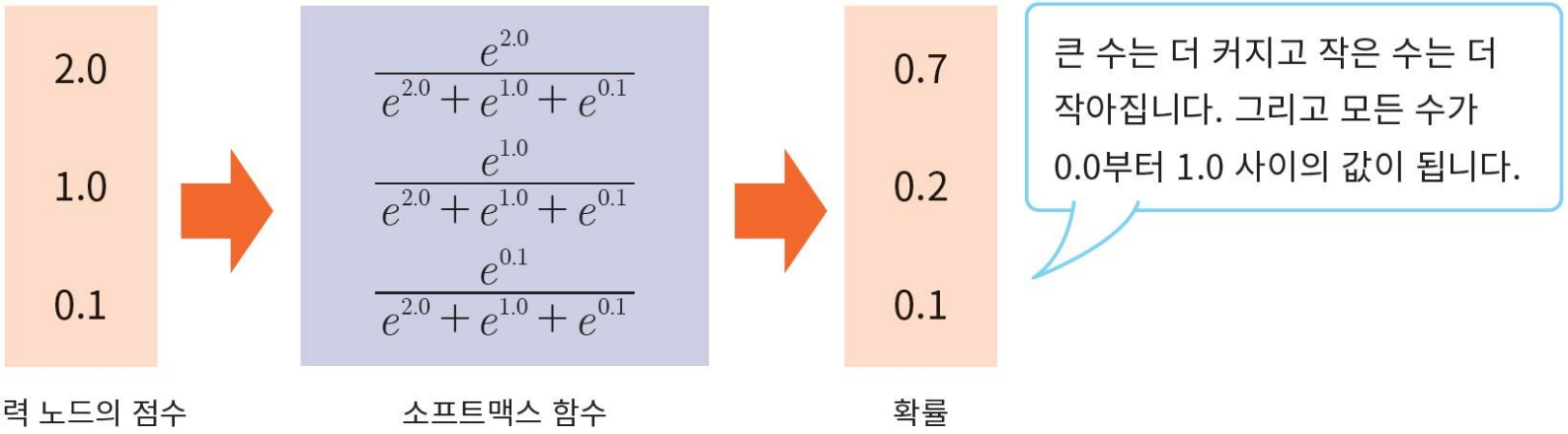
- 결과 역시 XOR 문제를 잘 해결한 것을 확인할 수 있다.

```
with torch.no_grad():
    hypothesis = model(X)
    predicted = (hypothesis > 0.5).float()
    accuracy = (predicted == Y).float().mean()
    print('모델의 출력값(Hypothesis): ', hypothesis.detach().cpu().numpy())
    print('모델의 예측값(Predicted): ', predicted.detach().cpu().numpy())
    print('실제값(Y): ', Y.cpu().numpy())
    print('정확도(Accuracy): ', accuracy.item())
```

```
모델의 출력값(Hypothesis):  [[2.3310218e-05]
 [9.9996090e-01]
 [9.9996424e-01]
 [5.2368901e-05]]
모델의 예측값(Predicted):  [[0.]
 [1.]
 [1.]
 [0.]]
실제값(Y):  [[0.]
 [1.]
 [1.]
 [0.]]
정확도(Accuracy):  1.0
```

0. 새로운 손실 함수

- 출력층의 노드에 소프트맥스 함수를 사용하고 손실 함수로 교차 엔트로피를 많이 사용한다
- 소프트 맥스 활성화 함수



- 교차 엔트로피는 2개의 확률분포 간의 거리를 측정한 것이다.
- 교차 엔트로피는 2개의 확률 분포 p, q에 대해서 다음과 같이 정의된다.

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

- 교차 엔트로피가 크면, 2개의 확률 분포가 많이 다른 것이다. 교차 엔트로피가 작으면 2개의 확률 분포가 거의 일치한다고 볼 수 있다.

0. 교차 엔트로피 손실 함수

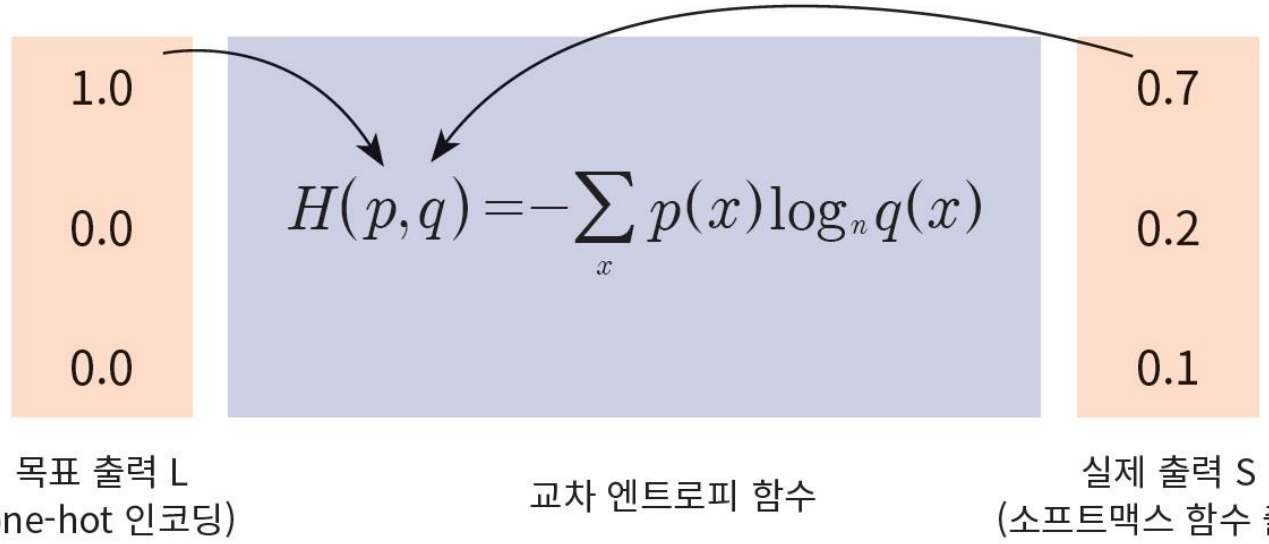


그림 14-9 교차 엔트로피 함수

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

0. 교차 엔트로피 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) = 0.51이 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.

0. 인공 신경망

MNIST 성능 향상

- 다층 퍼셉트론을 사용해 MNIST 모델의 성능을 향상시킬 수 있다.

```
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import matplotlib.pyplot as plt
import random

USE_CUDA = torch.cuda.is_available() # GPU를 사용가능하면 True, 아니라면 False를 리턴
device = torch.device("cuda" if USE_CUDA else "cpu") # GPU 사용 가능하면 사용하고 아니면 CPU 사용
print("다음 기기로 학습합니다:", device)

# for reproducibility
random.seed(777)
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)

# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)


```

0. 인공 신경망

MNIST 성능 향상

- 다층 퍼셉트론을 사용해 MNIST 모델의 성능을 향상시킬 수 있다.

```
# MNIST data image of shape 28 * 28 = 784
linear = nn.Sequential()
linear.add_module('fc1', nn.Linear(28*28*1, 100))
linear.add_module('relu1', nn.ReLU())
linear.add_module('fc2', nn.Linear(100, 100))
linear.add_module('relu2', nn.ReLU())
linear.add_module('fc3', nn.Linear(100, 10))
# MNIST data image of shape 28 * 28 = 784

# hyperparameters
training_epochs = 15
batch_size = 100

# 비용 함수와 옵티마이저 정의
criterion = nn.CrossEntropyLoss().to(device) # 내부적으로 소프트맥스 함수를 포함하고 있음.
optimizer = torch.optim.SGD(linear.parameters(), lr=0.1)

# dataset loader
data_loader = DataLoader(dataset=mnist_train,
                                          batch_size=batch_size, # 배치 크기는 100
                                          shuffle=True,
                                          drop_last=True)
```

0. 인공 신경망

MNIST 성능 향상

- 단층 퍼셉트론보다 비용이 작아진 것을 확인할 수 있다.

```
for epoch in range(training_epochs): # 앞서 training_epochs의 값은 15로 지정함.
    avg_cost = 0
    total_batch = len(data_loader)

    for X, Y in data_loader:
        # 배치 크기가 100이므로 아래의 연산에서 X는 (100, 784)의 텐서가 된다.
        X = X.view(-1, 28 * 28).to(device)
        # 레이블은 원-핫 인코딩이 된 상태가 아니라 0 ~ 9의 정수.
        Y = Y.to(device)

        optimizer.zero_grad()
        hypothesis = linear(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning finished')
```

```
Epoch: 0001 cost = 0.628847122
Epoch: 0002 cost = 0.239227369
Epoch: 0003 cost = 0.175555333
Epoch: 0004 cost = 0.138709322
Epoch: 0005 cost = 0.115421154
Epoch: 0006 cost = 0.096607938
Epoch: 0007 cost = 0.083727077
Epoch: 0008 cost = 0.072155662
Epoch: 0009 cost = 0.063731015
Epoch: 0010 cost = 0.056700680
Epoch: 0011 cost = 0.050654452
Epoch: 0012 cost = 0.044952631
Epoch: 0013 cost = 0.040506925
Epoch: 0014 cost = 0.036337849
Epoch: 0015 cost = 0.032280162
Learning finished
```

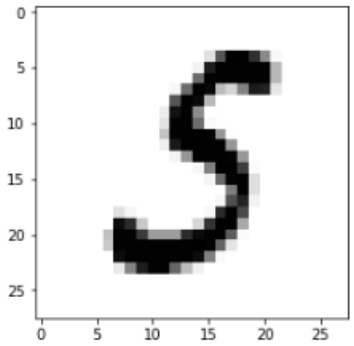
0. 인공 신경망

MNIST 성능 향상

- 정확도 역시 89% -> 97% 로 증가하였다.

```
# 테스트 데이터를 사용하여 모델을 테스트한다.
with torch.no_grad(): # torch.no_grad()를 하면 gradient 계산을 수행하지 않는다.
    X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = linear(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())

    # MNIST 테스트 데이터에서 무작위로 하나를 뽑아서 예측을 해본다
    r = random.randint(0, len(mnist_test) - 1)
    X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float().to(device)
    Y_single_data = mnist_test.test_labels[r:r + 1].to(device)

    print('Label: ', Y_single_data.item())
    single_prediction = linear(X_single_data)
    print('Prediction: ', torch.argmax(single_prediction, 1).item())

    plt.imshow(mnist_test.test_data[r:r + 1].view(28, 28), cmap='Greys', interpolation='nearest')
    plt.show()
```

```
Accuracy: 0.9725000262260437
Label:  5
Prediction:  5
```



1. CNN 적용 사례

- 정치인 재연

출처 :https://brunch.co.kr/@itschloe1/23

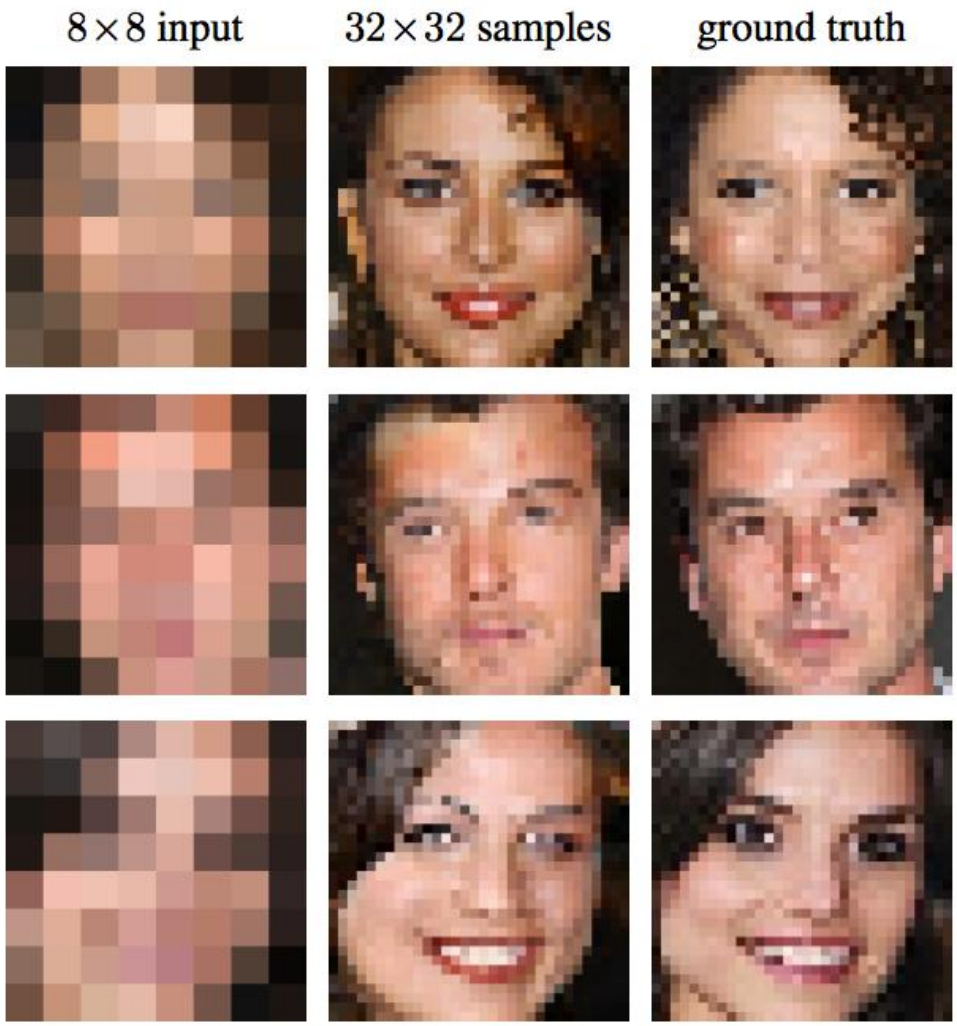
1. CNN 적용 사례

- 흑백 사진과 영상에 색 복원

출처 :https://brunch.co.kr/@itschloe1/23

1. CNN 적용 사례

- 픽셀 복원



출처 :https://brunch.co.kr/@itschloe1/23

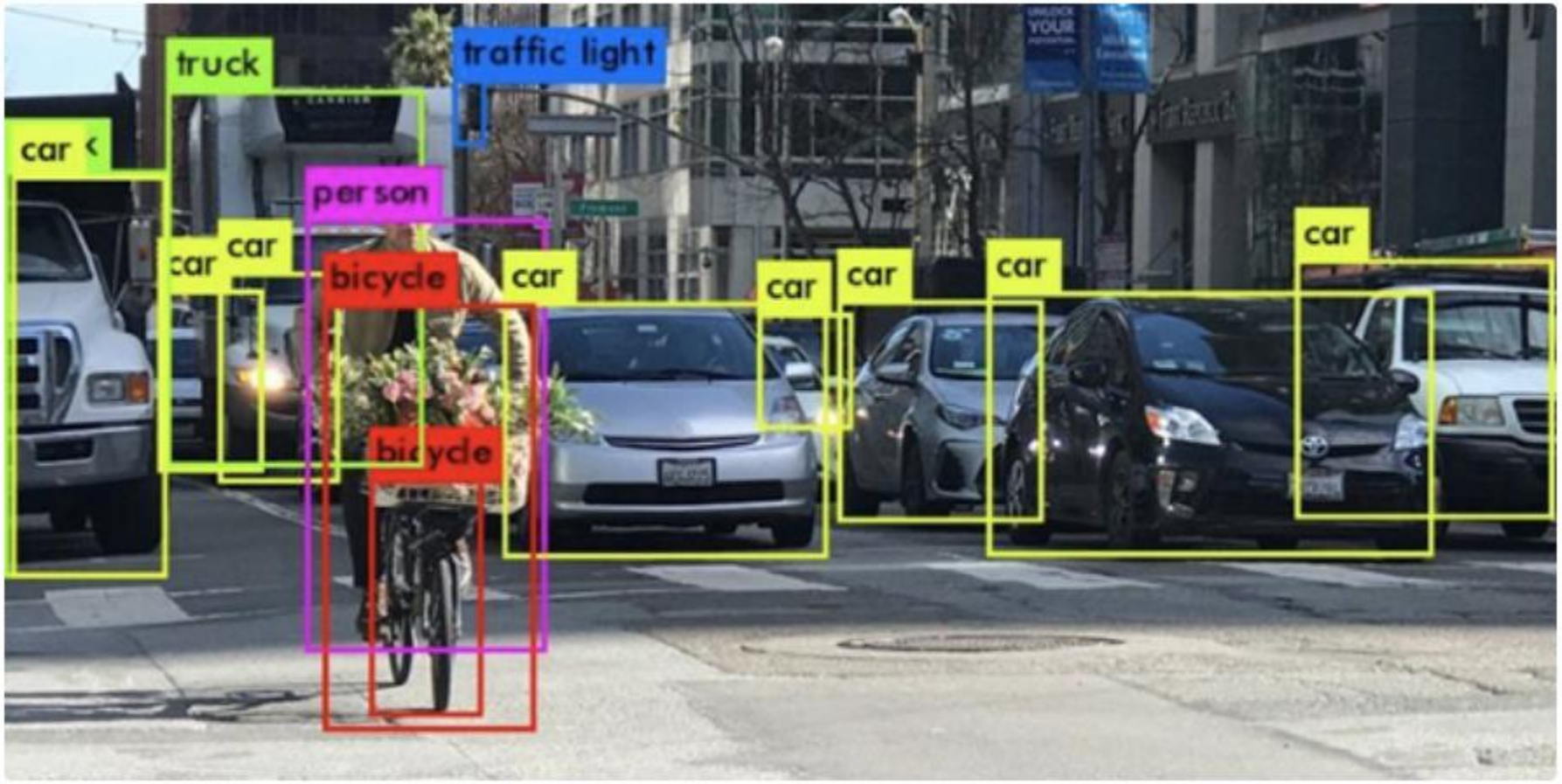
1. CNN 적용 사례

- 실시간으로 여러 사람의 움직임을 추정

출처 :https://brunch.co.kr/@itschloe1/23

1. CNN 적용 사례

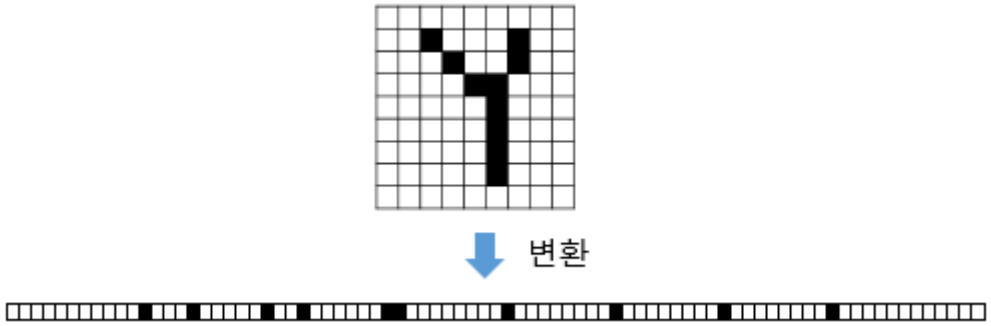
- 이미지 객체 인식



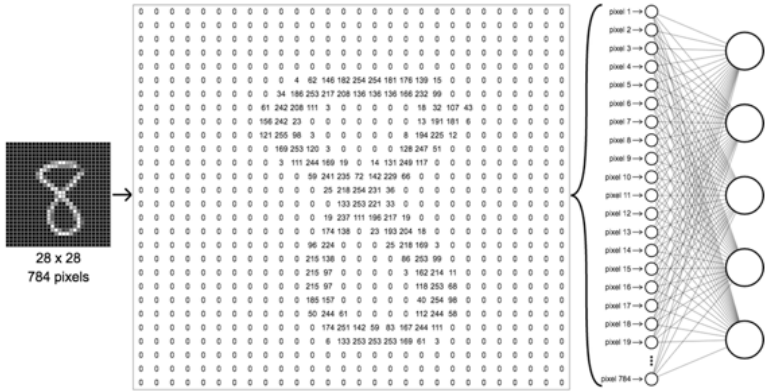
출처 :https://www.datamaker.io/posts/17/

2. CNN 이란?

- 기존 신경망 이미지 처리의 한계
  - 공간적 / 지역적 정보의 유실



  - 이미지의 크기가 커질수록 증가하는 가중치 학습 크기

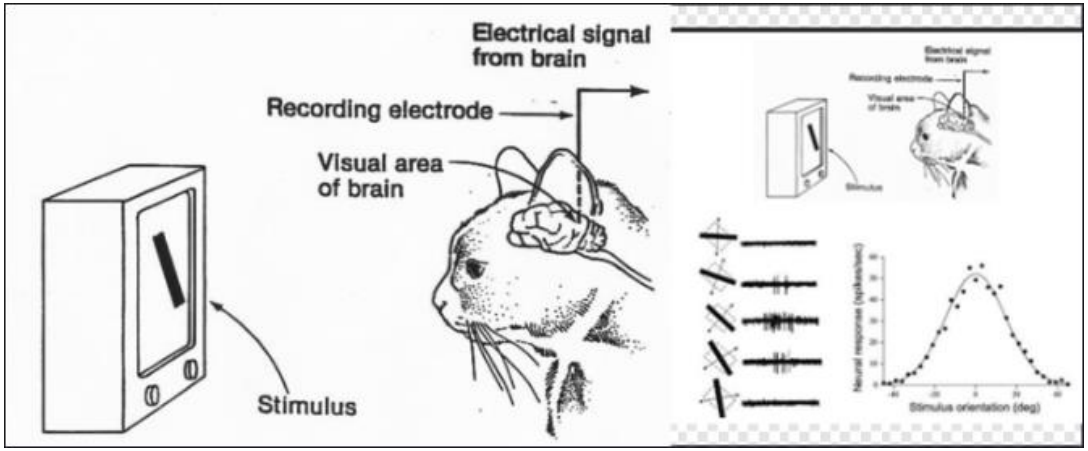


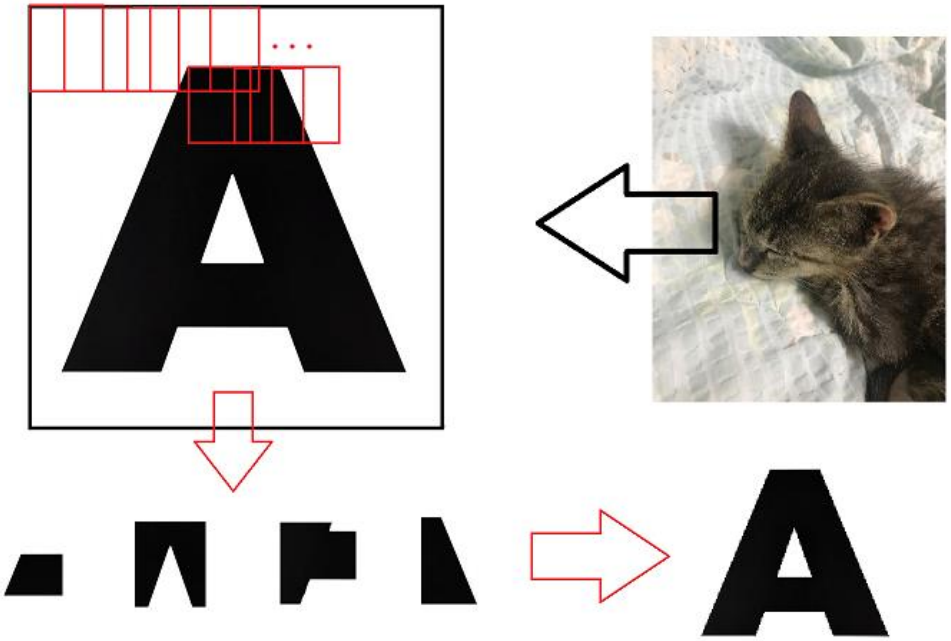
28 X 28 = 784

1024 X 768 = 786,432

출처: https://wikidocs.net/64066, https://www.kakaobrain.com/blog/9

2. CNN 이란?

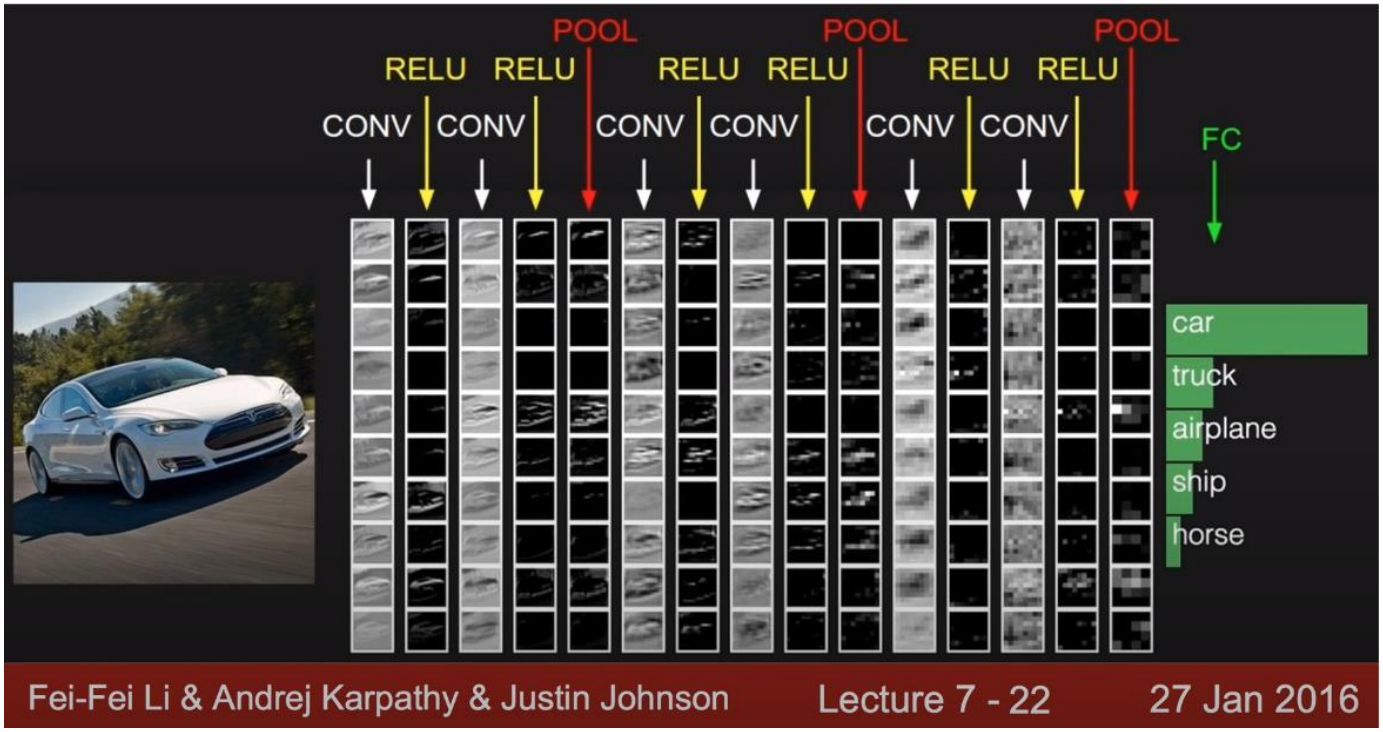




출처 : https://childult-programmer.tistory.com/47

2. CNN 이란?

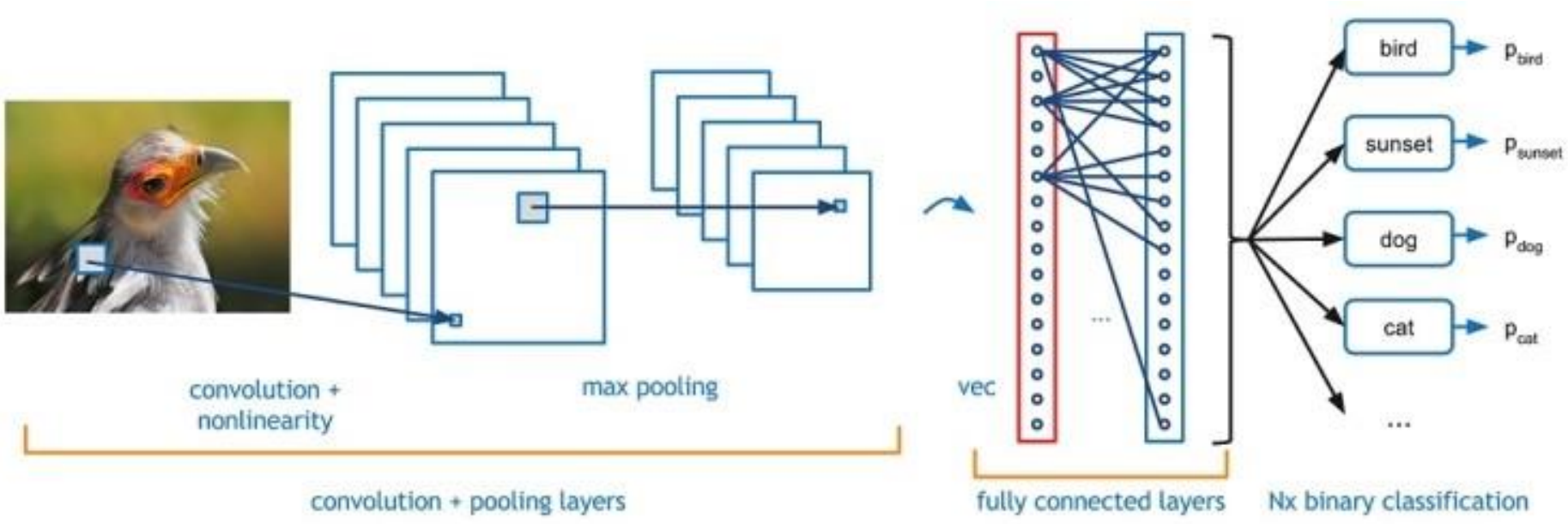
- Convolutional Neural Networks(CNN)
  - 시각적 영상을 분석하는 데 사용되는 인공신경망의 한 종류
  - 영상 및 동영상 인식, 추천 시스템, 영상 분류, 의료 영상 분석, 자연어 처리 등 다양한 분야에서 사용



출처: https://wikidocs.net/64066, https://childult-programmer.tistory.com/47

CNN(Convolutional Neural Network)

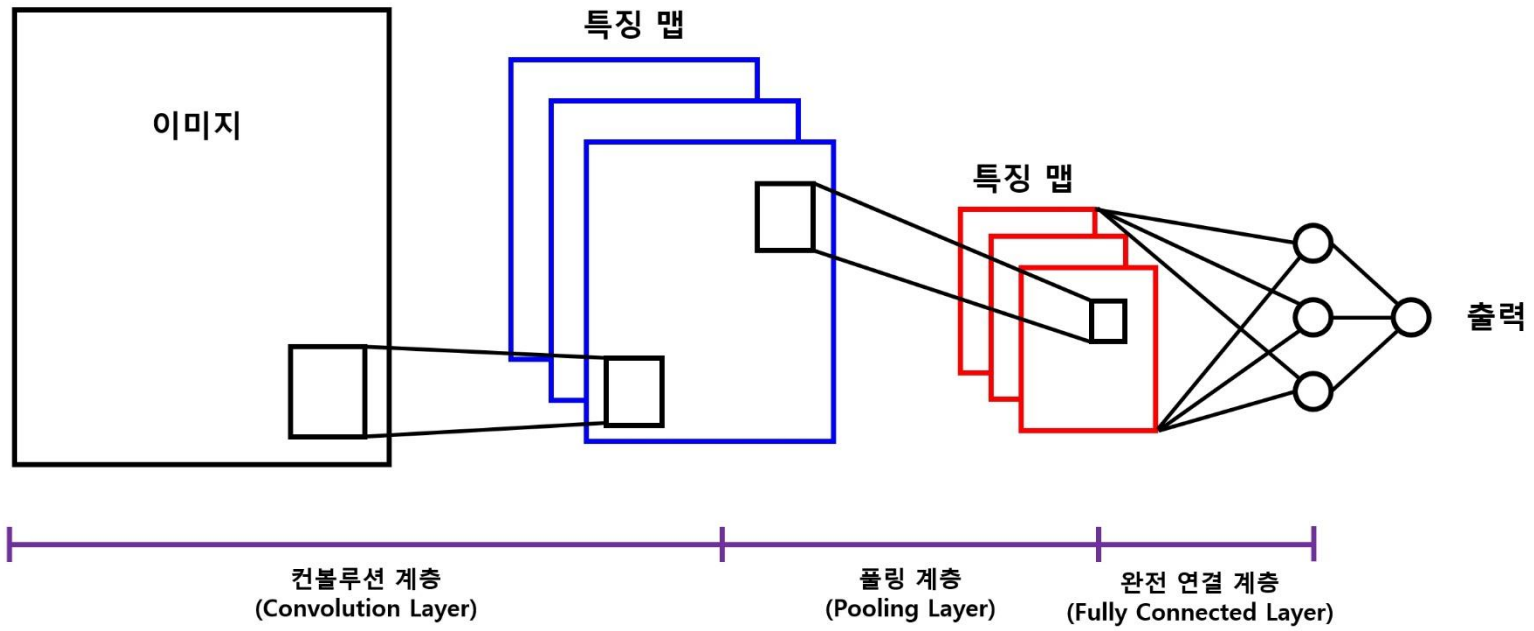
- 1998년 얀 레쿤(Yann LeCun) 교수가 소개
- 널리 사용되는 신경망
- 이미지 인식 분야에서는 거의 은총알이라고 할 정도로 강력한 성능을 발휘
- 손글씨 숫자 인식의 경우 99% 이상의 정확도 달성



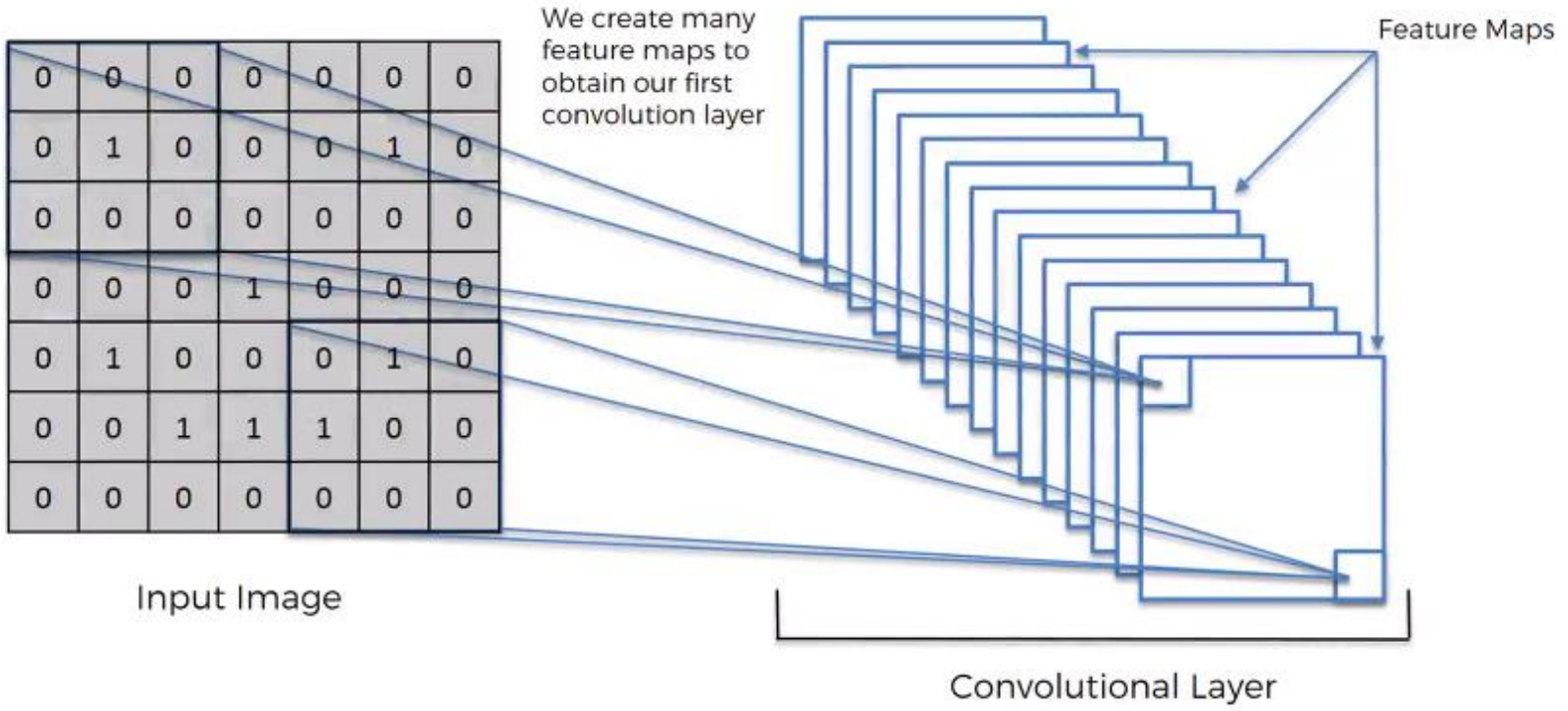
CNN 개념

컨볼루션 계층과 풀링계층

- 2차원의 평면 행렬에서 지정한 영역의 값들을 하나의 값으로 압축
- 단, 하나의 값으로 압축할 때 컨볼루션 계층은 가중치와 편향을 적용하고, 풀링계층은 단순히 값들 중 하나를 선택



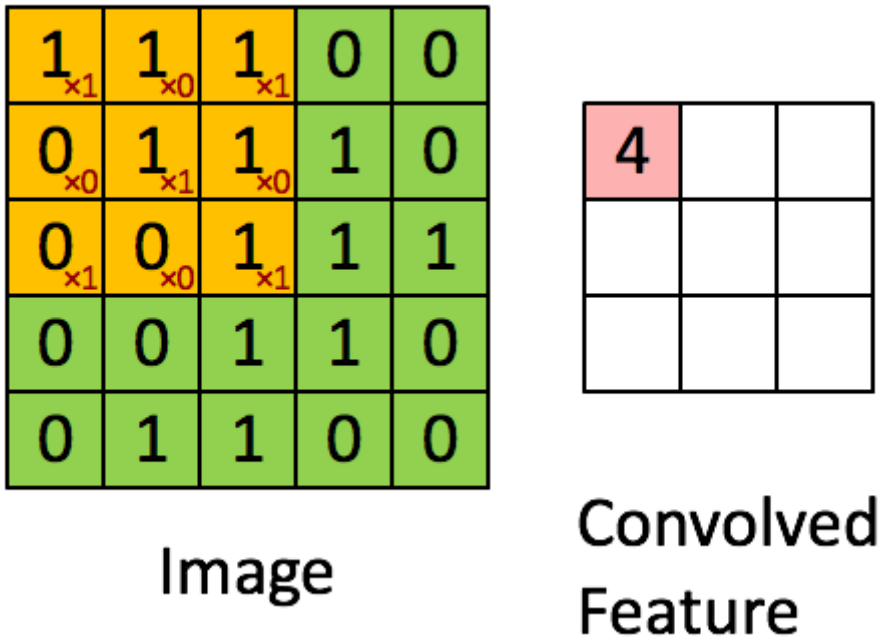
2. CNN 이란?



출처 : https://talkingaboutme.tistory.com/entry/DL-Convolution%EC%9D%98-%EC%A0%95%EC%9D%98

2. CNN 이란?

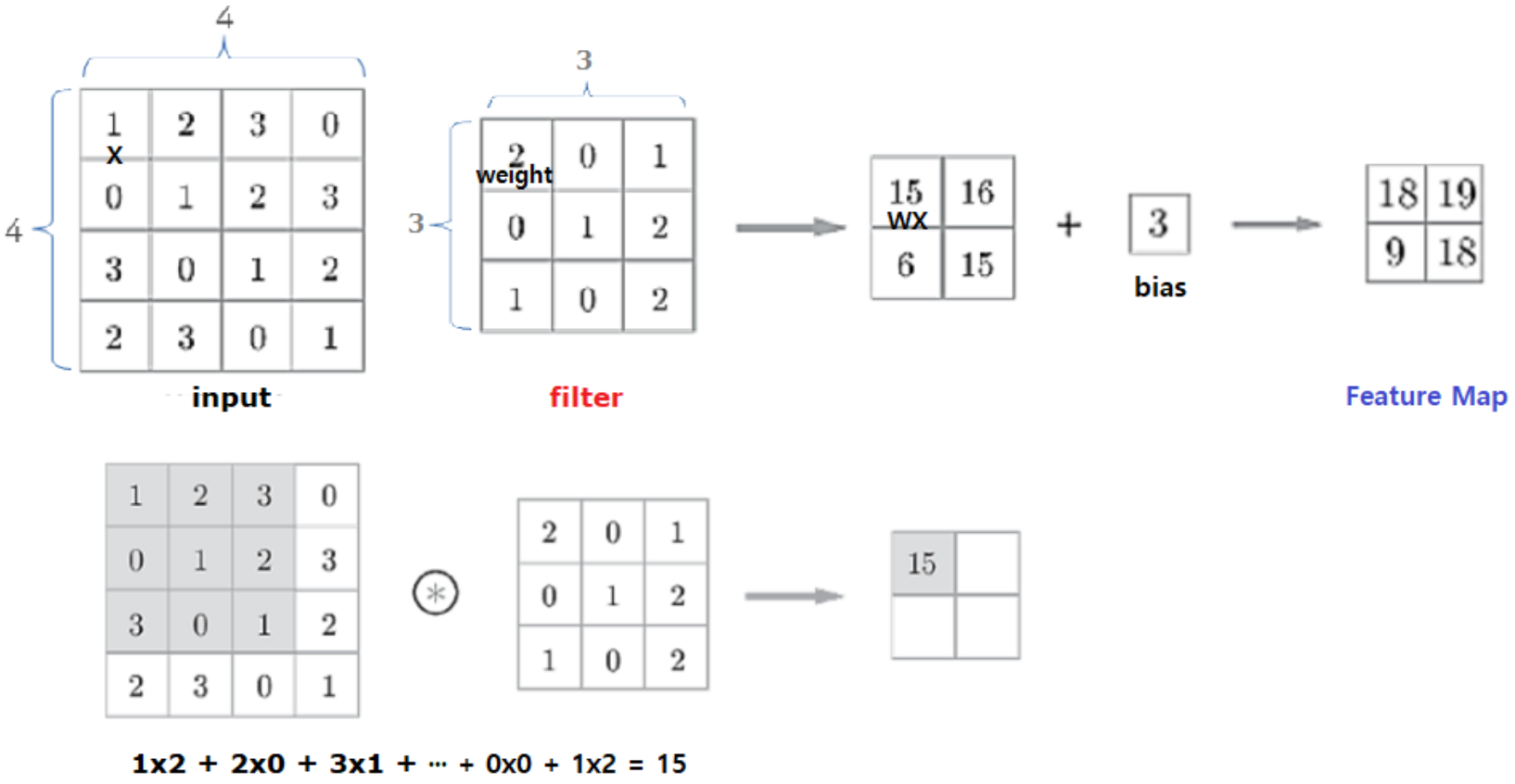
- Convolution에서 Feature Map을 만들어 내는 과정



출처 : https://childult-programmer.tistory.com/47

2. CNN 이란?

- Convolution에서 Feature Map을 만들어 내는 과정



출처 : https://childult-programmer.tistory.com/47

2. CNN 이란?

- Convolution에서 Feature Map을 만들어 내는 과정

-1	0	+1
-2	0	+2
-1	0	+1

Sobel-X (vertical)

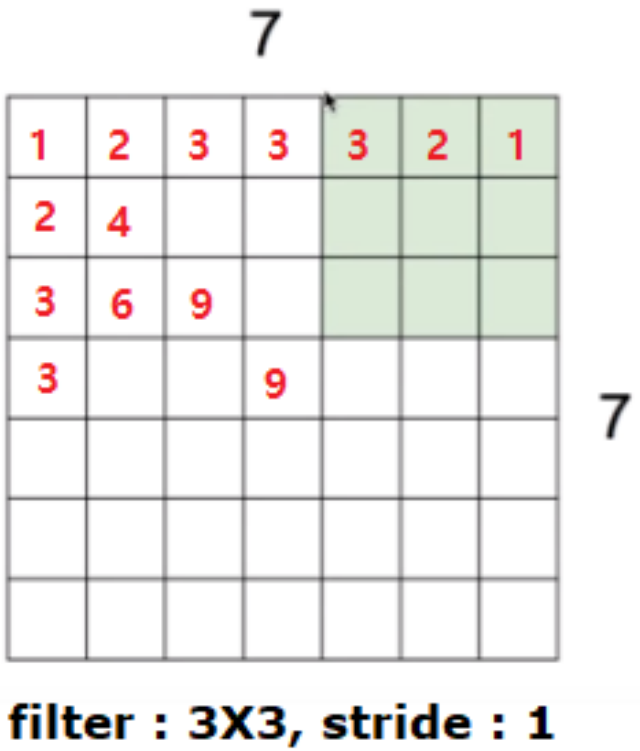
+1	+2	+1
0	0	0
-1	-2	-1

Sobel-Y (horizontal)

출처 : https://gruuuuu.github.io/machine-learning/cnn-doc/

2. CNN 이란?

- Padding이 필요한 이유
  - 모서리의 정보들의 연산이 가운데 정보들의 연산보다 적음

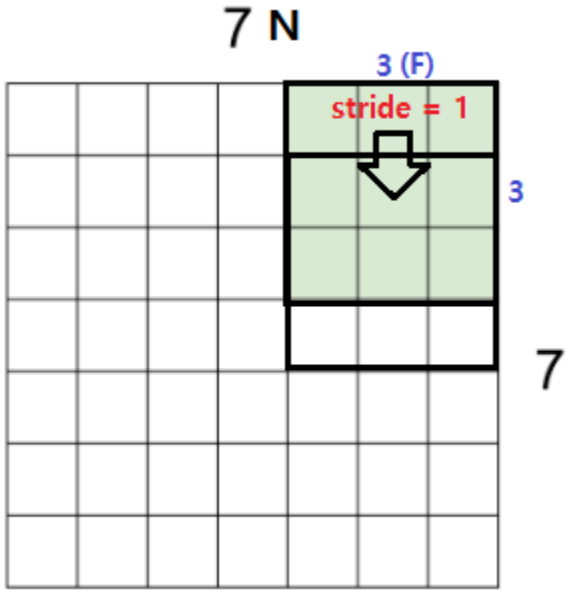


출처 : https://childult-programmer.tistory.com/47

2. CNN 이란?

- Padding이 필요한 이유
  - 전체 이미지 사이즈와 Filter 사이즈, Stride를 알면 Filter map의 사이즈 계산이 가능
  - Filter Map이 작아지는 특징이 있음

  ※ Stride - Filter가 계산하며 이동하는 칸 수



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

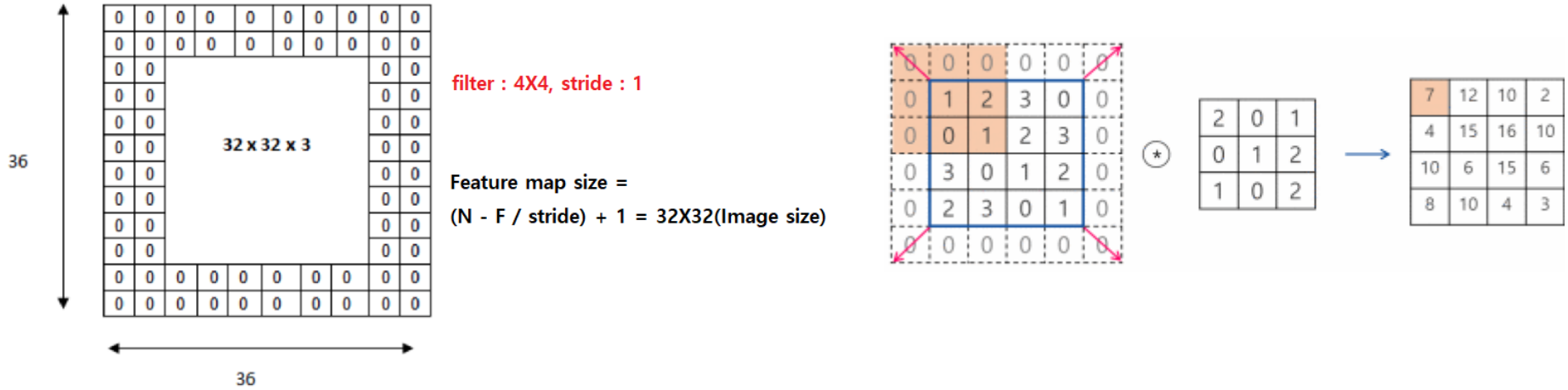
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

출처 : https://childult-programmer.tistory.com/47

2. CNN 이란?

- Padding이란?
  - 앞서 설명한 문제점을 방지하고자 합성 곱 연산 전에 선 처리하는 작업
  - 입력 데이터 주변을 특정 값(Hyper-Parameter)로 채워 공간적 크기를 조절하는 방법
  - 주로 Hyper-Parameter를 0으로 하는 Zero-Padding을 많이 사용
  - 덮는 층을 얼마나 두껍게 할지를 조절하여 Feature map을 원하는 사이즈로 조절 가능



※ Padding에서 0의 의미 : 의미 없는 정보, 가중치(W)는 0의 정보를 무시하게 된다.

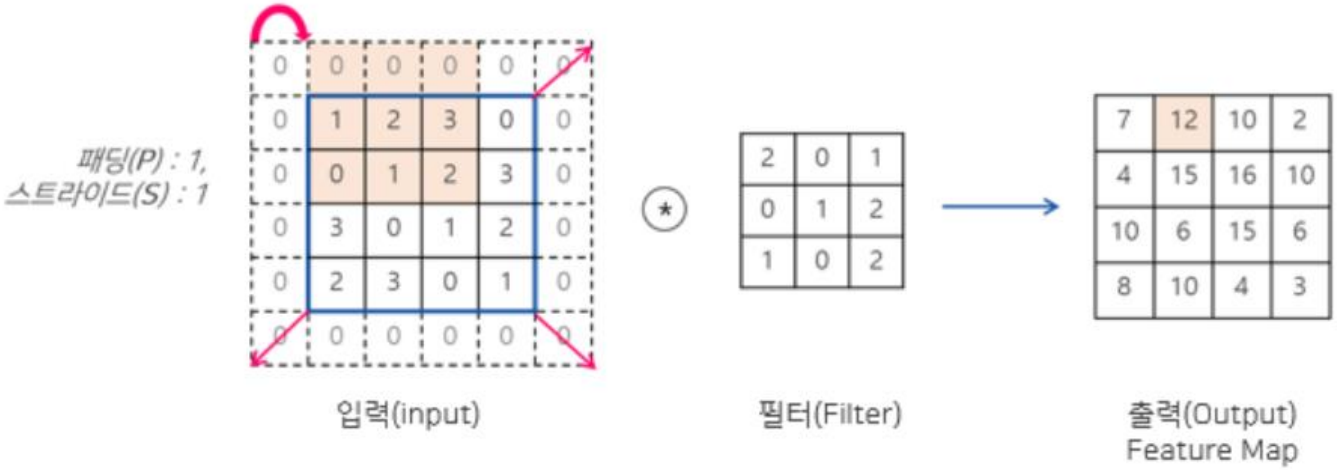
출처 : https://childult-programmer.tistory.com/47

2. CNN 이란?

- 출력 크기 계산 공식

$$(OH, OW) = \left(\frac{H + 2P - FH}{S} + 1, \frac{W + 2P - FW}{S} + 1 \right)$$

(H, W) : 입력 크기(input size)
(FH, FW) : 필터 크기 (filter/kernel size)
S : 스트라이드(stride)
P : 패딩(padding)
(OH, OW) : 출력 크기 (outpu size)



$$(OH, OW) = \left(\frac{4 + 2 \times 1 - 3}{1} + 1, \frac{4 + 2 \times 1 - 3}{1} + 1 \right) = (4, 4)$$

출처 : https://excelsior-cjh.tistory.com/180

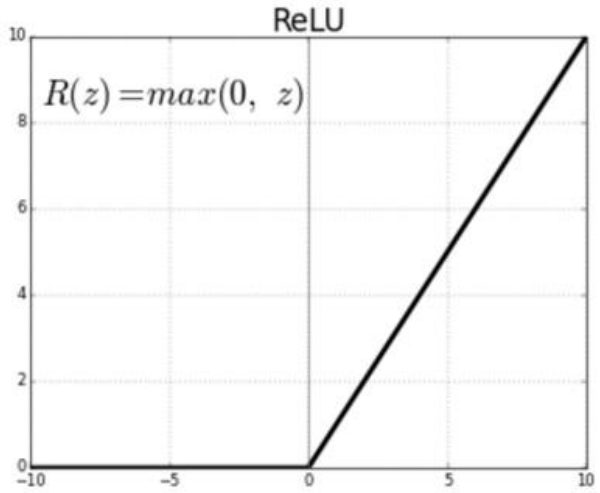
2. CNN 이란?

- **활성화 함수(Activation Function)**
  - 필터 들을 통해 특징맵이 추출되면 이 특징맵에 활성화 함수를 적용하여 값을 활성화 시킨다.
  - 예를 들면, 곡선 값의 특징을 가지고 있는 필터를 통해 입력 데이터를 적용시키면 정량적인 값이 나오는데 이러한 값들을 "곡선 특징을 얼마나 어떻게 가지고 있는지", "없다" 로 바꾸어 주는 과정이 필요하다.

  - CNN 에서는 주로 ReLU 함수를 사용한다.
  - Sigmoid, Tanh 함수대신 ReLU 함수를 사용하는 이유

    1) 층을 깊게 가져가는 CNN 특성상 역전파 과정에서 Gradient Vanishing 문제가 발생

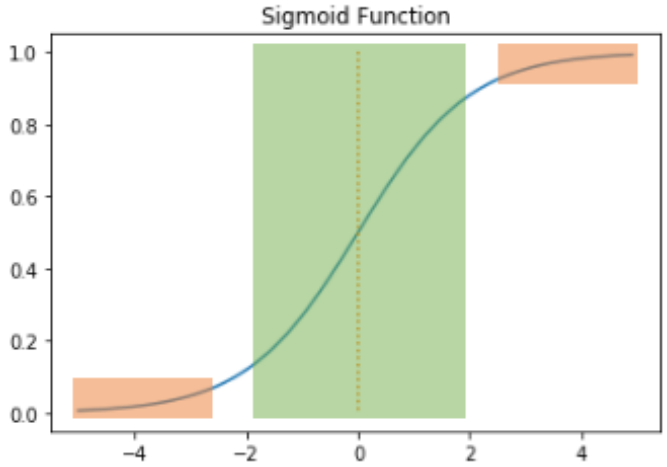
    2) 속도가 다른 활성화 함수들 보다 빠름

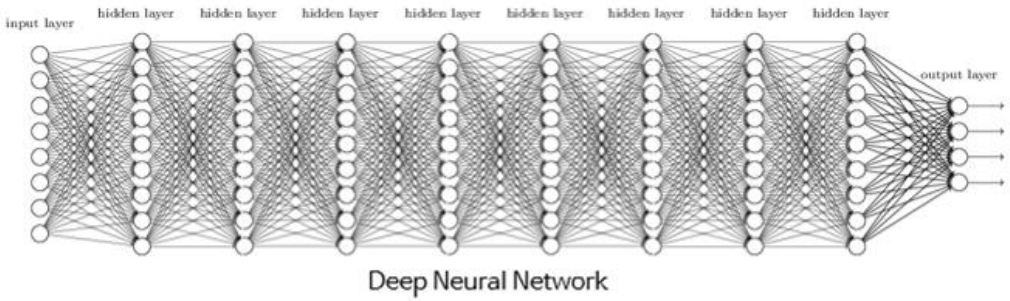


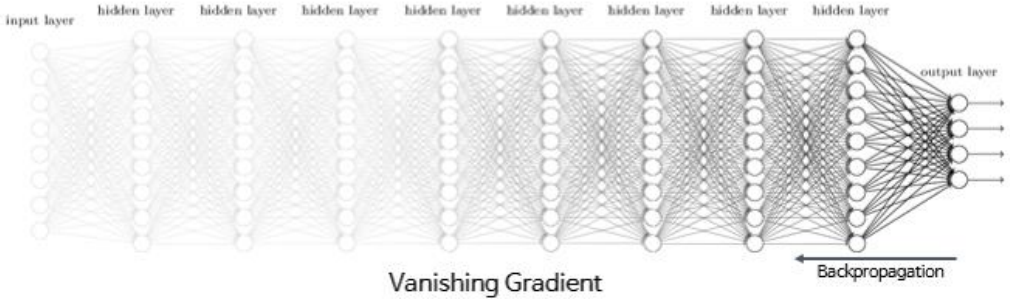
출처 : https://eehoeskrap.tistory.com/281

2. CNN 이란?

- Gradient Vanishing



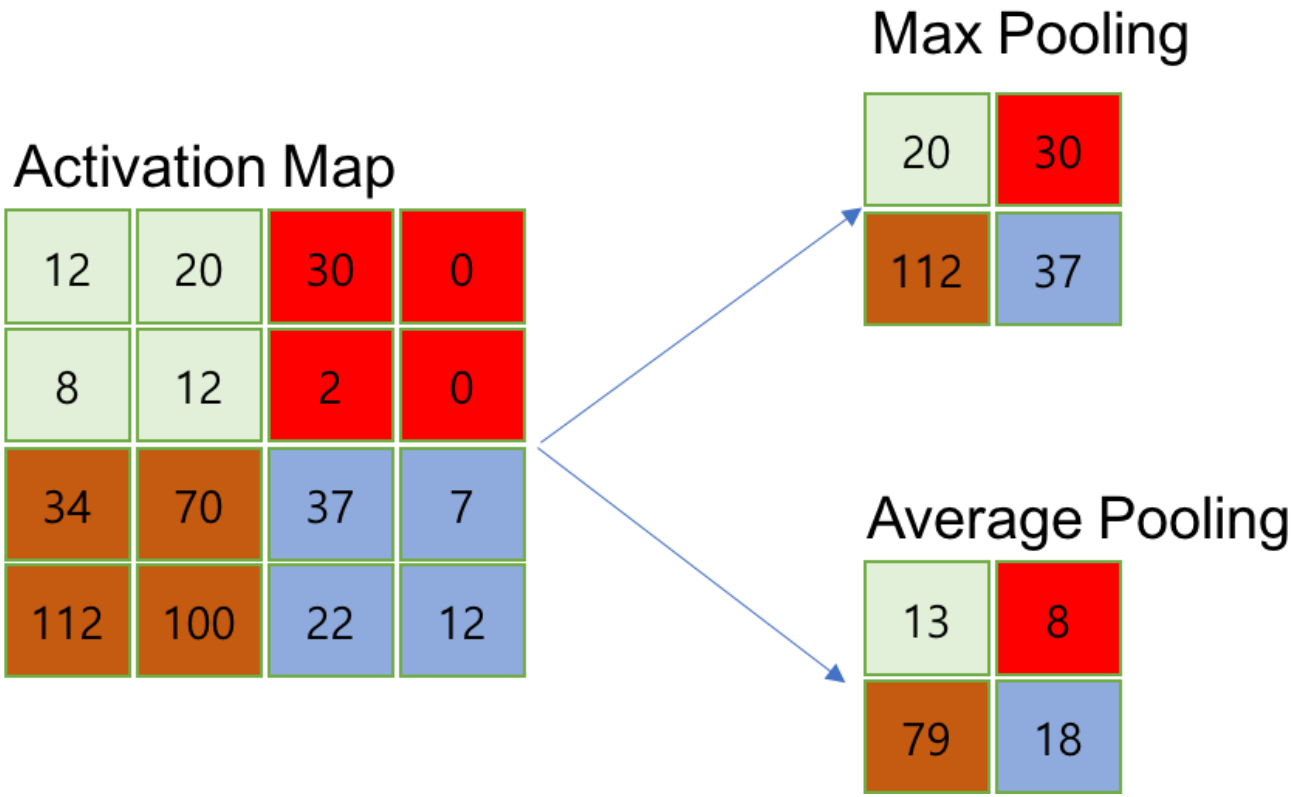




출처

2. CNN 이란?

- Pooling이란?
  - Convolution Layer의 출력 데이터를 입력으로 받아 출력 데이터의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용
  - Max, Average, Min Pooling 이 있음 ※ 주로 Max Pooling 사용



출처 : http://taewan.kim/post/cnn/

2. CNN 이란?

- Pooling Layer의 특징
  - Parameter가 없어 학습이 이루어지지 않고, Hyper-Parameter만 조정가능

※ Max Pooling과 Average Pooling의 차이점

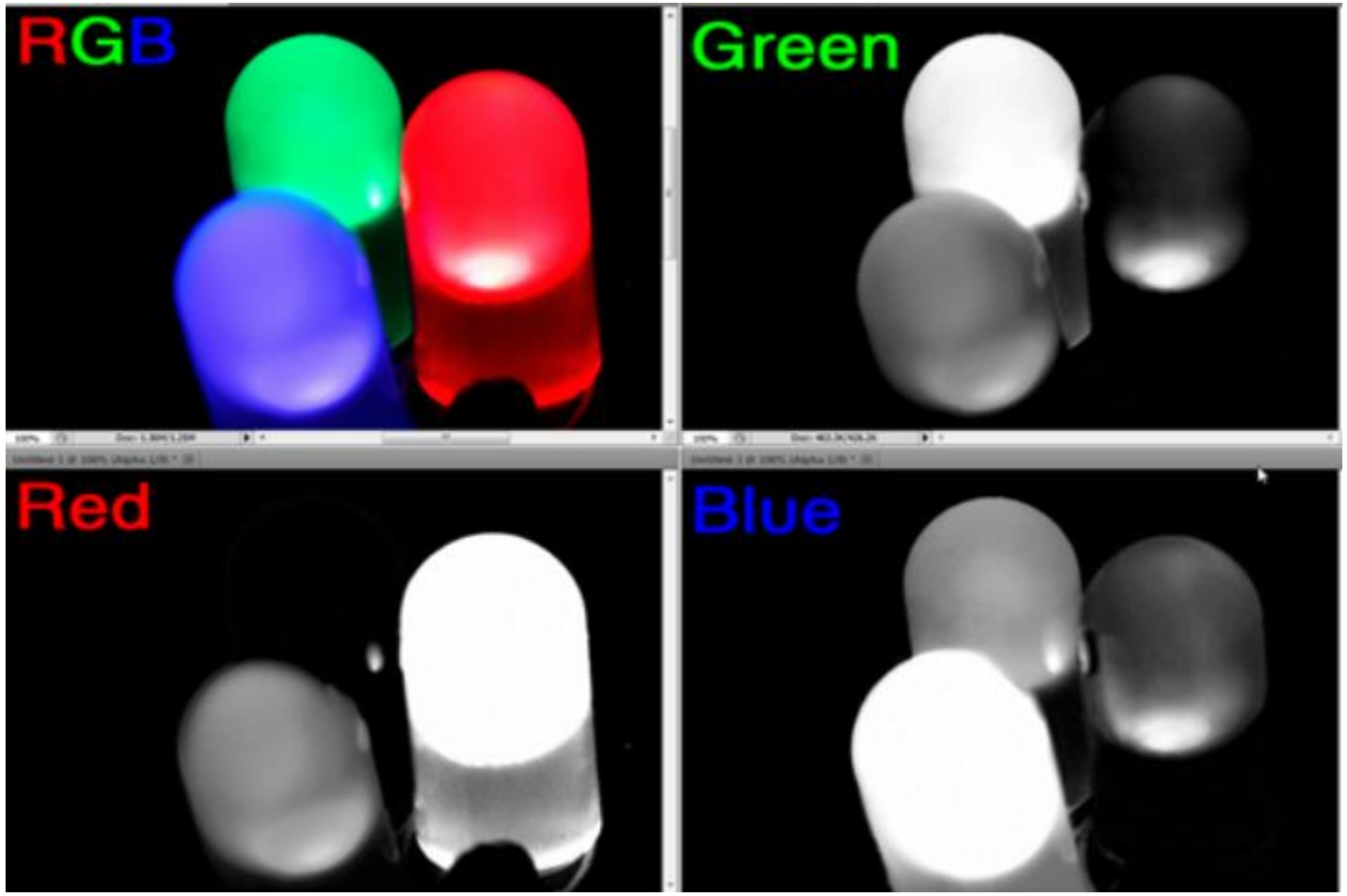
- Max Pooling
  - Input Layer에 가까울 수록 필요 없는 정보 혹은 중복된 정보들이 많음
  - 즉 Noise가 많아 Max Pooling을 사용하여 Special한 정보만 추출

- Average Pooling
  - Output Layer에 가까울 수록 필요한 정보 혹은 의미 있는 정보들이 많음
  - 이러한 정보들의 유실을 막기 위해 모든 정보의 평균을 구하는 Average Pooling 사용

출처 : https://warm-uk.tistory.com/47?category=824848

2. CNN 이란?

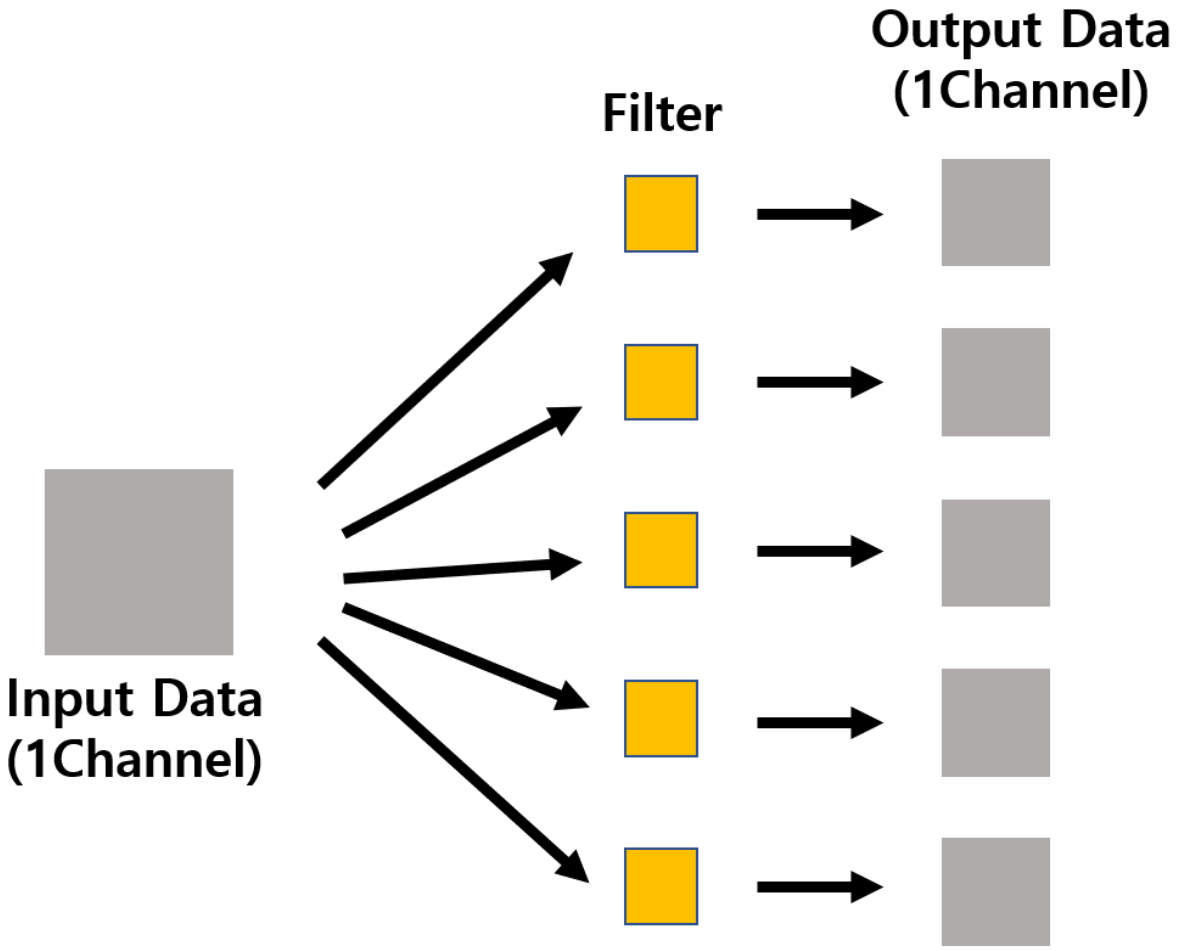
- Channel



출처 : https://gruuuuu.github.io/machine-learning/cnn-doc/

2. CNN 이란?

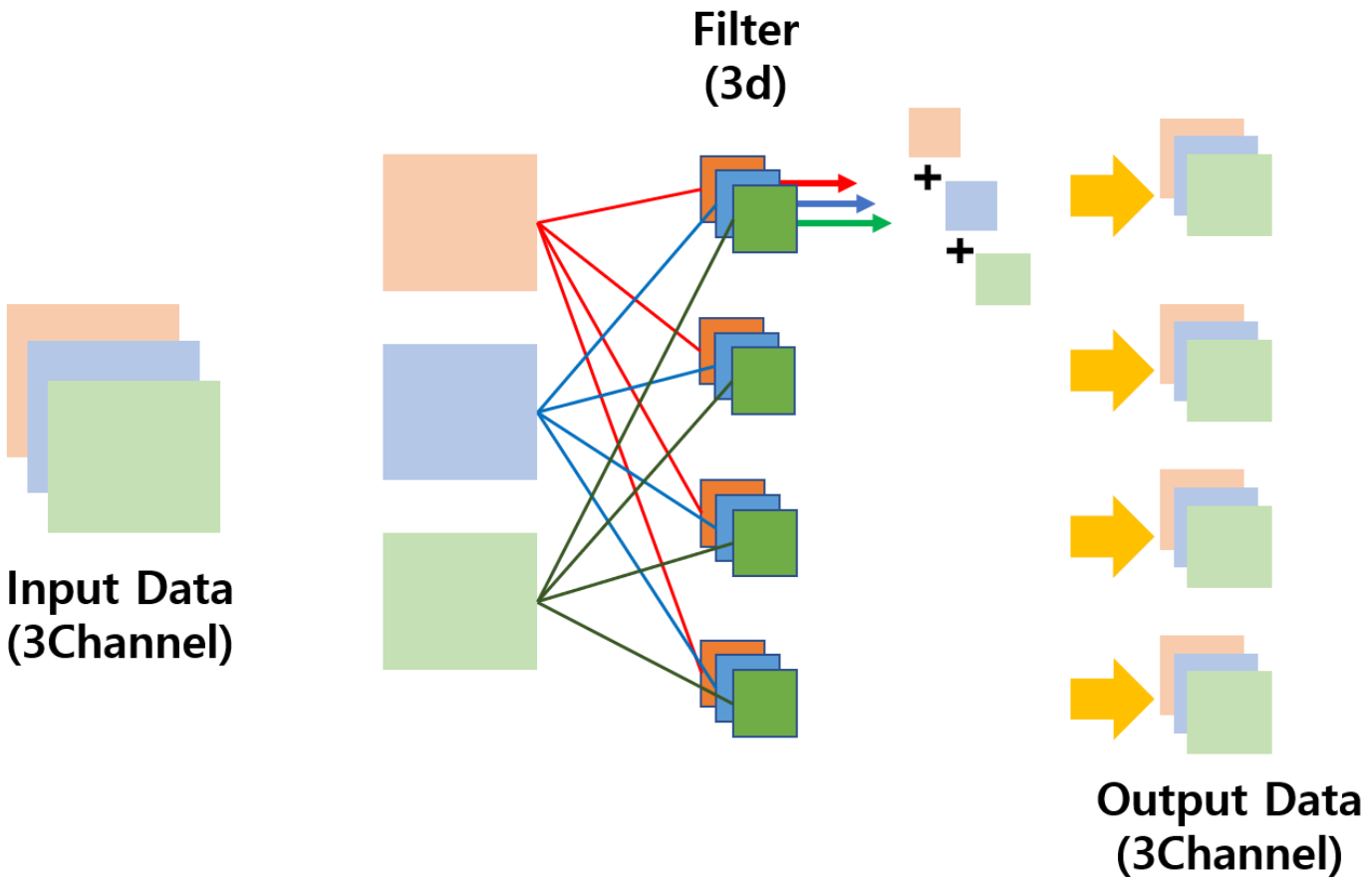
- Channel / 흑백인 경우



출처 : https://gruuuuu.github.io/machine-learning/cnn-doc/

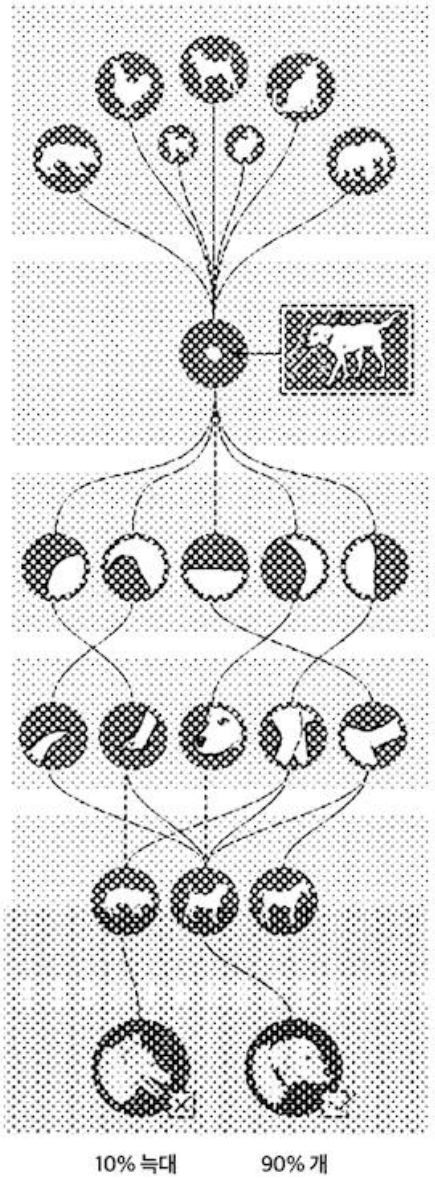
2. CNN 이란?

- Channel / 컬러인 경우



출처 : https://gruuuuu.github.io/machine-learning/cnn-doc/

2. CNN 이란?



훈련(training)
신경망은 동물 라벨이 붙여진 수천개의 사진을 가지고 분류하는 방법을 배운다.

입력(input)
훈련받은 신경망에 라벨이 붙지 않은 이미지를 보여준다.

입력층(first layer)
뉴런이 가장자리(edge)와 같은 단순한 형상(simple shape)에 반응한다.

은닉층(higher layer)
뉴런이 더 복잡한 조합의 특징(feature)에 반응한다.

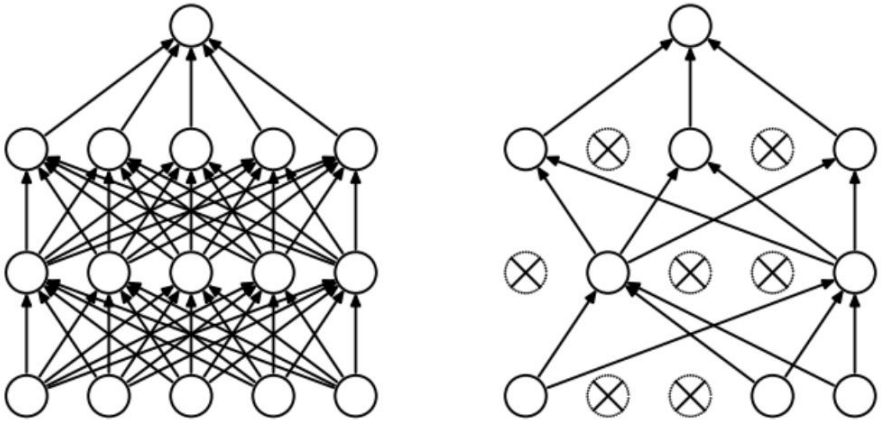
출력층(top layer)
뉴런이 좀 더 복잡하면서도 동물을 구분할 수 있는 추상화 개념에 반응한다.

출력
신경망은 자신이 훈련받은 내용에 근거해 사진 속 동물을 예측한다.

출처 : https://www.kakaobrain.com/blog/9

3. Dropout

- Dropout이란?



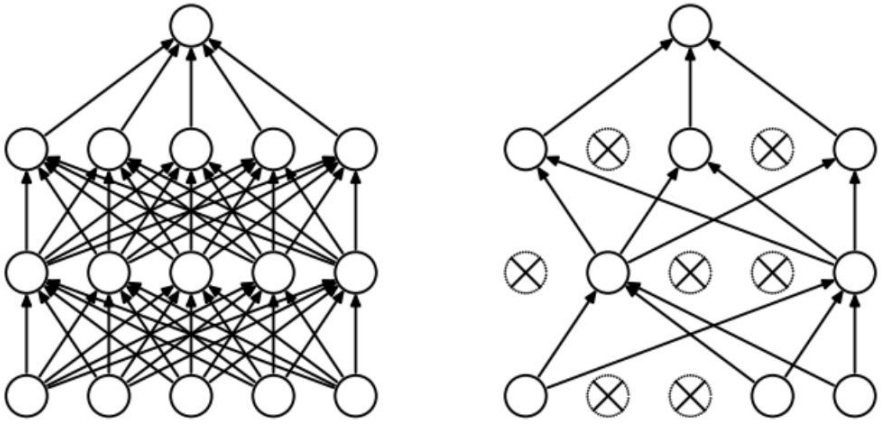
(a) Standard Neural Net (b) After applying dropout.

- 그림 (a)에 대한 학습을 진행할 때, 망에 있는 모든 Node들에 대한 학습을 수행하지 않고,
- 그림 (b)와 같이 일부 Node를 무작위로 생략하고 학습을 수행.
- 생략되는 Node의 비율은 사용자가 임의로 정해야 함.
- 일정한 Batch 구간 학습 후 생략되는 Node를 재 선정

출처 :

3. Dropout

- Dropout효과



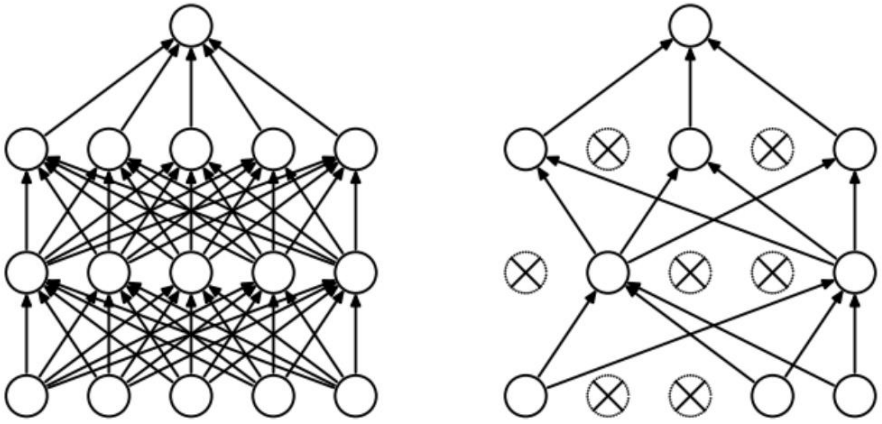
(a) Standard Neural Net (b) After applying dropout.

- Voting 효과
  - 일정한 batch 구간 동안 줄어든 망을 이용해 학습 시 줄어든 망에 대하여 오버피팅 되고,
  - 다음 Batch 구간 동안은 또 다른 줄어든 망에 대해 오버피팅 되어 Voting에 의한 평균 효과를 얻을 수 있음.
- Co-adaptation을 피하는 효과
  - 특정 노드의 가중치, 바이어스가 큰 값을 가지게 되면 특정 노드의 영향이 커져 다른 노드의 학습 속도가 느려지거나, 학습 진행이 잘 되지 않을 수 있음.
  - 즉 특정 노드에 영향을 받아 노드들이 서로 동조화(Co-adaptation) 되는 것을 피할 수 있음.

출처 :

3. Dropout

- Dropout 적용 방법



(a) Standard Neural Net (b) After applying dropout.

- 이번 Batch때 사용하지 않을 Node의 가중치를 0으로 임시 조정
- 또는 사용하지 않을 Node에 표식을 하여 표식이 있을 시 건너 뛰기

※ 테스트시에는 설정한 Dropout 확률 값 만큼 결과 값에 곱해줄 필요가 있음

출처 :

4. 실습

Fashion MNIST

- 옷, 신발 등의 사진을 분류하는 문제

```
from torchvision import datasets, transforms, utils
from torch.utils import data

import matplotlib.pyplot as plt
import numpy as np
```

- 필요한 라이브러리 임포트

```
transform = transforms.Compose([
    transforms.ToTensor()
])
```

```
trainset = datasets.FashionMNIST(
    root      = './.data/',
    train     = True,
    download  = True,
    transform = transform
)
testset = datasets.FashionMNIST(
    root      = './.data/',
    train     = False,
    download  = True,
    transform = transform
)
```

- 데이터 다운로드

4. 실습

Fashion MNIST

- 학습에 사용하기 편리하게 데이터 전처리

```
batch_size = 16

train_loader = data.DataLoader(
    dataset     = trainset,
    batch_size  = batch_size
)
test_loader = data.DataLoader(
    dataset     = testset,
    batch_size  = batch_size
)
```

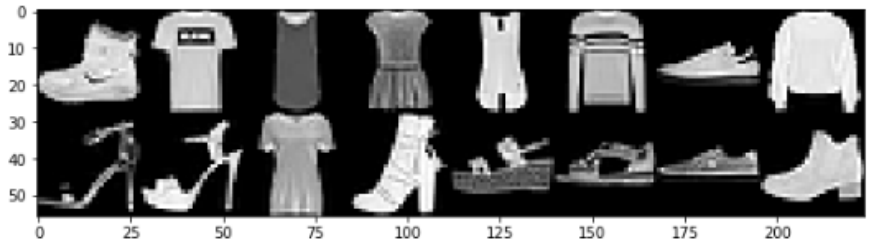
```
dataiter       = iter(train_loader)
images, labels = next(dataiter)
```

4. 실습

Fashion MNIST

- 학습 데이터 확인

```
img    = utils.make_grid(images, padding=0)
npimg = img.numpy()
plt.figure(figsize=(10, 7))
plt.imshow(np.transpose(npimg, (1,2,0)))
plt.show()
```



- 라벨링 확인

```
print(labels)
```

```
tensor([9, 0, 0, 3, 0, 2, 7, 2, 5, 5, 0, 9, 5, 5, 7, 9])
```

```
CLASSES = {
    0: 'T-shirt/top',
    1: 'Trouser',
    2: 'Pullover',
    3: 'Dress',
    4: 'Coat',
    5: 'Sandal',
    6: 'Shirt',
    7: 'Sneaker',
    8: 'Bag',
    9: 'Ankle boot'
}


for label in labels:
    index = label.item()
    print(CLASSES[index])
```

```
Ankle boot
T-shirt/top
T-shirt/top
Dress
T-shirt/top
Pullover
Sneaker
Pullover
Sandal
Sandal
T-shirt/top
Ankle boot
Sandal
Sandal
Sneaker
Ankle boot
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 필요한 라이브러리 임포트

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import transforms, datasets

USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
EPOCHS = 30
BATCH_SIZE = 64
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 데이터셋 불러오기

```
transform = transforms.Compose([
    transforms.ToTensor()
])
trainset = datasets.FashionMNIST(
    root      = './.data/',
    train     = True,
    download  = True,
    transform = transform
)
testset = datasets.FashionMNIST(
    root      = './.data/',
    train     = False,
    download  = True,
    transform = transform
)

train_loader = torch.utils.data.DataLoader(
    dataset     = trainset,
    batch_size  = BATCH_SIZE,
    shuffle     = True,
)
test_loader = torch.utils.data.DataLoader(
    dataset     = testset,
    batch_size  = BATCH_SIZE,
    shuffle     = True,
)
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분

입력 `x` 는 `[배치크기, 색, 높이, 넓이]` 로 이루어져 있습니다. `x.size()` 를 해보면 `[64, 1, 28, 28]` 이라고 표시되는 것을 보실 수 있습니다. Fashion MNIST에서 이미지의 크기는 28 x 28, 색은 흑백으로 1 가지 입니다. 그러므로 입력 x의 총 특성값 갯수는 28 x 28 x 1, 즉 784개 입니다.

우리가 사용할 모델은 3개의 레이어를 가진 인공신경망 입니다.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
model        = Net().to(DEVICE)
optimizer    = optim.SGD(model.parameters(), lr=0.01)
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분

```python
def train(model, train_loader, optimizer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # 학습 데이터를 DEVICE의 메모리로 보냄
        data, target = data.to(DEVICE), target.to(DEVICE)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
```

```python
def evaluate(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)

            # 모든 오차 더하기
            test_loss += F.cross_entropy(output, target,
                                         reduction='sum').item()

            # 가장 큰 값을 가진 클래스가 모델의 예측입니다.
            # 예측과 정답을 비교하여 일치할 경우 correct에 1을 더합니다.
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    return test_loss, test_accuracy
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 86.93%의 정확도를 보여준다.

```
for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer)
    test_loss, test_accuracy = evaluate(model, test_loader)

    print('[{}] Test Loss: {:.4f}, Accuracy: {:.2f}%'.format(
          epoch, test_loss, test_accuracy))
```

```
[1] Test Loss: 0.8437, Accuracy: 69.50%
[2] Test Loss: 0.6743, Accuracy: 75.98%
[3] Test Loss: 0.5830, Accuracy: 79.28%
[4] Test Loss: 0.5413, Accuracy: 80.55%
[5] Test Loss: 0.5131, Accuracy: 81.77%
[6] Test Loss: 0.5202, Accuracy: 80.67%
[7] Test Loss: 0.4880, Accuracy: 82.71%
[8] Test Loss: 0.4806, Accuracy: 83.13%
[9] Test Loss: 0.4625, Accuracy: 83.45%
[10] Test Loss: 0.4596, Accuracy: 83.49%
[11] Test Loss: 0.4620, Accuracy: 83.78%
[12] Test Loss: 0.4398, Accuracy: 84.48%
[13] Test Loss: 0.4384, Accuracy: 84.58%
[14] Test Loss: 0.4306, Accuracy: 84.76%
[15] Test Loss: 0.4295, Accuracy: 84.92%
[16] Test Loss: 0.4197, Accuracy: 85.25%
[17] Test Loss: 0.4381, Accuracy: 84.70%
[18] Test Loss: 0.4056, Accuracy: 85.57%
[19] Test Loss: 0.4151, Accuracy: 85.27%
[20] Test Loss: 0.4023, Accuracy: 85.72%
[21] Test Loss: 0.4096, Accuracy: 85.52%
[22] Test Loss: 0.3959, Accuracy: 85.98%
[23] Test Loss: 0.3995, Accuracy: 85.99%
[24] Test Loss: 0.4055, Accuracy: 85.72%
[25] Test Loss: 0.3880, Accuracy: 86.21%
[26] Test Loss: 0.3828, Accuracy: 86.40%
[27] Test Loss: 0.4017, Accuracy: 85.58%
[28] Test Loss: 0.3839, Accuracy: 86.20%
[29] Test Loss: 0.3696, Accuracy: 86.93%
[30] Test Loss: 0.3715, Accuracy: 86.79%
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 앞서 배운 정규화와 드롭아웃을 통해 정확도를 높여보자

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import transforms, datasets
```

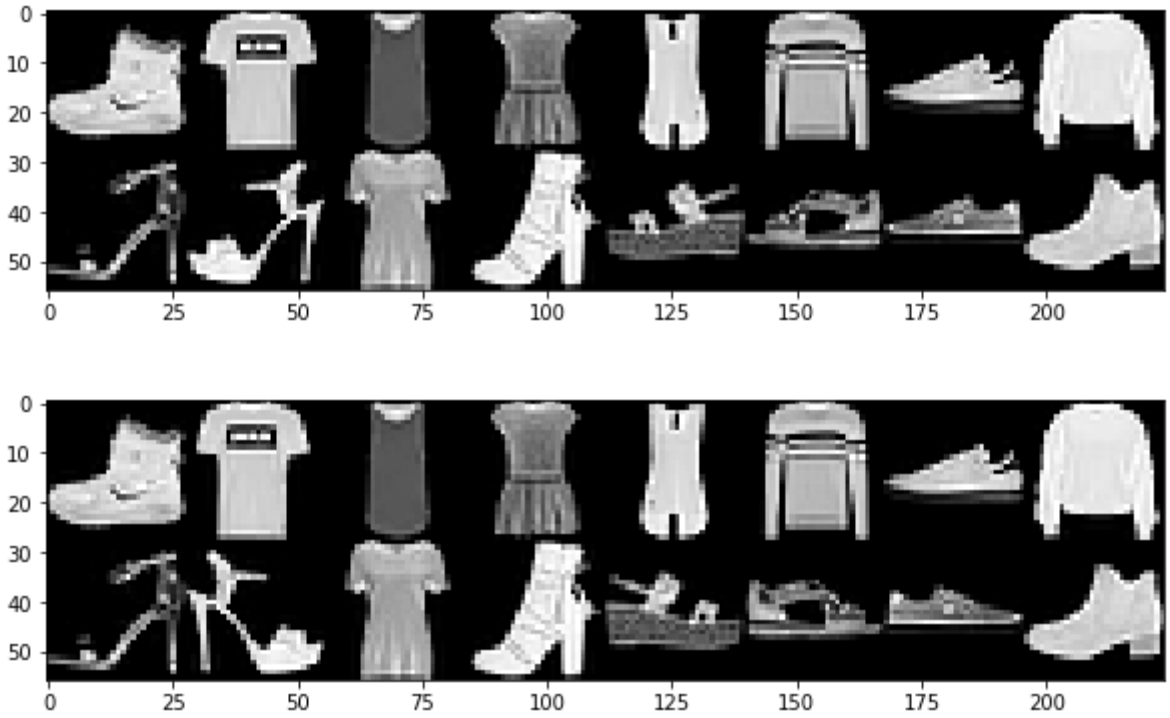
```
USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
```

```
EPOCHS = 50
BATCH_SIZE = 64
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 앞서 배운 정규화와 드롭아웃을 통해 정확도를 높여보자
- 데이터셋의 모양을 바꾸고 노이즈를 추가하여 데이터의 양을 증가시킨다.



4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 앞서 배운 정규화와 드롭아웃을 통해 정확도를 높여보자
- 데이터셋의 모양을 바꾸고 노이즈를 추가하여 데이터의 양을 증가시킨다.

```
train_loader = torch.utils.data.DataLoader(
    datasets.FashionMNIST('./.data',
                   train=True,
                   download=True,
                   transform=transforms.Compose([
                       transforms.RandomHorizontalFlip(),
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.FashionMNIST('./.data',
                   train=False,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 앞서 배운 정규화와 드롭아웃을 통해 정확도를 높여보자
- 모델에 드롭아웃 추가

```python
class Net(nn.Module):
    def __init__(self, dropout_p=0.2):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        # 드롭아웃 확률
        self.dropout_p = dropout_p

    def forward(self, x):
        x = x.view(-1, 784)
        x = F.relu(self.fc1(x))
        # 드롭아웃 추가
        x = F.dropout(x, training=self.training,
                      p=self.dropout_p)
        x = F.relu(self.fc2(x))
        # 드롭아웃 추가
        x = F.dropout(x, training=self.training,
                      p=self.dropout_p)
        x = self.fc3(x)
        return x
```

```python
model        = Net(dropout_p=0.2).to(DEVICE)
optimizer    = optim.SGD(model.parameters(), lr=0.01)
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 학습 함수, 평과 함수

```python
def train(model, train_loader, optimizer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(DEVICE), target.to(DEVICE)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
```

```python
def evaluate(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)
            test_loss += F.cross_entropy(output, target,
                                         reduction='sum').item()

            # 맞춘 갯수 계산
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    return test_loss, test_accuracy
```

4. 실습

Fashion MNIST

- 인공신경망으로 패션 아이템 구분
- 88.98% 정확도를 보여준다. 2% 정확도 상승

```
for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer)
    test_loss, test_accuracy = evaluate(model, test_loader)

    print('[{}] Test Loss: {:.4f}, Accuracy: {:.2f}%'.format(
          epoch, test_loss, test_accuracy))
```

```
[1] Test Loss: 0.6486, Accuracy: 77.37%
[2] Test Loss: 0.5417, Accuracy: 80.26%
[3] Test Loss: 0.4887, Accuracy: 82.27%
[4] Test Loss: 0.4639, Accuracy: 83.17%
[5] Test Loss: 0.4424, Accuracy: 84.02%
[6] Test Loss: 0.4253, Accuracy: 84.70%
[7] Test Loss: 0.4214, Accuracy: 84.63%
[8] Test Loss: 0.4077, Accuracy: 85.06%
[9] Test Loss: 0.4085, Accuracy: 84.99%
[10] Test Loss: 0.3914, Accuracy: 85.86%
[11] Test Loss: 0.3899, Accuracy: 85.90%
[12] Test Loss: 0.3774, Accuracy: 86.36%
[13] Test Loss: 0.3809, Accuracy: 86.33%
[14] Test Loss: 0.3718, Accuracy: 86.55%
[15] Test Loss: 0.3686, Accuracy: 86.57%
[16] Test Loss: 0.3628, Accuracy: 86.86%
[17] Test Loss: 0.3622, Accuracy: 86.67%
[18] Test Loss: 0.3585, Accuracy: 87.04%
[19] Test Loss: 0.3562, Accuracy: 86.94%
[20] Test Loss: 0.3496, Accuracy: 87.51%
[21] Test Loss: 0.3502, Accuracy: 87.31%
[22] Test Loss: 0.3437, Accuracy: 87.68%
[23] Test Loss: 0.3459, Accuracy: 87.26%
[24] Test Loss: 0.3544, Accuracy: 87.20%
[25] Test Loss: 0.3383, Accuracy: 87.98%
[26] Test Loss: 0.3387, Accuracy: 88.03%
[27] Test Loss: 0.3408, Accuracy: 87.90%
[28] Test Loss: 0.3329, Accuracy: 88.07%
[29] Test Loss: 0.3393, Accuracy: 87.80%
[30] Test Loss: 0.3296, Accuracy: 88.25%
[31] Test Loss: 0.3277, Accuracy: 88.31%
[32] Test Loss: 0.3305, Accuracy: 88.23%
[33] Test Loss: 0.3253, Accuracy: 88.37%
[34] Test Loss: 0.3269, Accuracy: 88.26%
[35] Test Loss: 0.3261, Accuracy: 88.30%
[36] Test Loss: 0.3194, Accuracy: 88.87%
[37] Test Loss: 0.3207, Accuracy: 88.75%
[38] Test Loss: 0.3186, Accuracy: 88.55%
[39] Test Loss: 0.3216, Accuracy: 88.41%
[40] Test Loss: 0.3164, Accuracy: 88.50%
[41] Test Loss: 0.3183, Accuracy: 88.45%
[42] Test Loss: 0.3139, Accuracy: 88.77%
[43] Test Loss: 0.3161, Accuracy: 88.66%
[44] Test Loss: 0.3112, Accuracy: 88.89%
[45] Test Loss: 0.3105, Accuracy: 89.05%
[46] Test Loss: 0.3138, Accuracy: 88.82%
[47] Test Loss: 0.3135, Accuracy: 88.73%
[48] Test Loss: 0.3128, Accuracy: 88.98%
[49] Test Loss: 0.3183, Accuracy: 88.52%
[50] Test Loss: 0.3213, Accuracy: 88.53%
```

4. 실습

Fashion MNIST

- CNN(Convolutional Neural Network)를 사용하여 패션아이템 구분 성능 높이기
- 필요한 라이브러리 임포트

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import transforms, datasets
USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
EPOCHS     = 40
BATCH_SIZE = 64
```

4. 실습

Fashion MNIST

- CNN(Convolutional Neural Network)를 사용하여 패션아이템 구분 성능 높이기
- 데이터 셋 불러오기

```
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./.data',
                   train=True,
                   download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./.data',
                   train=False,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
```

4. 실습

Fashion MNIST

- CNN(Convolutional Neural Network)를 사용하여 패션아이템 구분 성능 높이기
- 모델 구축

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return x
```

```
model     = Net().to(DEVICE)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
```

4. 실습

Fashion MNIST

- CNN(Convolutional Neural Network)를 사용하여 패션아이템 구분 성능 높이기
- 학습 함수와 평가 함수

```python
def train(model, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(DEVICE), target.to(DEVICE)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % 200 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

```python
def evaluate(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)

            # 배치 오차를 합산
            test_loss += F.cross_entropy(output, target,
                                         reduction='sum').item()

            # 가장 높은 값을 가진 인덱스가 바로 예측값
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    return test_loss, test_accuracy
```

4. 실습

Fashion MNIST

- CNN(Convolutional Neural Network)를 사용하여 패션아이템 구분 성능 높이기
- 정확도가 99% 까지 나오는 것을 확인할 수 있다.

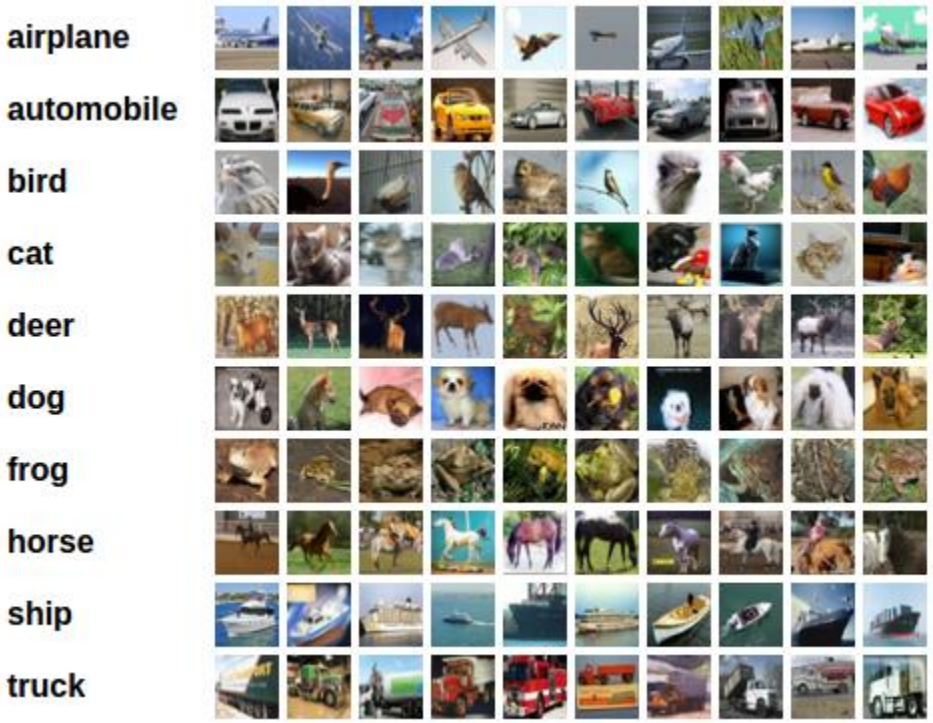
```
for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer, epoch)
    test_loss, test_accuracy = evaluate(model, test_loader)

    print('[{}] Test Loss: {:.4f}, Accuracy: {:.2f}%'.format(
          epoch, test_loss, test_accuracy))
```

```
[37] Test Loss: 0.0310, Accuracy: 99.02%
Train Epoch: 38 [0/60000 (0%)]  Loss: 0.227352
Train Epoch: 38 [12800/60000 (21%)]     Loss: 0.249107
Train Epoch: 38 [25600/60000 (43%)]     Loss: 0.250444
Train Epoch: 38 [38400/60000 (64%)]     Loss: 0.073755
Train Epoch: 38 [51200/60000 (85%)]     Loss: 0.227937
[38] Test Loss: 0.0276, Accuracy: 99.15%
Train Epoch: 39 [0/60000 (0%)]  Loss: 0.194536
Train Epoch: 39 [12800/60000 (21%)]     Loss: 0.051924
Train Epoch: 39 [25600/60000 (43%)]     Loss: 0.095845
Train Epoch: 39 [38400/60000 (64%)]     Loss: 0.036518
Train Epoch: 39 [51200/60000 (85%)]     Loss: 0.172633
[39] Test Loss: 0.0292, Accuracy: 99.09%
Train Epoch: 40 [0/60000 (0%)]  Loss: 0.076790
Train Epoch: 40 [12800/60000 (21%)]     Loss: 0.155643
Train Epoch: 40 [25600/60000 (43%)]     Loss: 0.040076
Train Epoch: 40 [38400/60000 (64%)]     Loss: 0.114211
Train Epoch: 40 [51200/60000 (85%)]     Loss: 0.232939
[40] Test Loss: 0.0283, Accuracy: 99.09%
```

4. 실습

이미지 분류

- CIFAR-10 데이터 셋 / R, G, B 3채널의 32*32 크기의 이미지 데이터
- 총 10가지의 클래스로 이루어진 데이터셋



4. 실습

이미지 분류

- 데이터 셋 불러오기와, 정규화

```python
import torch
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

#데이터 불러오기, 학습여부 o
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)

#학습용 셋은 섞어서 뽑기
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)
#데이터 불러오기, 학습여부 x
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
#테스트 셋은 굳이 섞을 필요가 없음
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)
#클래스들
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data₩cifar-10-python.tar.gz
```

```
170500096it [00:25, 6725551.93it/s]
```

```
Extracting ./data₩cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

4. 실습

이미지 분류

- 이미지 데이터셋 확인

```python
import matplotlib.pyplot as plt
import numpy as np

#이미지 확인하기

def imshow(img):
    img = img / 2 + 0.5     # 정규화 해제
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# 학습용 이미지 뽑기
dataiter = iter(trainloader)
images, labels = dataiter.next()

# 이미지 보여주기
imshow(torchvision.utils.make_grid(images))

# 이미지별 라벨 (클래스) 보여주기
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```


```
bird  frog horse   car
```

4. 실습

이미지 분류

- CNN 정의하기 / Loss와 Optimizer 정의하기

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        #input = 3, output = 6, kernal = 5
        self.conv1 = nn.Conv2d(3, 6, 5)
        #kernal = 2, stride = 2, padding = 0 (default)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        #input feature, output feature
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    # 값 계산
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

```python
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

4. 실습

이미지 분류

- 학습하기

```python
for epoch in range(2):  #데이터셋 2번 받기

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # 입력 받기 (데이터 [입력, 라벨(정답)]으로 이루어짐)
        inputs, labels = data

        #학습
        #optimizer의 기울기를 0으로 만들기 (변화도가 누적되지 않게 하기 위해)
        optimizer.zero_grad()
        # output 구하기
        outputs = net(inputs)
        # loss 계산
        loss = criterion(outputs, labels)
        #backpropagation (기울기 계산)
        loss.backward()
        #업데이트
        optimizer.step()

        # 결과 출력
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000개마다
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')

#여기에 학습한 모델 저장
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

```
[1,  2000] loss: 2.163
[1,  4000] loss: 1.811
[1,  6000] loss: 1.673
[1,  8000] loss: 1.570
[1, 10000] loss: 1.496
[1, 12000] loss: 1.445
[2,  2000] loss: 1.381
[2,  4000] loss: 1.346
[2,  6000] loss: 1.322
[2,  8000] loss: 1.314
[2, 10000] loss: 1.280
[2, 12000] loss: 1.248
Finished Training
```

4. 실습

이미지 분류

- 테스트하기 / 4개의 이미지 중 3개를 맞춘 것을 확인할 수 있다.

```
dataiter = iter(testloader)
images, labels = dataiter.next()

# 실험용 데이터와 결과 출력
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))

# 학습한 모델로 예측값 뽑아보기
net = Net()
net.load_state_dict(torch.load(PATH))
outputs = net(images)
_, predicted = torch.max(outputs, 1)
print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```


```
GroundTruth:    cat  ship  ship plane
Predicted:    cat  ship   car plane
```

4. 실습

이미지 분류

- 모델 평가
- 모델의 정확도는 56%이며,
- 클래스별 정확도는 밑의 코드와 같다.

```python
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 56 %
```

```python
# 각 분류(class)에 대한 예측과 계산을 위해 준비
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# 변화도는 여전히 필요하지 않습니다
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # 각 분류별로 올바른 예측 수를 모읍니다
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


# 각 분류별 정확도(accuracy)를 출력합니다
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

```
Accuracy for class: plane is 71.3 %
Accuracy for class: car   is 77.3 %
Accuracy for class: bird  is 33.4 %
Accuracy for class: cat   is 55.5 %
Accuracy for class: deer  is 42.2 %
Accuracy for class: dog   is 22.3 %
Accuracy for class: frog  is 81.8 %
Accuracy for class: horse is 64.7 %
Accuracy for class: ship  is 64.7 %
Accuracy for class: truck is 53.8 %
```

4. 실습

이미지 분류

- 모델 성능 향상 방법
- CNN 수정
- 조금 더 복잡한 CNN 모델 구축
- Dropout 기법 사용
- 학습 횟수 증가

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 5)
        self.conv2 = nn.Conv2d(32, 64, 5)
        self.conv3 = nn.Conv2d(64, 128, 5)

        self.pool = nn.MaxPool2d(2, 2, 1)

        self.fc1 = nn.Linear(128 * 2 * 2, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 84)
        self.fc4 = nn.Linear(84, 10)

        self.dropout1 = nn.Dropout(p=0.5, inplace=False)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)

        x = F.relu(self.conv2(x))
        x = self.pool(x)

        x = F.relu(self.conv3(x))
        x = self.dropout1(x)
        x = self.pool(x)

        x = x.view(-1, 128 * 2 * 2)
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

4. 실습

이미지 분류

- 모델 성능 향상 방법
- CNN 수정
- 조금 더 복잡한 CNN 모델 구축
- Dropout 기법 사용
- 학습 횟수 증가

```python
for epoch in range(20):  #데이터셋 2번 받기

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # 입력 받기 (데이터 [입력, 라벨(정답)]으로 이루어짐)
        inputs, labels = data

        #학습
        #optimizer의 기울기를 0으로 만들기 (변화도가 누적되지 않게 하기 위해)
        optimizer.zero_grad()
        # output 구하기
        outputs = net(inputs)
        # loss 계산
        loss = criterion(outputs, labels)
        #backpropagation (기울기 계산)
        loss.backward()
        #업데이트
        optimizer.step()

        # 결과 출력
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000개마다
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')

#여기에 학습한 모델 저장
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

4. 실습

이미지 분류

- 정확도가 74% 까지 상승한 것을 확인할 수 있다.

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 74 %
```

```
# 각 분류(class)에 대한 예측값 계산을 위해 준비
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# 변화도는 여전히 필요하지 않습니다
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # 각 분류별로 올바른 예측 수를 모읍니다
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


# 각 분류별 정확도(accuracy)를 출력합니다
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

```
Accuracy for class: plane is 77.3 %
Accuracy for class: car   is 83.1 %
Accuracy for class: bird  is 70.3 %
Accuracy for class: cat   is 59.4 %
Accuracy for class: deer  is 64.7 %
Accuracy for class: dog   is 58.6 %
Accuracy for class: frog  is 82.5 %
Accuracy for class: horse is 80.2 %
Accuracy for class: ship  is 83.6 %
Accuracy for class: truck is 82.3 %
```

Q & A

Thank you