# Deep learning with pytorch

https://youtu.be/ks4k73PVy-Q

송경주

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

파이토치란?

선형회귀분석

다양한 신경망

# 파이토치란?

- PyTorch

- 파이썬 기반 오픈소스 머신러닝 라이브러리
- 페이스북 인공지능 연구집단 개발
- Neural Network 구현 가능

# 선형회귀분석

주어진 데이터들을 설명하기 위한 가장 적합한 직선을 찾는 것.

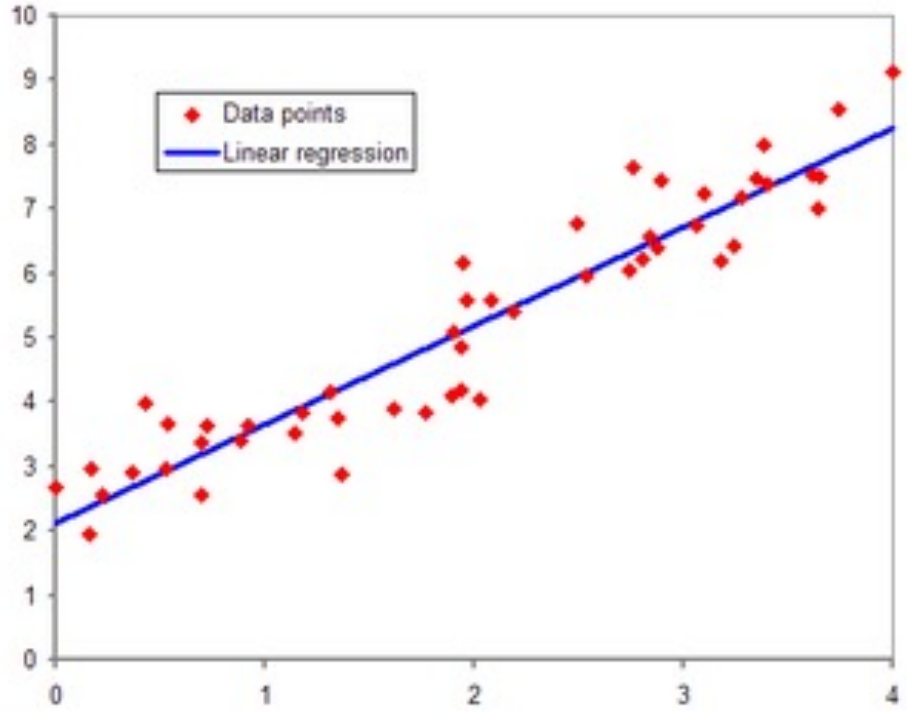
1. 단순선형회귀 : 하나의 독립변수
2. 다중선형회귀 : 여러 개의 독립변수

Example)

$y = wx + b$ 의 직선으로 많은 데이터를 가장 잘 표현하는

변수 w, b 찾기 (w: weight, b: bias)

→평균제곱오차(MSE) 등을 사용..



# 선형회귀분석

- 평균제곱오차(MSE)

-w, b를 찾으면서 데이터와 얼마나 일치하는지 계산하는 방법.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2$$

w, b 를 통해 예측한 결과값 $\hat{y}$ 와 실제 데이터 $y$ 의 차이를 제곱하여 평균을 낸다.

오차가 적을수록 데이터를 정확하게 표현하는 직선이 되므로 w, b 를 바꿔가며 오차를 줄여나가는 것이 목표!

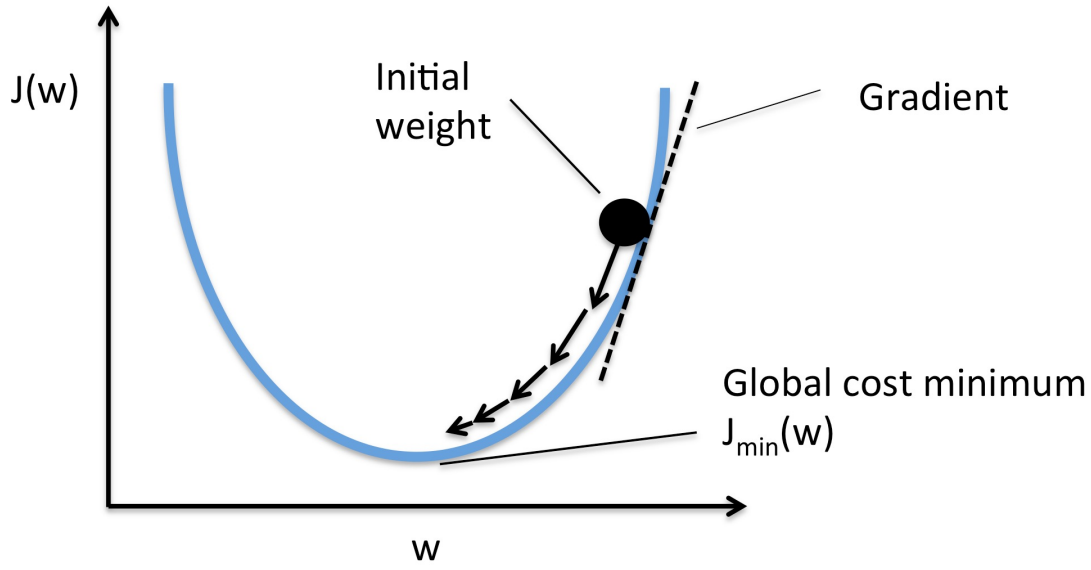
- w, b 를 찾는 방법은..? 무작위로..?
  - 무작위 방법은 모든 범위의 w, b 쌍을 계산해야 하므로 매우 비효율적...
  - 경사하강법 (gradient descent)을 사용하여 효율적으로 구함

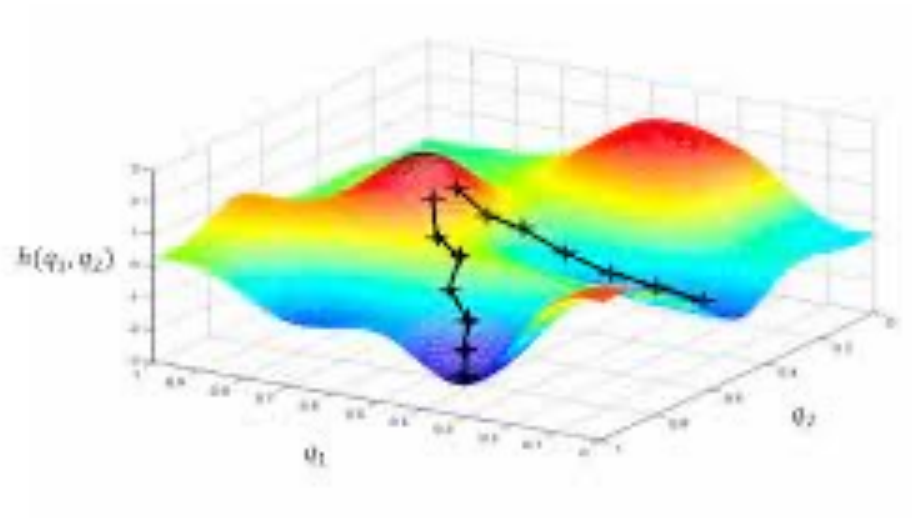
# 선형회귀분석

- 경사하강법 (gradient descent) $w_{t+1} = w_t - gradient \times learning\ rate$

: 현재 w에서 경사를 구하고 이것을 이용하여 지속적으로 w를 업데이트하는 방법

→ 손실함수(loss function)의 극솟값을 구함.



# 선형회귀분석 – PyTorch

PyTorch 에서는 텐서(Tensor)를 데이터의 기본단위로 사용

```
import troch
X = torch.Tensor(5,2)
```

5x2 형태의 tensor 생성 (원소 : 임의의 난수)

```
import troch
X = torch.tensor([1,2,3],[4,5,6])
```

2x3 형태의 tensor 생성 (원소 : $\begin{matrix}1 & 2 & 3\\ 4 & 5 & 6\end{matrix}$)

```
X = torch. tensor(data=[1.0, 2.0], requires_grad = True)
```

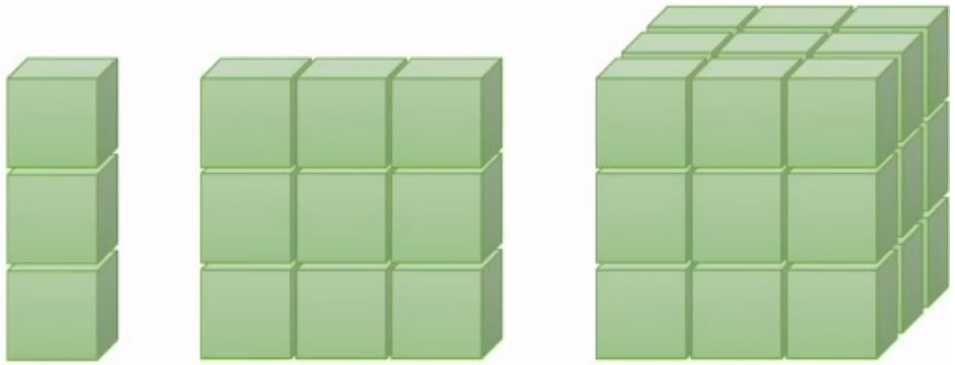
requires_grad : 텐서에 대한 기울기 저장 (기본값 False)

Ex) $z = 3x^2 + 15$ 에서 $x$에 대한 기울기 찾기

```
import torch

x = torch.tensor(data = [1.0, 2.0], requires_grad = True)
y = x**2 + 5
z = 3*y

target = torch.tensor([2.0,3.0])  목표값
loss = torch.sum(torch.abs(z-target))  z와 target 차이의 절대값 계산
loss.backward()  X에 대한 기울기 계산
```



벡터 행렬 텐서

```
print(x.grad, y.grad, z.grad)
```

```
tensor([ 6., 12.]) None None
```

**x에 대한 기울기만 계산됨.**

# 선형회귀분석 – PyTorch

- 선형회귀분석 모델 생성 및 w,b 업데이트

```
data = 500
epoch = 200

x = init.uniform_(torch.Tensor(data, 1), -10, 10)
noise = init.normal_(torch.FloatTensor(data,1),std=1)
y = 5*x+2
y_noise = y + noise
```

텐서의 값을 -10~10 까지 균등하게 초기화, 1개의 특성을 가지는 데이터 500개 생성

현실성을 주기 위해 데이터에 노이즈 추가, 이때 노이즈는 데이터와 같은 형태의 텐서로 설정해야 함

```
model = nn.Linear(1,1)
loss_f = nn.L1Loss()
```

선형 회기 모델 생성 , 인수: 입력 특성, 결과 특성, 편차 사용여부 // 입력 : 특성 1개, 출력: 특성 1개 이므로 (1,1)

손실 함수로 L1 손실 사용, $L1\ loss(x,y) = \frac{1}{n}\sum|x_i - y_i|$

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

최적화 함수 (optimization function), 경사하강법을 적용하여 오차를 줄임

*SGD : 한번에 들어오는 데이터의 수대로 경사하강법 알고리즘을 적용하는 최적화 함수
Model.parameters() : 선형회기 모델의 변수 w와 b를 전달
lr : 학습률

# 선형회귀분석 – PyTorch

```
label = y_noise
for i in range (epoch):
  optimizer.zero_grad()
  output = model(x)

  loss = loss_f(output, label)
  loss.backward()
  optimizer.step()

  para = list(model.parameters())
  print(para[0].item(), para[1].item())
```

경사하강법을 사용한 최적화 과정 epoch 수만큼 반복

이전의 기울기를 0으로 초기화 (새로운 w, b에 대해 기울기를 구하기 위해)

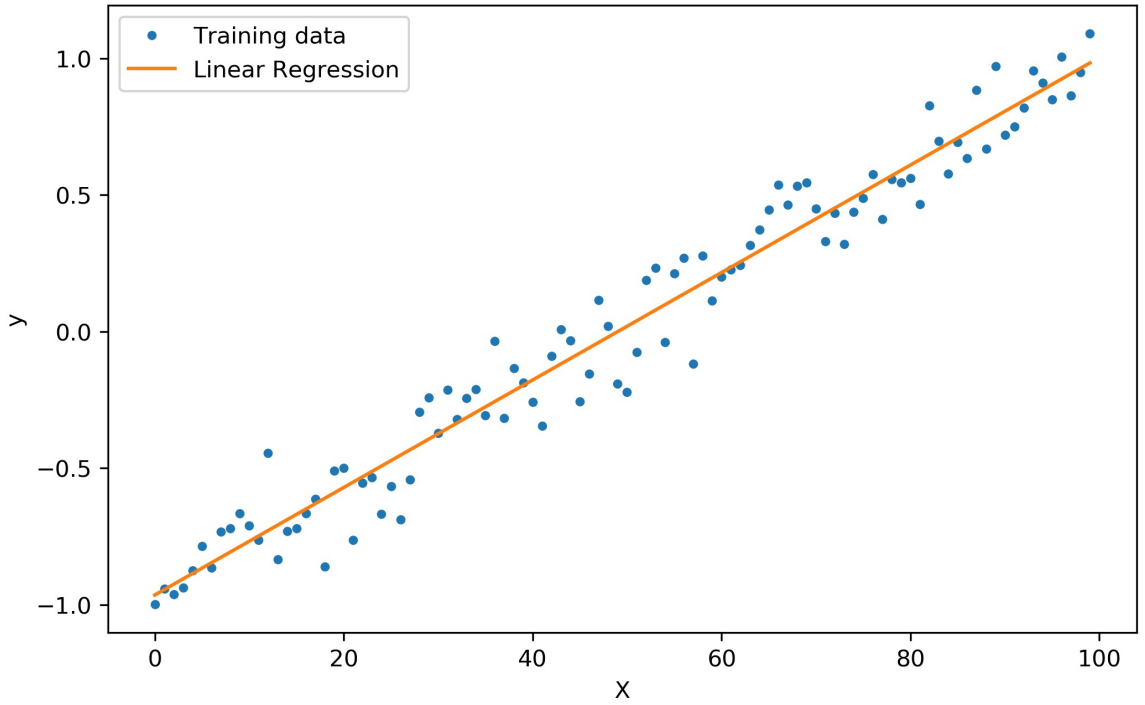
선형회기 모델에 x를 전달하고 결과를 output에 저장

loss = output과 label(y_noise)의 차이(손실) 저장

w, b 에 대한 기울기 계산

w, b 에 대한 기울기 업데이트 (학습률 : 0.01)

선형모형 학습 결과

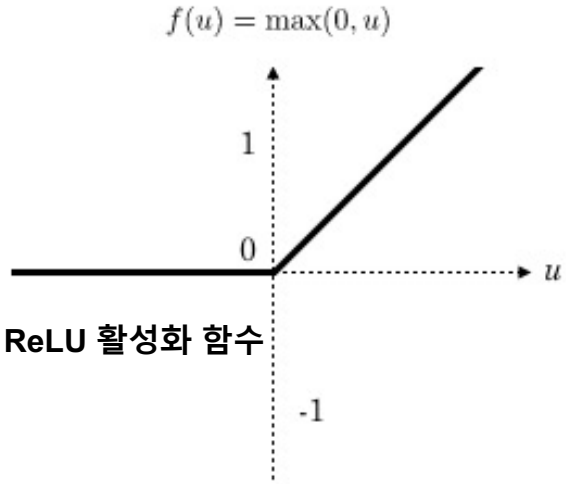


# 인공신경망

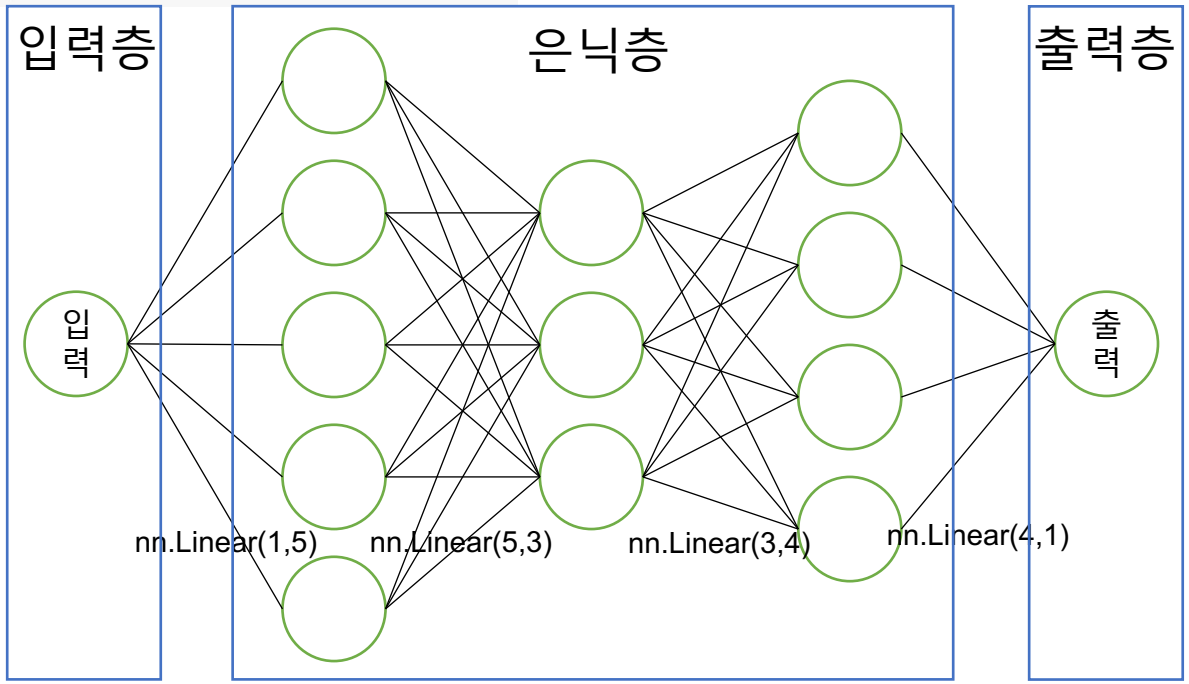
```
model = nn.Sequential(
    nn.Linear(1,5),
    nn.ReLU(),
    nn.Linear(5,3),
    nn.ReLU(),
    nn.Linear(3,4),
    nn.ReLU(),
    nn.Linear(4,1)
)
```

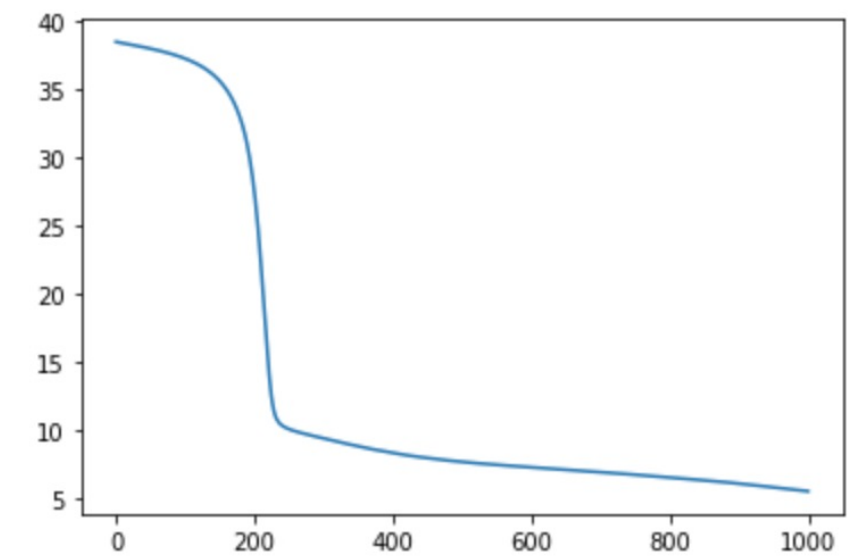
**인수로 받은 모듈을 순서대로 실행하여 결과 리턴**

**렐루 활성화 함수** $y(x) = max(0, x)$ 활성화 함수 : 신경망에 비선형성을 줌.



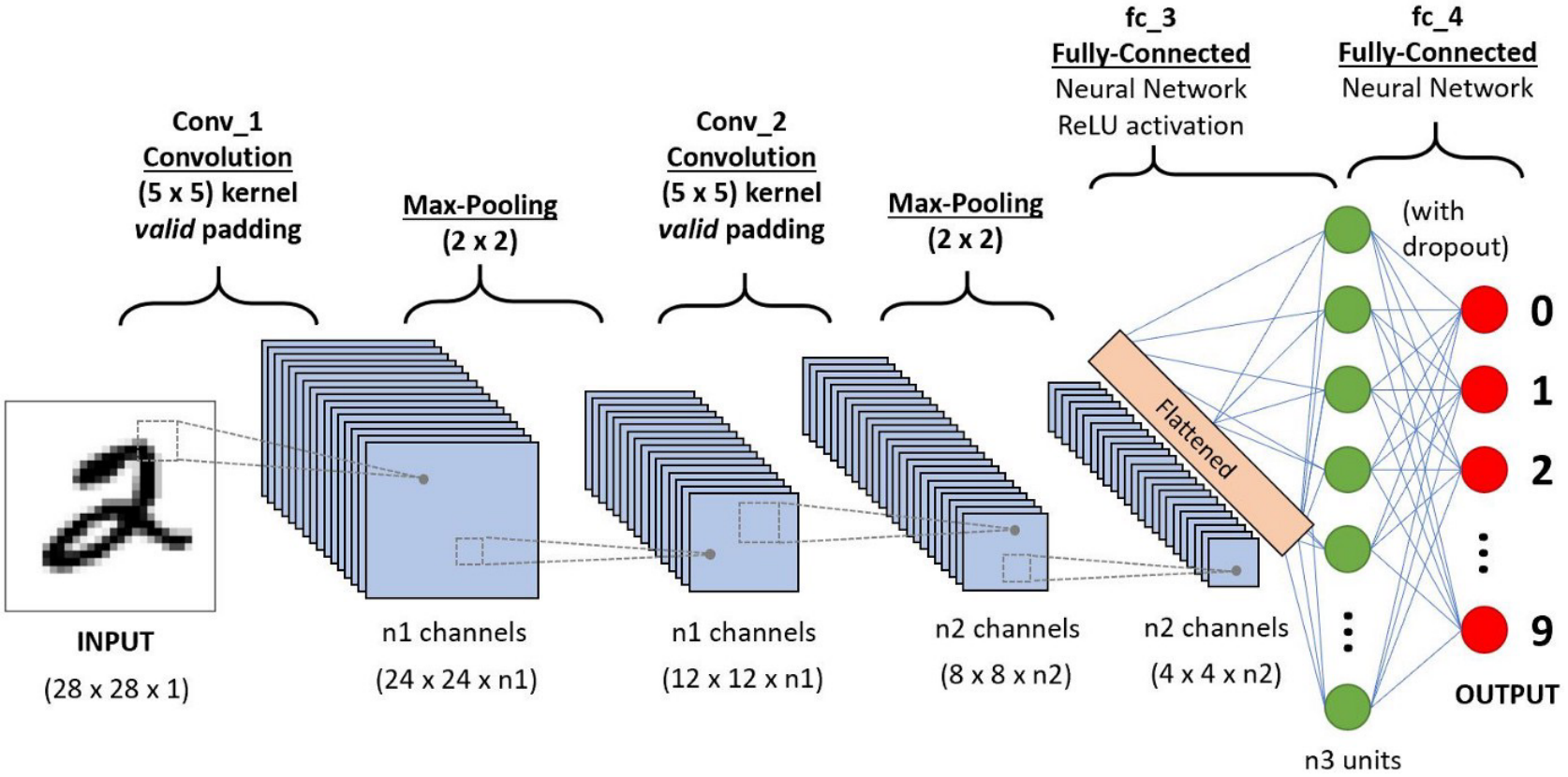

**ReLU 활성화 함수**



# 합성곱 신경망(convolutional neural network, CNN)

- CNN 동작과정

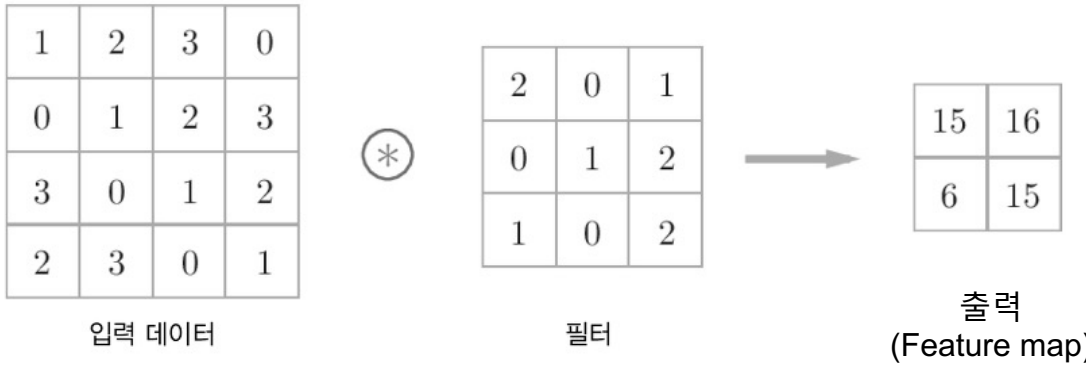



입력 → 합성곱 연산(convolution) → 활성화 함수 → 풀링(pooling) → 완전 연결 레이어(fully connected layer)

반복

# 합성곱 신경망(convolutional neural network, CNN)

- 합성곱 연산 (convolution)



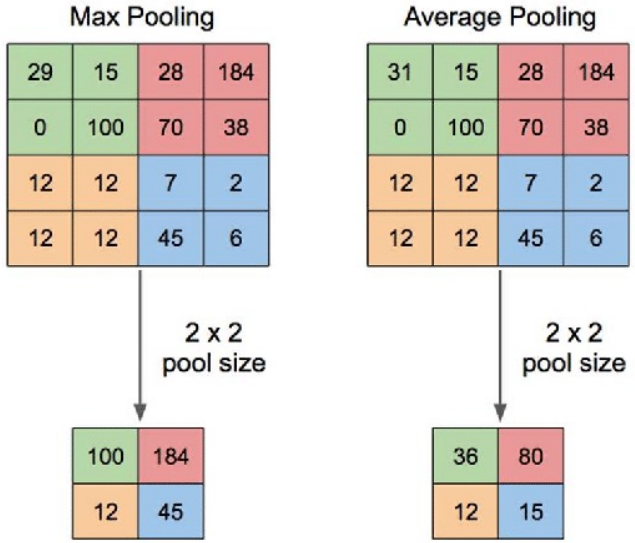

- 패딩 (padding)

  필터를 적용할수록 크기가 작아지기 때문에 패딩을 통해 특징을 충분히 뽑을 수 있도록 함.

- 풀링 (pooling)

  크기가 커서 연산량이 많을때 이미지 축소

  1. 맥스풀링(MAX pooling) : 가장 강한 자극만 남기고 나머지 무시
  2. 평균풀링(average pooling) : 일정 구간 내의 값들의 평균 전달



# 합성곱 신경망(convolutional neural network, CNN)

- CNN 모델

[batch_size , in_channels, 가로 세로]

입력 :[batch_size , 1, 28, 28]

```
class CNN(nn.Module):
  def __init__(self):
      super(CNN,self).__init__()
      self.layer = nn.Sequential(
          nn.Conv2d(1,16,5),
          nn.ReLU(),
          nn.Conv2d(16,32,5),
          nn.ReLU()
          nn.MaxPool2d(2,2),
          nn.Conv2d(32,64,5),
          nn.ReLU(),
          nn.MaxPool2d(2,2)
      )

      self.fc_layer = nn.Sequential(
          nn.Linear(64*3*3,100),
          nn.ReLU()
          nn.Linear(100,10)
      )

      def forword(self,x):
        out = self.layer(x)
        out = out.view(batch_size, -1)
        out = self.fc_layer(out)
        return out
```

[batch_size , 16, 24, 24]

nn.Conv2d(1,16,5), 합성곱 연산, out_channels = 16, kernel_size = 5 // 기본값: stride=1, padding=0

nn.ReLU(), 렐루 활성화 함수 $y(x) = max(0, x)$

[batch_size , 32, 20, 20]

[batch_size , 32, 10, 10]

nn.MaxPool2d(2,2), 풀링 연산, kernel_size=2, stride = 2 // 2x2 영역에서 풀링 후 2만큼 이동, 텐서가 반으로 줆

[batch_size , 64, 6, 6]

[batch_size , 64, 3, 3]

인공신경망, 출력을 10개의 카테고리로 줄임

목표하는 형태로 만들어 줌

목표로 하는 형태 [batch_size , -1 ] // -1 : -1인 부분 알아서 계산하라는 의미

이미지 크기 계산

$$O = floor\left(\frac{I - K + 2P}{S} + 1\right)$$

I : 입력메시지 크기
K : 커널 크기
P : 패딩 크기
S : Stride 크기

# Q & A