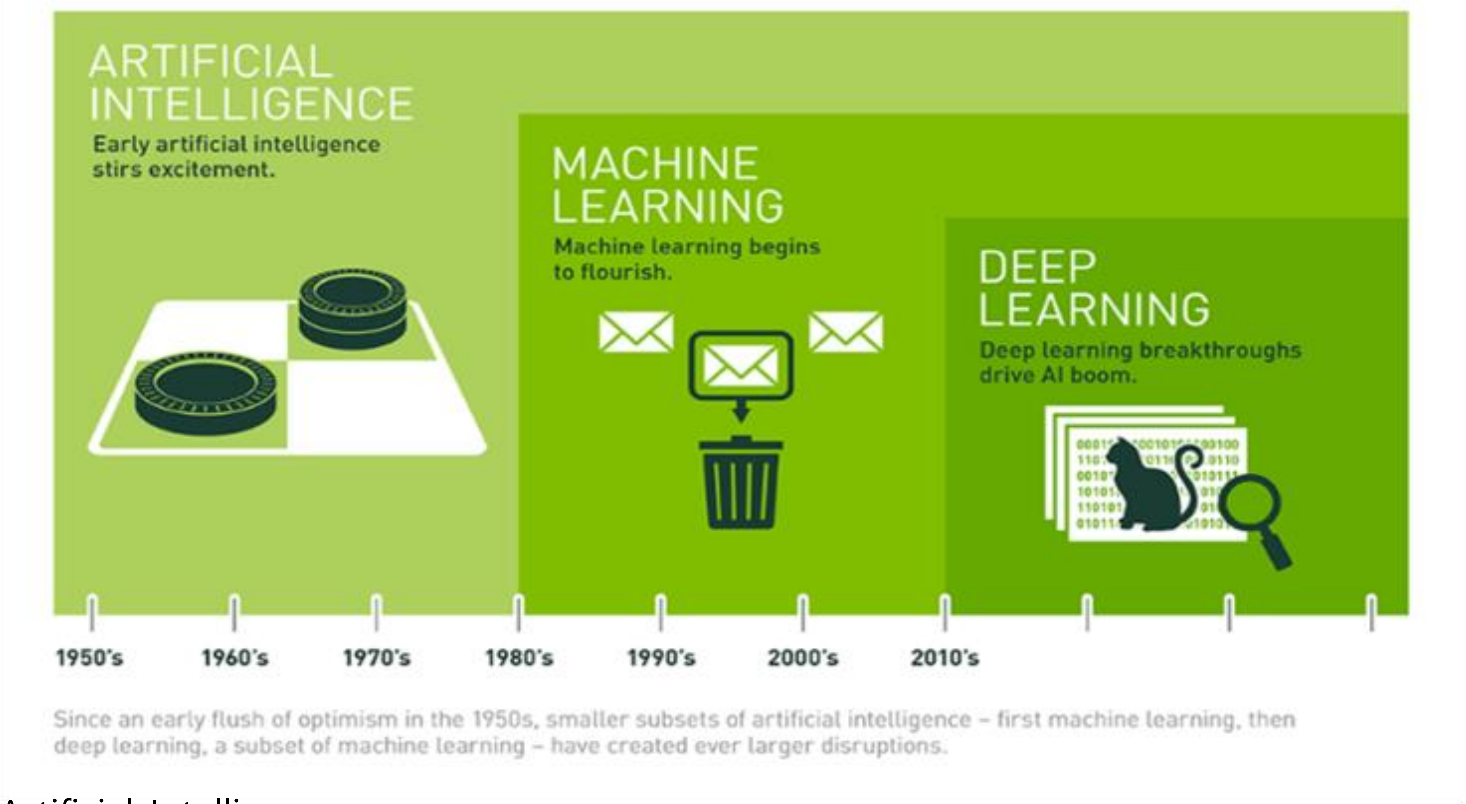
Deep Learning with Pytorch

CNN

강진규

CONTENTS

1. Artificial Intelligence, Machine Learning, Deep Learning



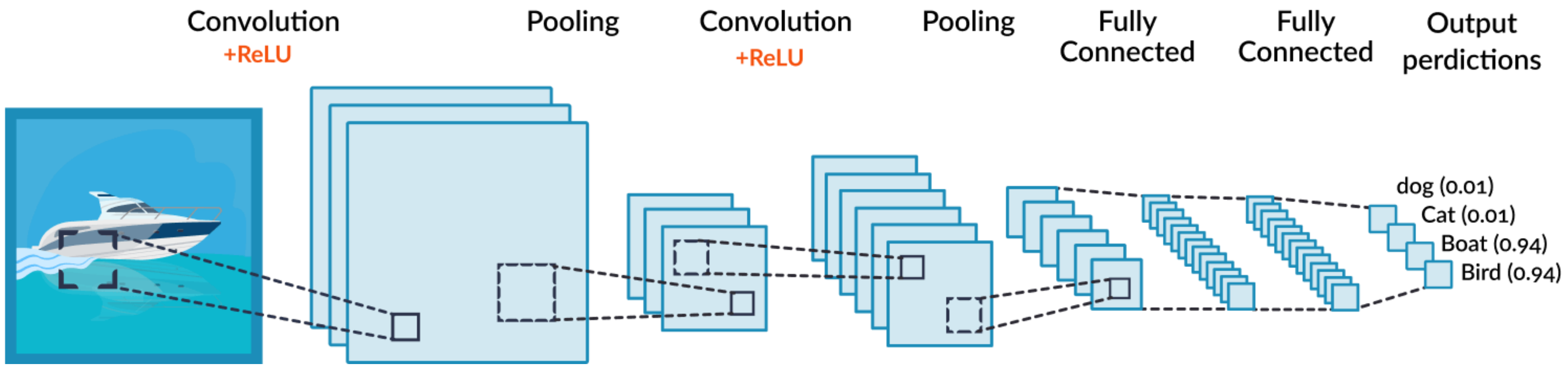
- Artificial Intelligence
  - ✓ 인간의 사고를 모방하는 모든 방법론

- Machine Learning
  - ✓ 인간이 지식을 얻는 과정인 학습을 이용한 접근 방식
  - ✓ Decision Tree, Bayesian Network, K-NN , Support Vector Machine (SVM), etc.

- Deep Learning
  - ✓ Artificial Neural Network (ANN)에서 발전한 형태로, 현재 가장 주목받는 학습법
  - ✓ Convolutional Neural Network (CNN), Generative Adversarial Network(GAN), etc.

2. Convolutional Neural Network

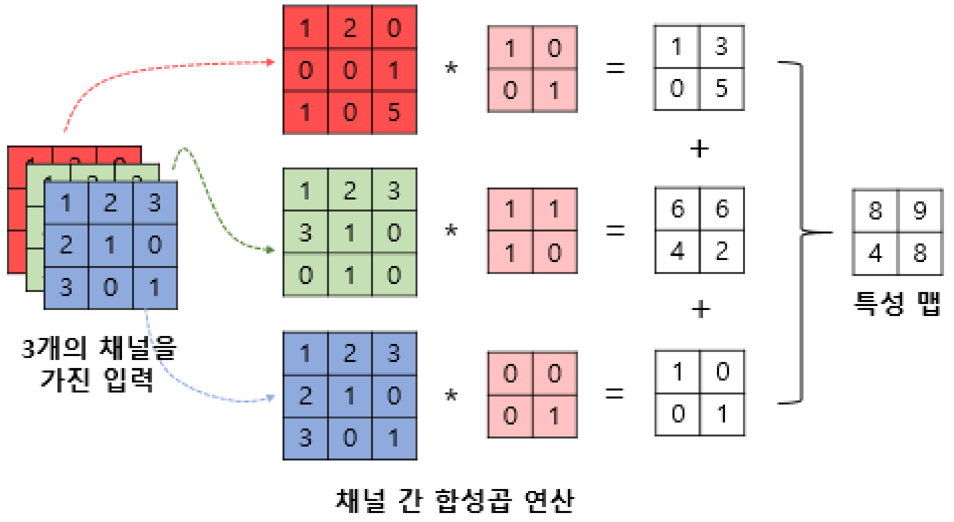
◆ Convolutional Neural Network 기본 구조 예시



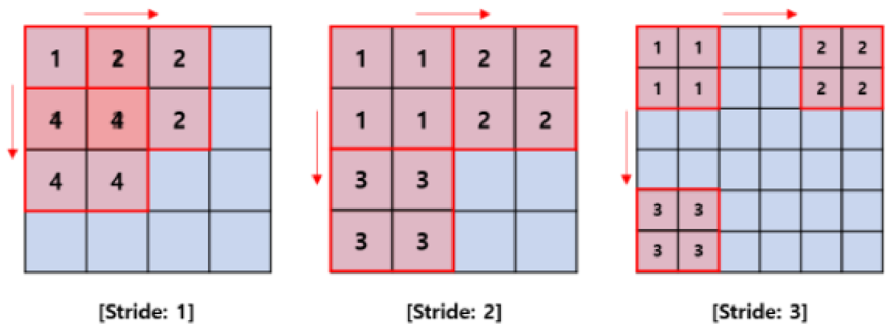
- ✓ Convolutional Layer
- ✓ Activation Function
- ✓ Pooling Layer
- ✓ Fully-Connected layer
- ✓ Softmax
- ✓ Loss Function
- ✓ Weight Initialization
- ✓ Optimization

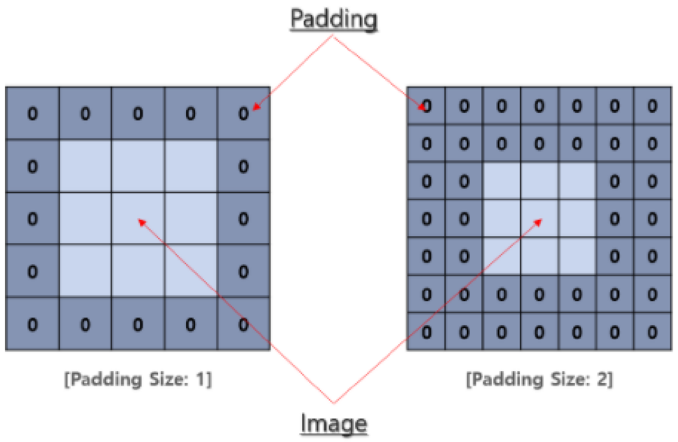
2. Convolutional Neural Network

- Convolutional Layer
  - ✓ Convolution 연산



  - ✓ Stride & Padding





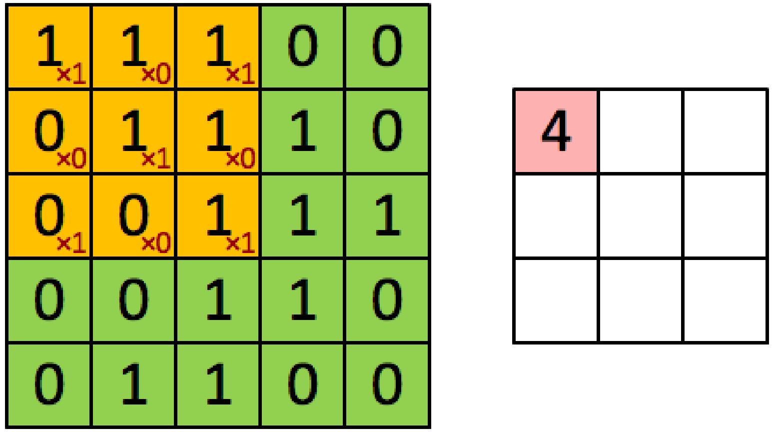
Stride: convolution연산을 수행하는 kernel이 움직이는 간격
Padding: Image의 외곽 영역에 대한 데이터 손실을 막기위해 Image 주변에 일정한 값을 채워 넣는 방법(ex. zero-padding, same padding, etc.)

2. Convolutional Neural Network

◆ Convolutional Layer

✓ 예시 1

Input = 5 x 5
Kernel = 3 x 3
Padding = 0
Stride = 1
Output = 3 x 3

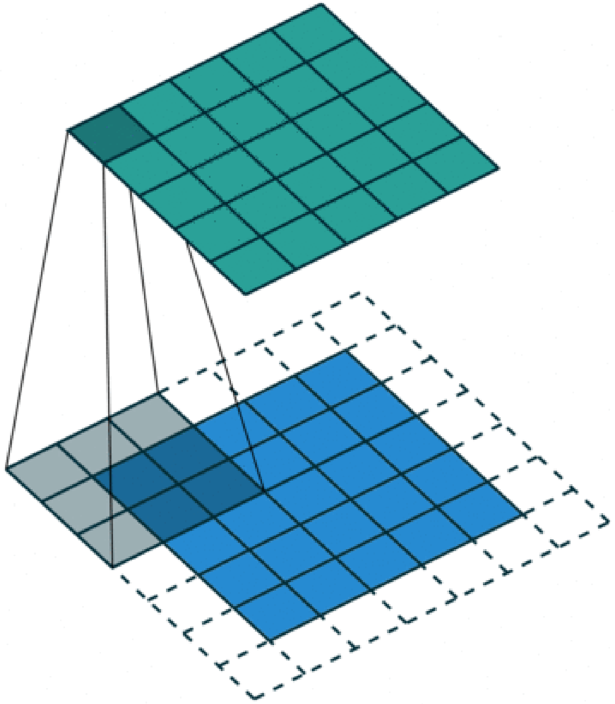


Image

Convolved Feature

✓ 예시 2

Input = 5 x 5
Kernel = 3 x 3
Padding = 1
Stride = 1
Output = 5 x 5



✓ Feature Map(output) 계산

I_h : 입력 데이터의 높이
I_w : 입력 데이터의 너비
K_h : 커널의 높이
K_w : 커널의 너비
S : stride
P : padding
O_h : feature map 의 높이
O_w : feature map 의 크기

$$(O_h, O_w) = \left(\left\lfloor \frac{I_h - K_h + 2P}{S} + 1 \right\rfloor, \left\lfloor \frac{I_w - K_w + 2P}{S} + 1 \right\rfloor\right)$$

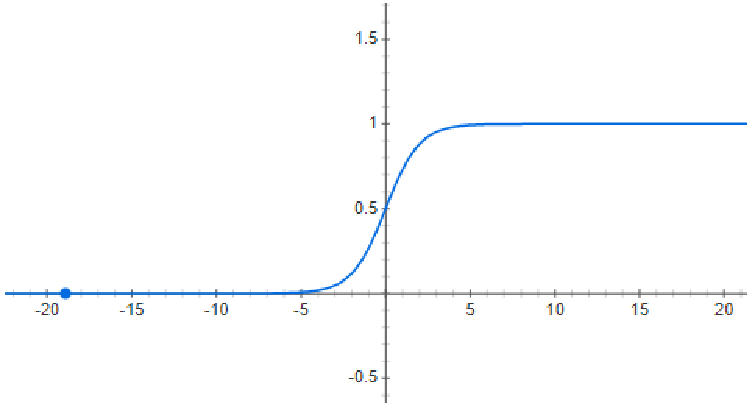
2. Convolutional Neural Network

◆ Activation Function

✓ Non-Linearity: Deep Neural Network(DNN)에 가치 부여

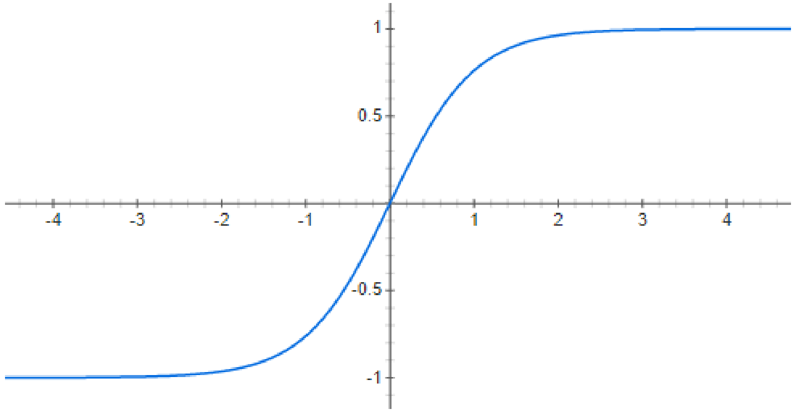
✓ Sigmoid Function

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$



✓ Hyperbolic tangent

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



- X가 0에서 증가하거나 감소하는 방향으로 gradient 값이 빠르게 0으로 수렴

  ➡ Back propagation (weight update)과정에서 업데이트가 잘 되지 않는 Saturation현상 (Vanishing Gradient) 발생

- Sigmoid Function은 output과 gradient 값이 항상 0보다 크거나 같으므로 Zigzag현상 발생

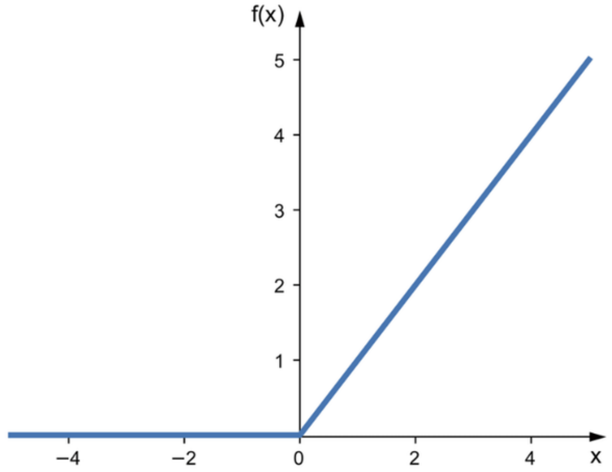
  ➡ 업데이트 속도가 느려짐

2. Convolutional Neural Network

Activation Function

✓ ReLU (Rectified Linear Unit)

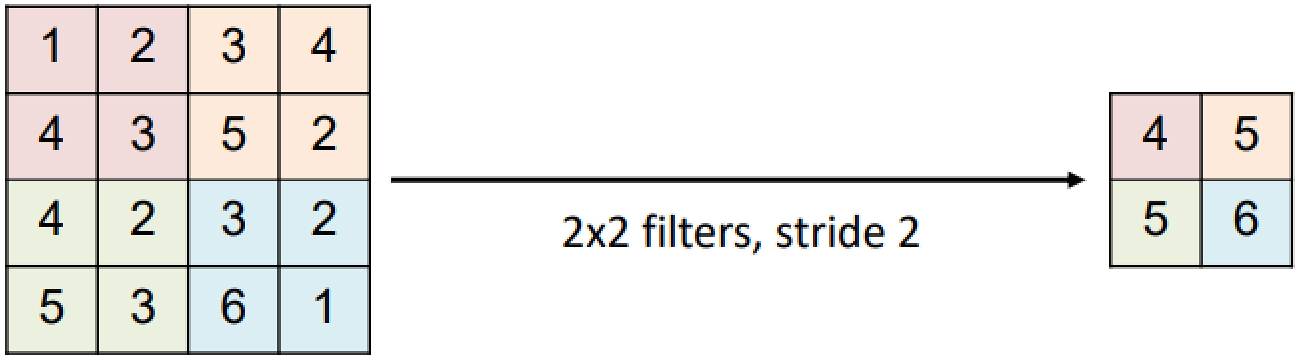
$$f(x) = \max(0, x)$$



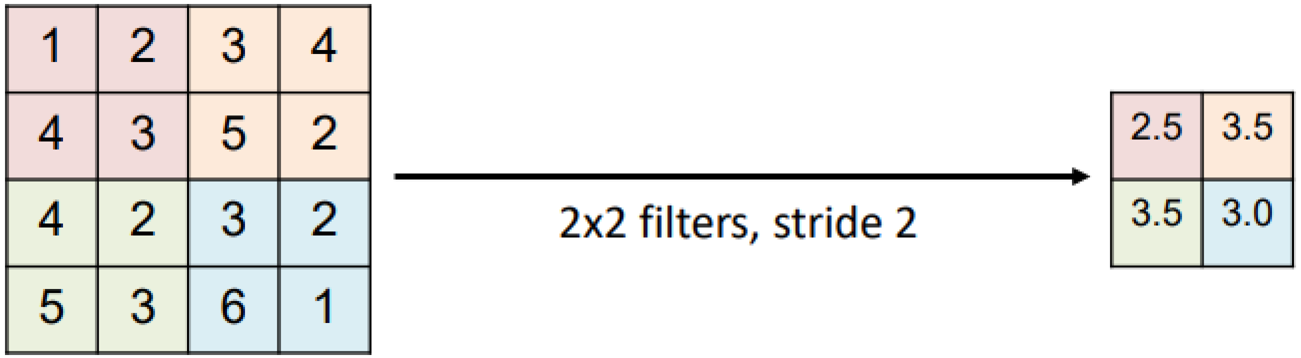
- Saturation 현상을 절반으로 감소
- 계산 복잡도가 낮음 (exponential function x)
- ReLU의 output과 gradient값이 항상 0보다 크거나 같으므로, zigzag현상 발생
- ReLU 이외에 Leacky ReLU, Parametric ReLU, ELU, SELU 등 다양한 Activation Function에 대한 연구

2. Convolutional Neural Network

- **Pooling Layer**
  - ✓ Down sampling을 통해 feature map의 크기를 줄여주는 연산을 수행
  - ✓ Max pooling

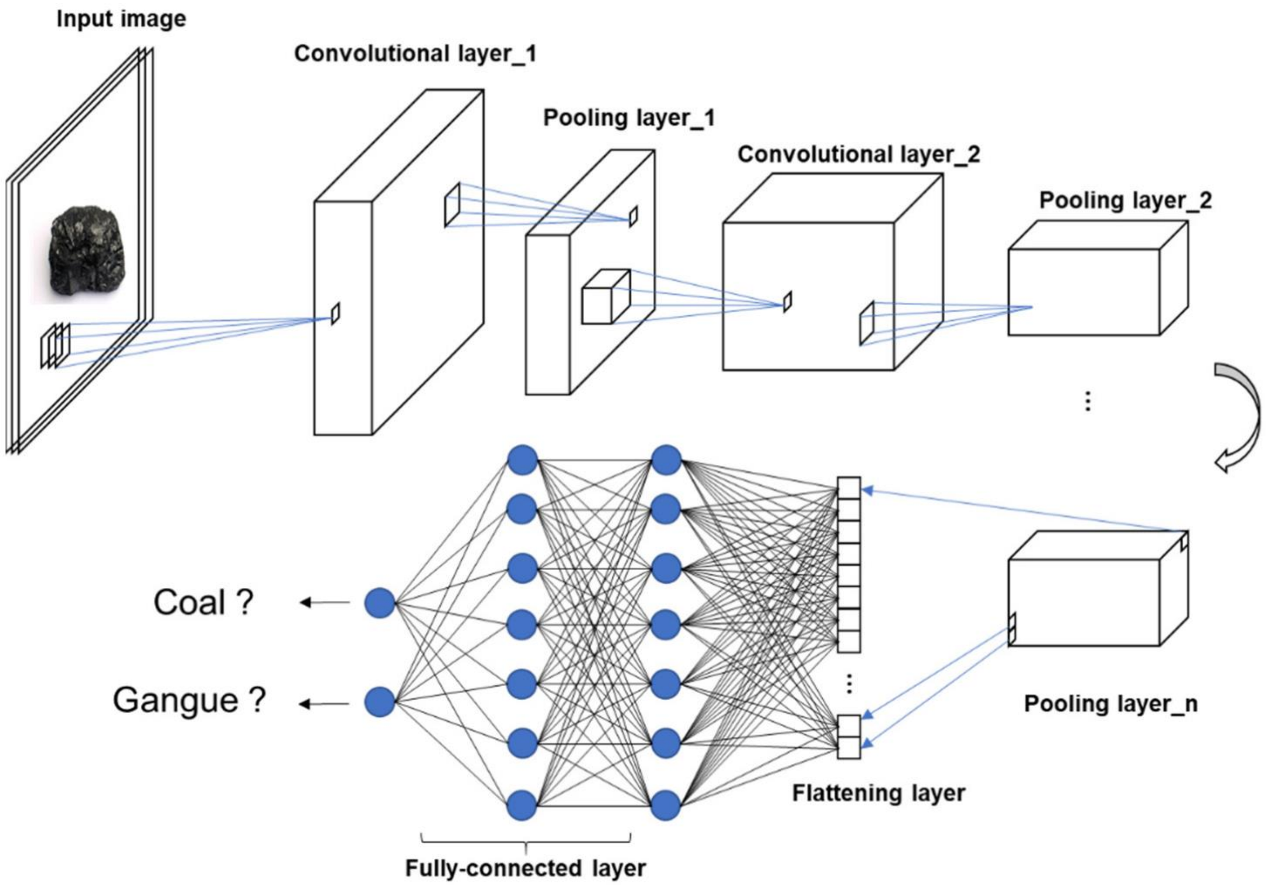


  - ✓ Average pooling



2. Convolutional Neural Network

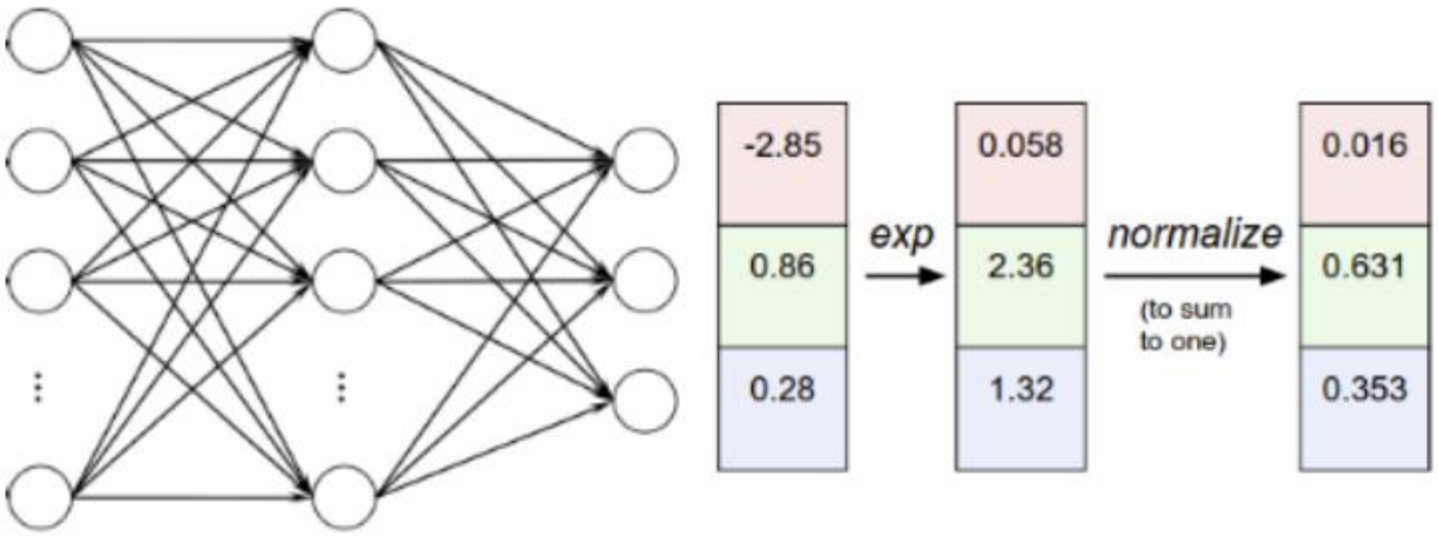
- Fully-Connected Layer
  - ✓ 전체 입력 node가 각 출력 node와 연결된 layer
  - ✓ Spatial Feature 손실 발생
  - ✓ 입력 데이터의 크기 변형에 취약



2. Convolutional Neural Network

- Softmax
  - ✓ 주로 마지막 layer에서 확률분포를 얻기 위해 사용

$$P(y=j\mid x)=\frac{e^{w_j x}}{\sum_{k=1}^{K} e^{w_k x}}$$



2. Convolutional Neural Network

- Loss Function
  - ✓ 학습의 목적을 의미하는 함수로, Loss 를 최소화 하는 방향으로 학습
  - ✓ MSE (Mean Squared Error)

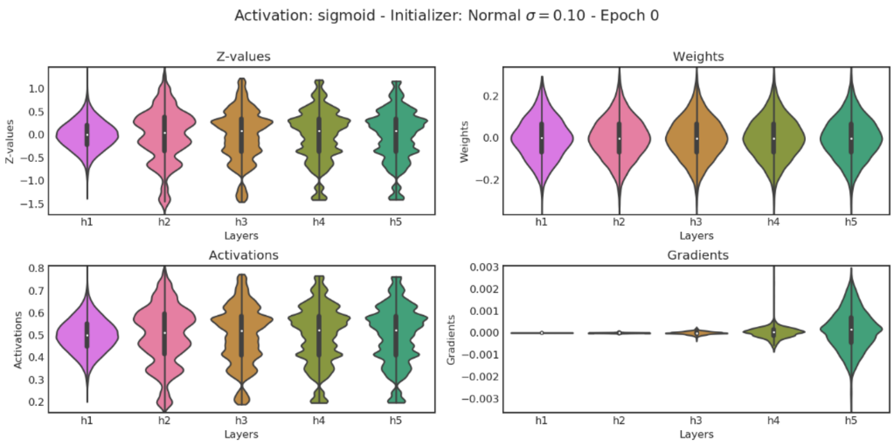
$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

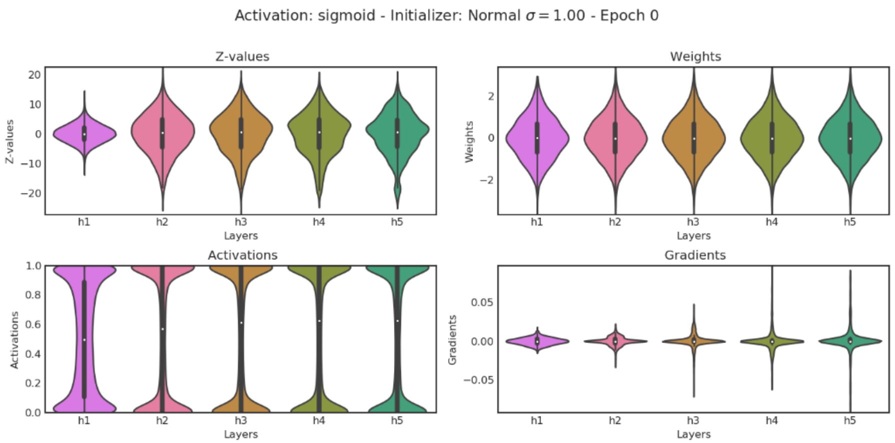
  - ✓ Cross Entropy
    - 두 개의 확률분포 p 와 q 에 대해 하나의 사건 X 가 갖는 정보량
    - $H_{p,q}(X) = -\sum_{i=1}^{N} p(x_i) \log q(x_i)$
    - p : True probability, q : prediction probability, N : # of classes

2. Convolutional Neural Network

◆ Weight Initialization

✓ Weight를 초기화 하는 방법에 따라, 성능이 달라짐

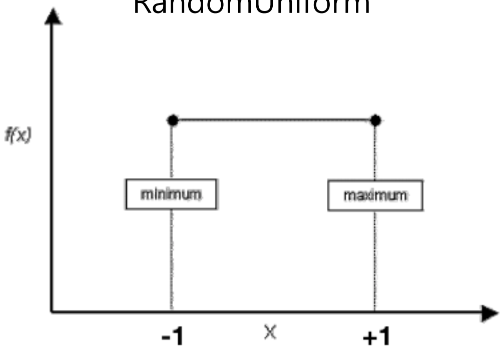




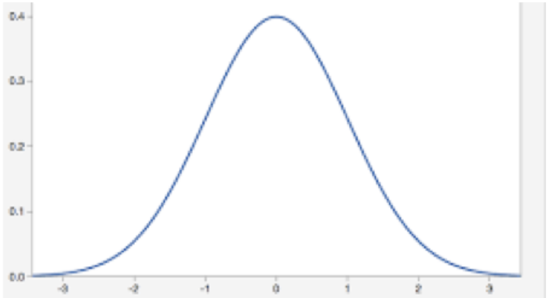
✓ 확률 분포 기반 초기화 (RandomUniform, RandomNormal, TruncatedNormal)

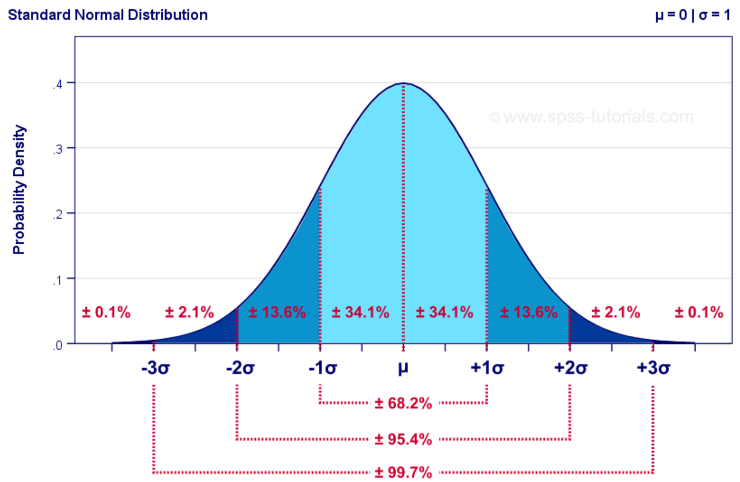
- 레이어가 깊어질수록 vanishing gradient problem 발생 가능성 증가

RandomUniform



RandomNormal





2. Convolutional Neural Network

- Weight Initialization
  - ✓ 분산 조정 기반 초기화 (**Xavier**, Lecun, He, etc.)
    - Assumption: Activation Value가 평균과 표준편차가 일정하게 weight를 초기화 하면 Layer를 계속 쌓을 수 있을 것
    - Fan in(입력 뉴런의 수)와 Fan out(출력 뉴런의 수)를 고려하여 확률 분포를 조정
    - Fan in과 Fan out의 평균은 0, 분산은 일치하도록 초기화

Gaussian

$$Var(W) = \frac{2}{n_{in} + n_{out}}$$

Uniform

$$Var(W) = \sqrt{\frac{6}{n_{in} + n_{out}}}$$

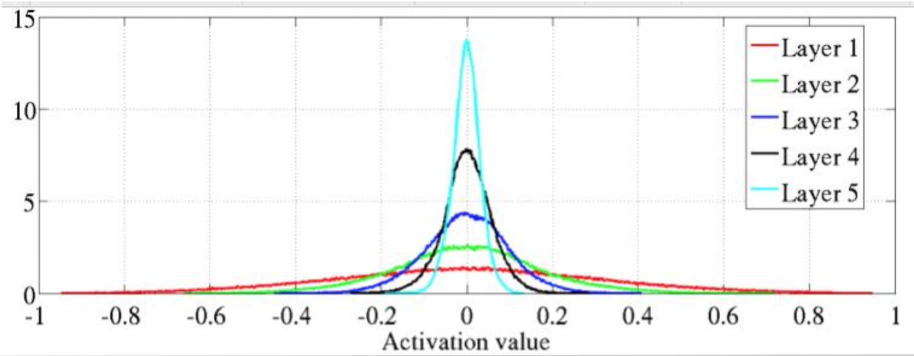
2. Convolutional Neural Network

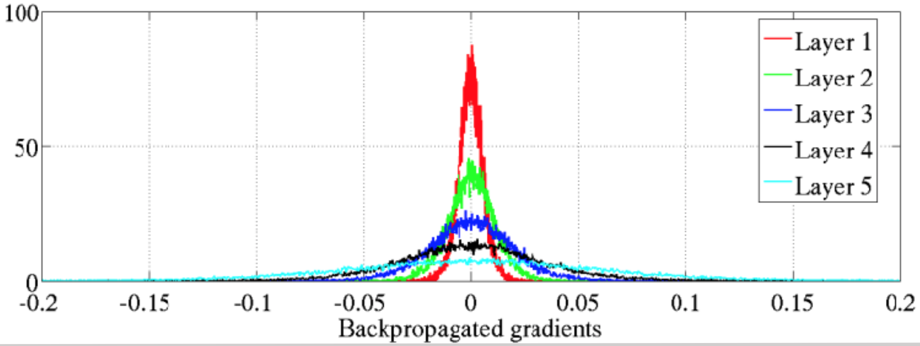
◆ Weight Initialization

- ✓ 분산 조정 기반 초기화 (**Xavier**, Lecun, He, etc.)

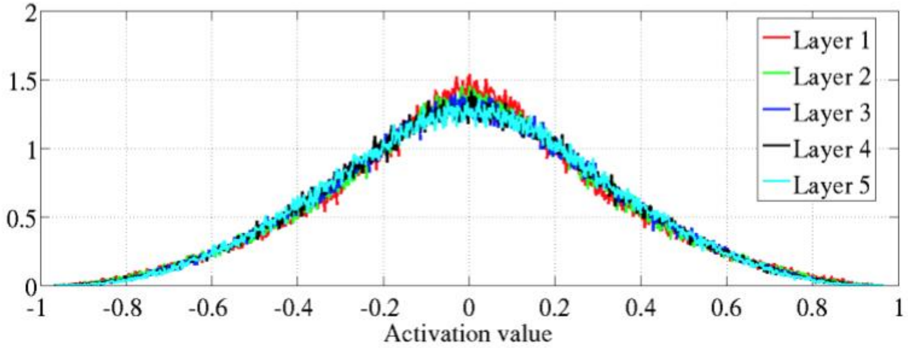
- ✓ torch.nn.init.xavier_uniform_(layer.weight)
- ✓ torch.nn.init.xavier_normal_(layer.weight)

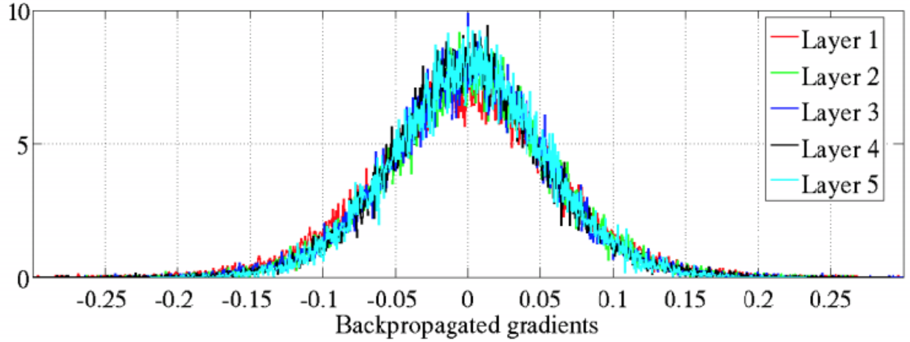
RandomNormal





Xavier





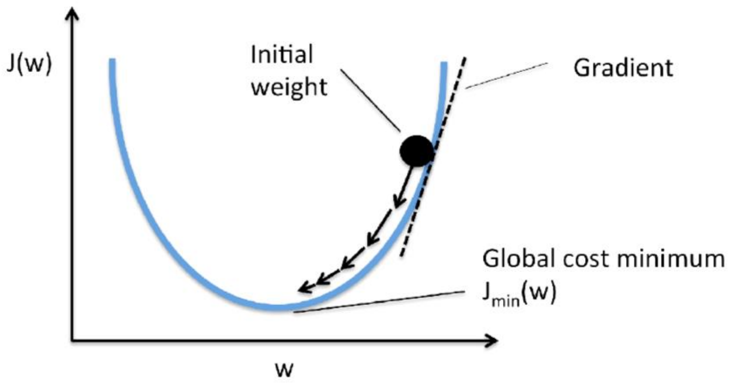
forward propagation

backpropagation

2. Convolutional Neural Network

◆ Optimization

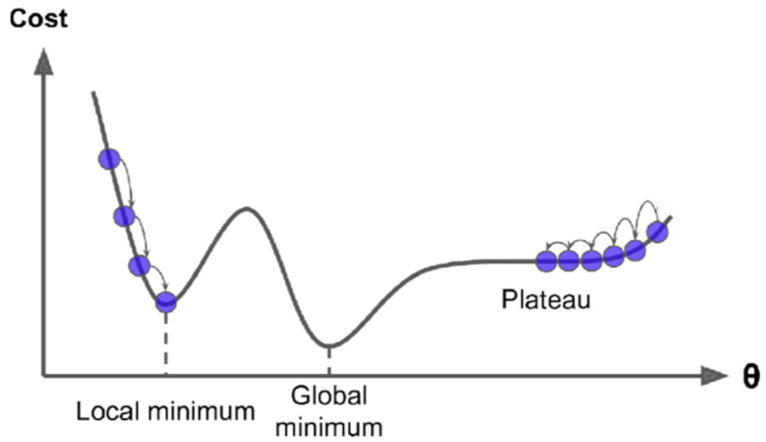
✓ Weight를 업데이트 하는 알고리즘



✓ SGD (Stochastic Gradient Decent)

- Loss function을 미분하여 weight를 업데이트하는 가장 기본적인 방법

- Local optimization 위험성, Plateau 취약성, zigzag현상, 기울기가 작은 곳에서 학습 속도 느림



$$\theta = \theta - \alpha * \frac{dJ(\theta)}{d\theta}$$

2. Convolutional Neural Network

◆ Optimization

✓ Momentum method (momentum, nesterov momentum)

- 현재 gradient와 이전 gradient를 고려하여 weight를 업데이트 하는 방법

$$vdw = \gamma * vdw + \nu * \frac{dJ(\theta)}{d\theta}$$

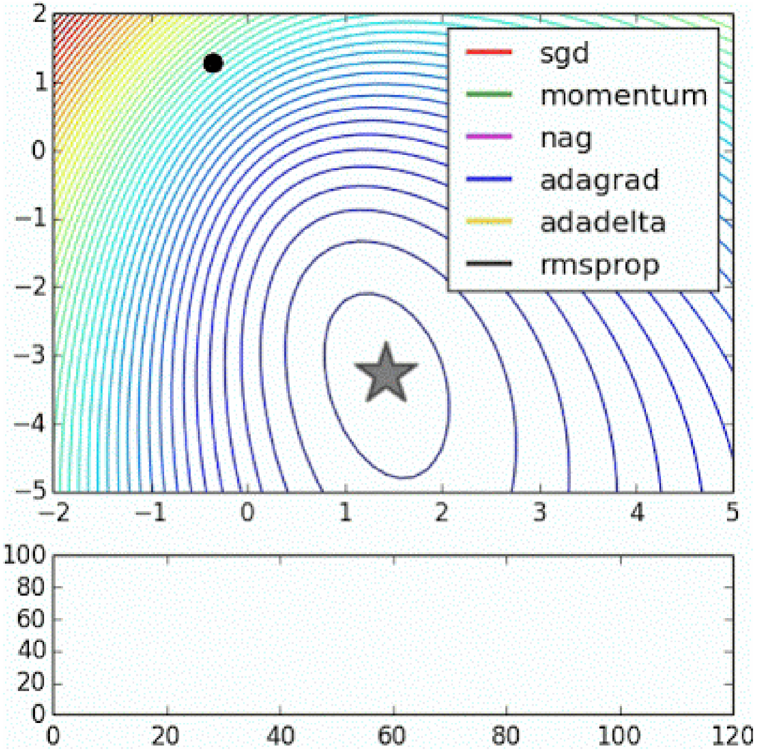
- Zigzag 현상

✓ Adaptive method

- Learning rate를 이전 gradient의 크기의 합으로 나누어주어, 학습이 진행됨에 따라 현재 gradient값을 작게 반영함으로써 효율적 학습을 진행하는 방법

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

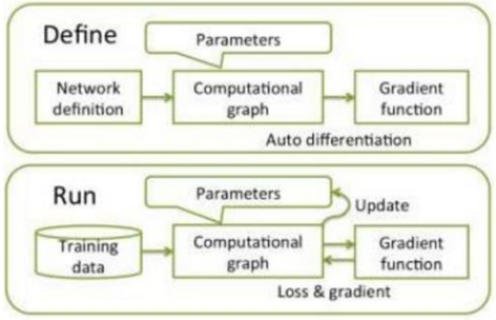
- G_t의 누적으로 학습이 진행될수록 업데이트가 느려짐



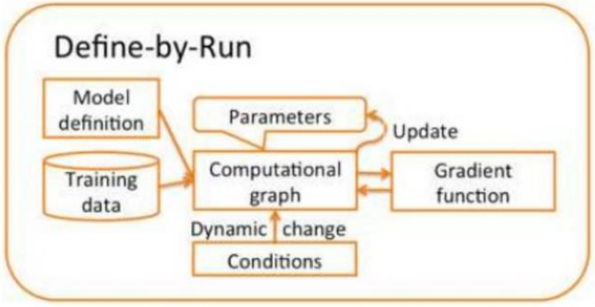
3. Pytorch

◆ Pytorch (Facebook) vs Tensorflow (Google)

	Pytorch	TensorFlow
패러다임	Define by Run	Define and Run
그래프 형태	Dynamic Graph	Static Graph



[Define-and-Run (TensorFlow)]



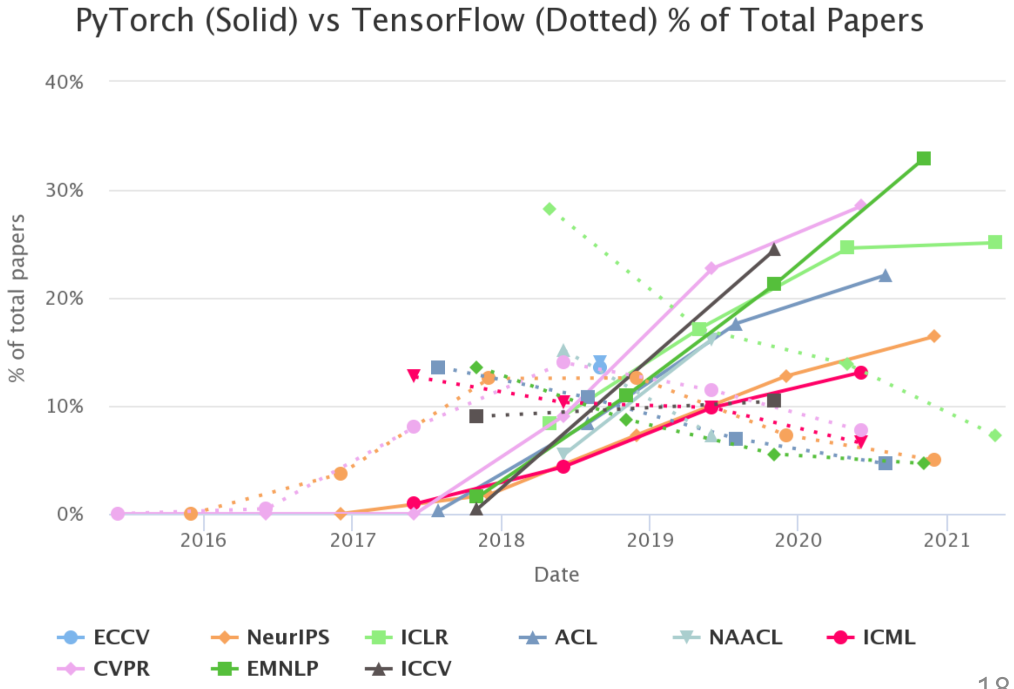
[Define-by-Run (Chainer, PyTorch)]

◆ Pytorch의 장점

✓ 코드의 간결성과 직관성

✓ API (https://pytorch.org/)

✓ 최신 이용자 및 연구 증가



3. Pytorch

◆ 개발 환경 갖추기

✓ 파이썬(ver. 3.9.1) 설치
- https://www.python.org/downloads/release/python-391/
- "Add python 3.9.1 to path" 선택
- 버전 확인: cmd에서 "python --version" 입력

✓ Pytorch(1.7.1 cpu ver) 설치
- "pip install torch==1.7.1+cpu torchvision==0.8.2+cpu torchaudio==0.7.2 –f https://download.pytorch.org/whl/torch_stable.html"

- "pip install numpy matplotlib scikit-learn"

✓ Jupyter Notebook 설치
- "pip install jupyter"
- 실행: "jupyter notebook" 입력

4. CNN 실습

◆ 실습 1-(1) MNIST 를 이용한 간단한 CNN 예제

```
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
```

```
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)
```

```
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 F.log_softmax(x,dim=1)
```

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

```
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()))
```

4. CNN 실습

◆ 실습 1-(2) MNIST 를 이용한 간단한 CNN 예제

```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()

            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
```

```python
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))
```

```
Train Epoch: 37 [0/60000 (0%)]  Loss: 0,054104
Train Epoch: 37 [12800/60000 (21%)]     Loss: 0,038025
Train Epoch: 37 [25600/60000 (43%)]     Loss: 0,080054
Train Epoch: 37 [38400/60000 (64%)]     Loss: 0,090231
Train Epoch: 37 [51200/60000 (85%)]     Loss: 0,078049
[37] Test Loss: 0,0342, Accuracy: 98,88%
Train Epoch: 38 [0/60000 (0%)]  Loss: 0,078049
Train Epoch: 38 [12800/60000 (21%)]     Loss: 0,270210
Train Epoch: 38 [25600/60000 (43%)]     Loss: 0,115710
Train Epoch: 38 [38400/60000 (64%)]     Loss: 0,152536
Train Epoch: 38 [51200/60000 (85%)]     Loss: 0,045493
[38] Test Loss: 0,0336, Accuracy: 98,90%
Train Epoch: 39 [0/60000 (0%)]  Loss: 0,259200
Train Epoch: 39 [12800/60000 (21%)]     Loss: 0,318306
Train Epoch: 39 [25600/60000 (43%)]     Loss: 0,042251
Train Epoch: 39 [38400/60000 (64%)]     Loss: 0,025996
Train Epoch: 39 [51200/60000 (85%)]     Loss: 0,136765
[39] Test Loss: 0,0350, Accuracy: 98,95%
Train Epoch: 40 [0/60000 (0%)]  Loss: 0,056306
Train Epoch: 40 [12800/60000 (21%)]     Loss: 0,055596
Train Epoch: 40 [25600/60000 (43%)]     Loss: 0,142690
Train Epoch: 40 [38400/60000 (64%)]     Loss: 0,279247
Train Epoch: 40 [51200/60000 (85%)]     Loss: 0,147692
[40] Test Loss: 0,0331, Accuracy: 99,03%
```