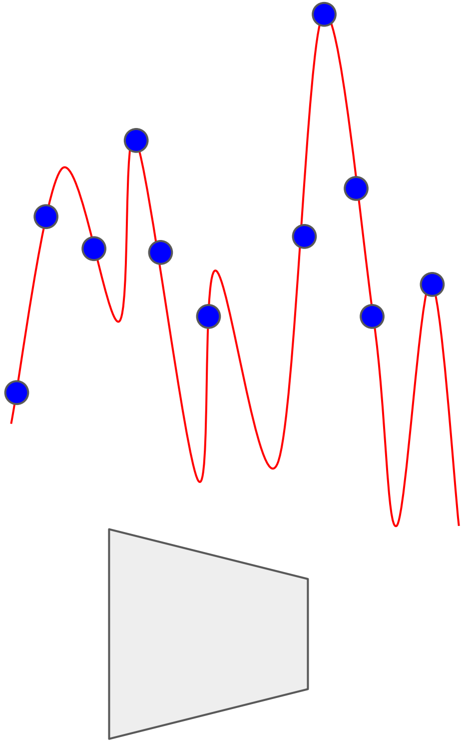
딥러닝 이해하기

leejeyeol92@gmail.com

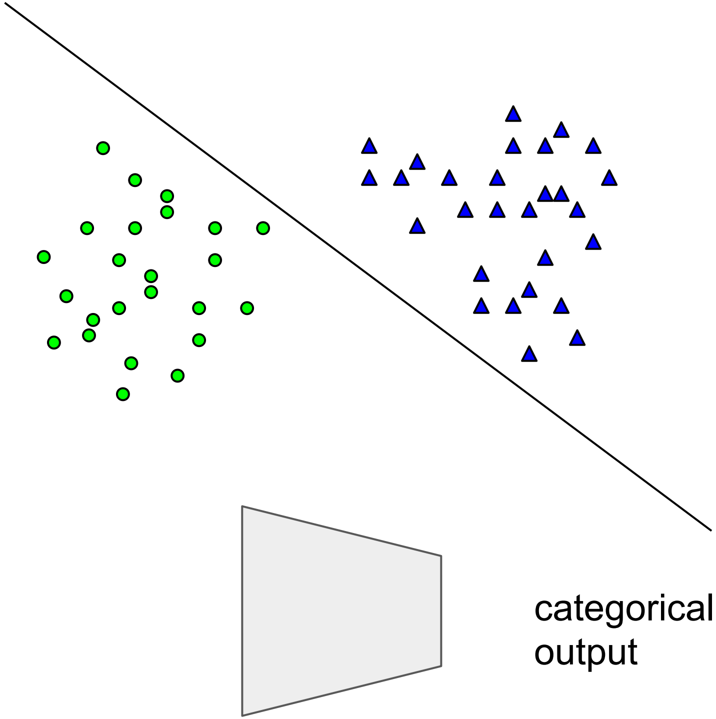
Classification

Classification Problem

regression

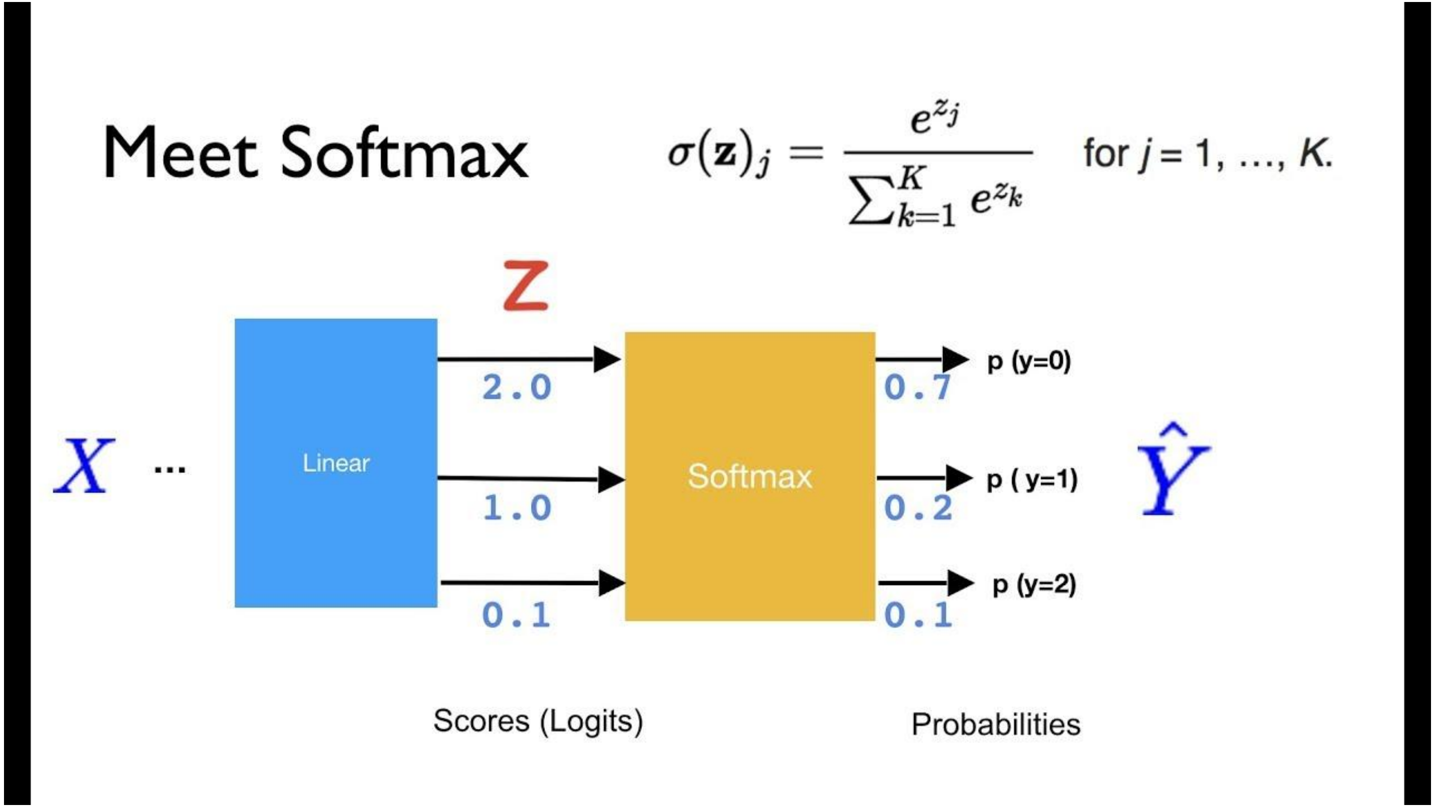


classification



categorical
output

Softmax function



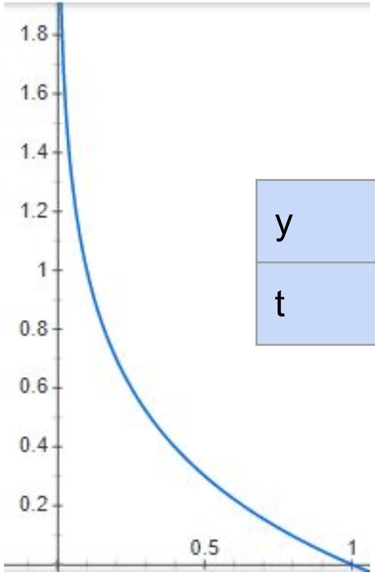
One Hot encoding

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Cross Entropy loss function

$$-\sum_k t_k \log y_k$$



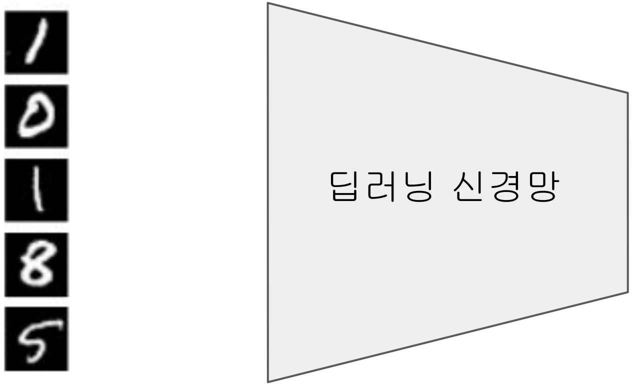
y	0.1	0.6	0.1	0.1	0.1
t	0	1	0	0	0

실습 6

MNIST Classifier 구현 ver pytorch

28x28 크기의
숫자 손글씨
이미지 데이터

입력



출력

[0,1,0,0,0,0,0,0,0,0]
[1,0,0,0,0,0,0,0,0,0]
[0,1,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,1,0]
[0,0,0,0,0,1,0,0,0,0]

실습 6

필요한 함수들

```
import torch
import torch.utils.data
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import matplotlib.pyplot as plt

def one_hot_embedding(labels, num_classes):
    # 단일 라벨 텐서를 원핫 벡터로 바꿔줍니다.
    y = torch.eye(num_classes)
    one_hot = y[labels]
    return one_hot

def softmax_to_one_hot(tensor):
    # softmax 결과를 가장 높은 값이 1이 되도록 하여 원핫 벡터로 바꿔줍니다. acuuracy 구할 때 씁니다.
    max_idx = torch.argmax(tensor, 1, keepdim=True)
    if tensor.is_cuda :
        one_hot = torch.zeros(tensor.shape).cuda()
    else:
        one_hot = torch.zeros(tensor.shape)
    one_hot.scatter_(1, max_idx, 1)
    return one_hot

def weight_init(m):
    classname = m.__class__.__name__
    # m에서 classname이 Linear(신경망 레이어)인 경우
    if classname.find('Linear') != -1:
        # weight를 uniform distribution을 이용하여 초기화하고 bias는 0으로 초기화
        m.weight.data.uniform_(0.0, 1.0)
        m.bias.data.fill_(0)
```

실습 6

신경망 모델

train, test용 데이터로더

```
class TwoLayerNet_pytorch(nn.Module):
  def __init__(self, input_size, hidden_size, output_size):
    super().__init__()
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.output_size = output_size

    self.network1 = nn.Sequential(
        nn.Linear(self.input_size, self.hidden_size),
        nn.Sigmoid(),
        nn.Linear(self.hidden_size, self.output_size),
        nn.Softmax()
    )
  def forward(self, x):
    y = self.network1(x)
    return y

epochs = 5
learning_rate = 0.01
batch_size = 100
loss_function = nn.BCELoss()

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor()
                       #,transforms.Normalize((0.1307,), (0.3081,)) # 한번 돌려본 후, 돌아가는것을 확인했다면 이 주석을 지우세요.
                   ])),
    batch_size=batch_size, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor()
                       #,transforms.Normalize((0.1307,), (0.3081,)) # 한번 돌려본 후, 돌아가는것을 확인했다면 이 주석을 지우세요.
                   ])),
    batch_size=batch_size, shuffle=True)
```

실습 6

학습

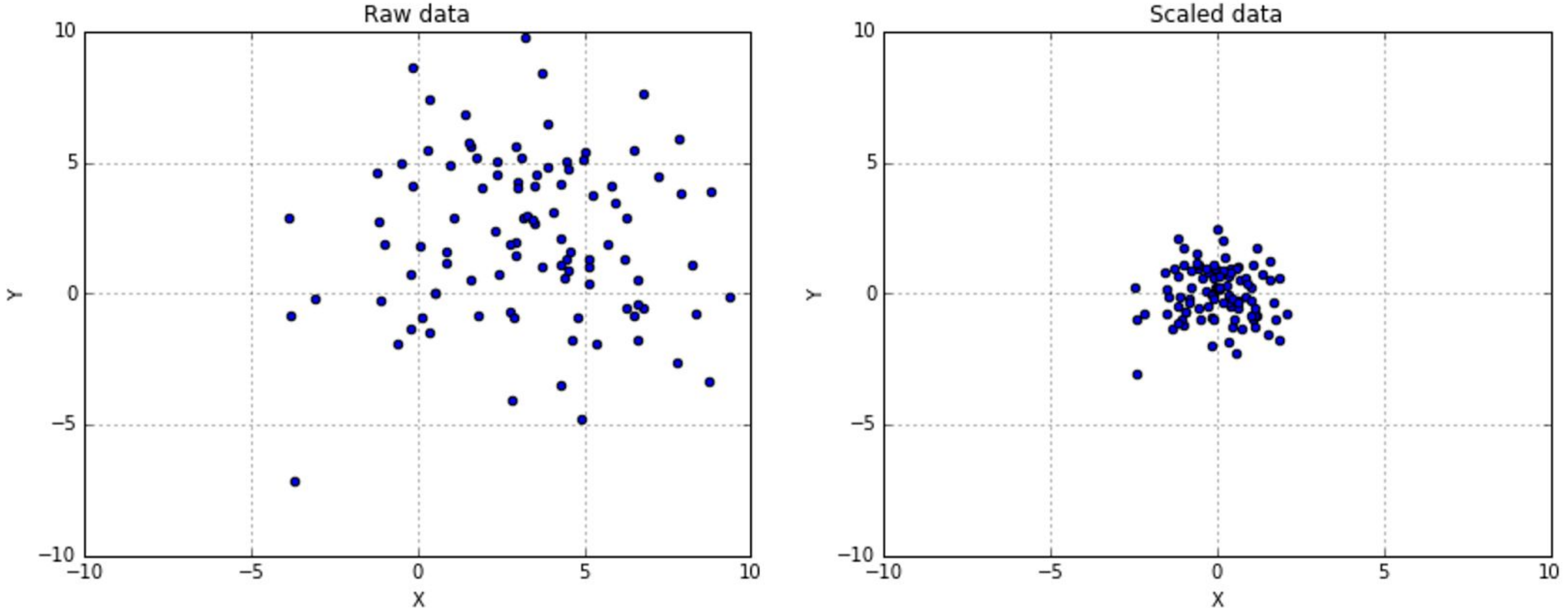
```
net = TwoLayerNet_pytorch(input_size=784, hidden_size=50, output_size=10)
net.apply(weight_init)

optimizer = optim.SGD(net.parameters(), lr=learning_rate)

train_loss_list = [] # 결과 출력을 위한 코드
net.train() # 학습할것을 명시하여 자원낭비를 줄이는 코드
for epoch in range(epochs):
    for i, (X, t) in enumerate(train_loader):
        X = X.view(-1, 784) # 1 x 28 x 28 형태임으로, 784 형태의 벡터로 바꿔준다.
        t = one_hot_embedding(t, 10) # 숫자로 출력됨으로 원핫코드로 바꿔준다.

        # 순전파
        Y = net(X)
        loss = loss_function(Y, t)

        train_loss_list.append(loss) # 결과 출력을 위한 코드
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        print("[%d/%d][%d/%d] loss : %f"%(i,len(train_loader),epoch,epochs, loss))
```

실습 6

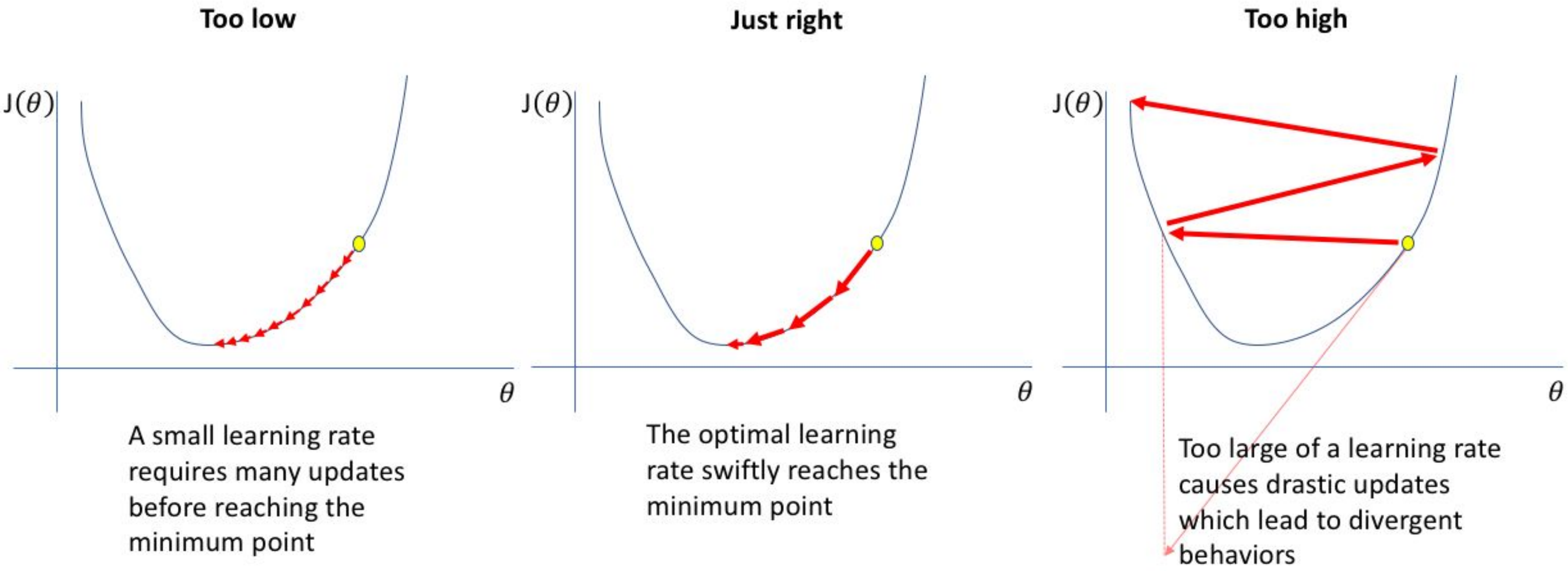
```
print("calculating accuracy...")
net.eval() # 학습하지 않을 것을 명시하여 자원낭비를 줄이는 코드

correct = 0
with torch.no_grad():
    for i, (X, t) in enumerate(test_loader):
        X = X.view(-1, 784)
        t = one_hot_embedding(t, 10)
        Y = net(X)

        onehot_y= softmax_to_one_hot(Y)
        correct += int(torch.sum(onehot_y * t)) # testset에서 정답을 맞춘 횟수 저장
print("Accuracy : %f" % (100. * correct / len(test_loader.dataset)))
plt.plot(train_loss_list)
```

Improving Deep Learning Networks

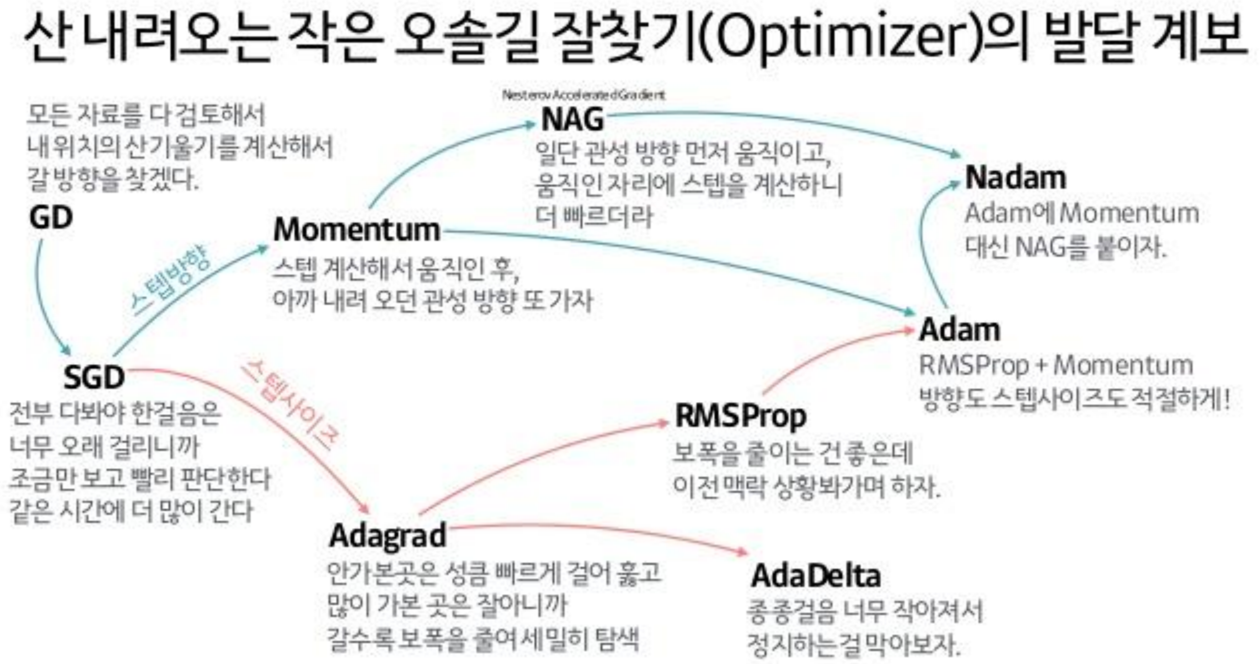
Data Preprocessing



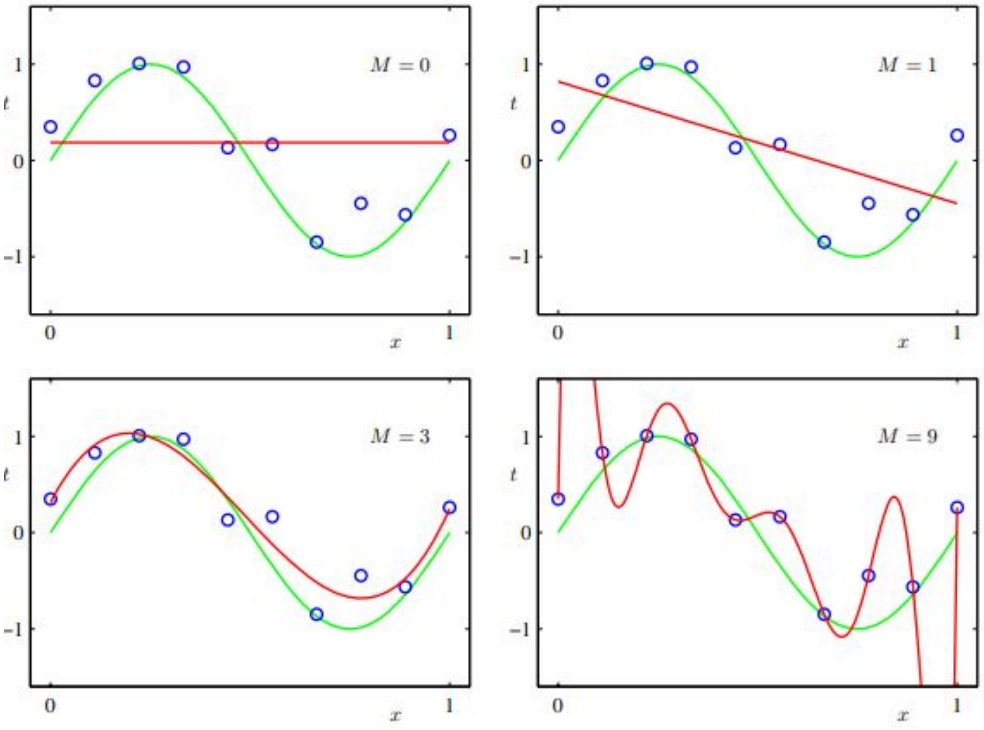
Learning rate



optimizers


산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보
모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
스텝사이즈
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
RMSProp
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.
Adagrad
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색
AdaDelta
종종걸음 너무 작아져서
정지하는걸 막아보자.

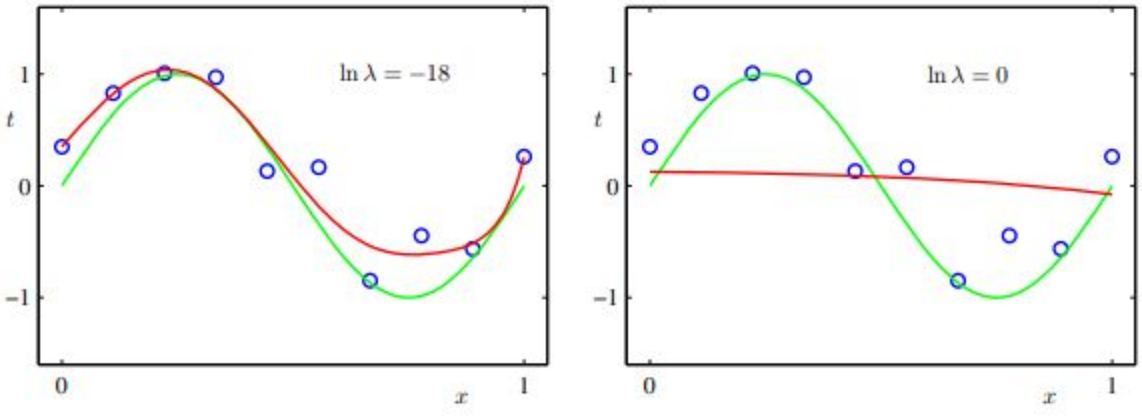
weight decay (L2 normalization)



	$M = 0$	$M = 1$	$M = 6$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

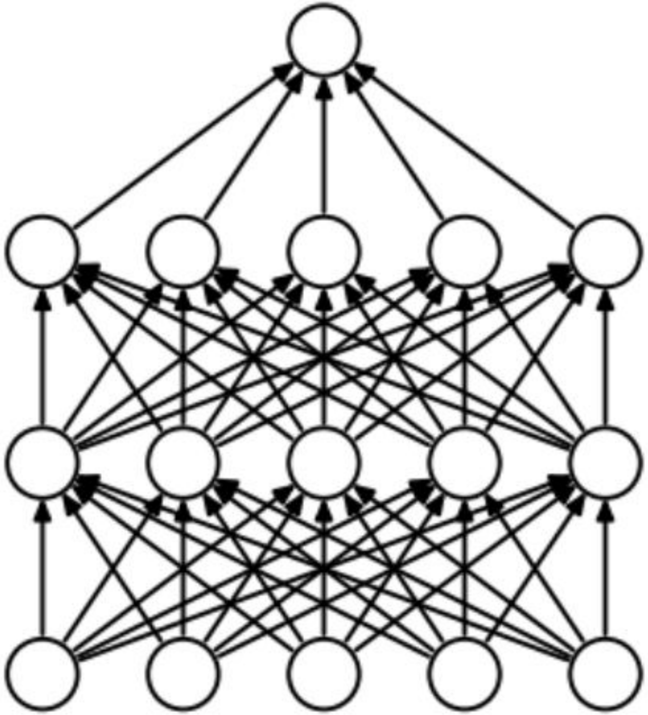
weight decay (L2 normalization)

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

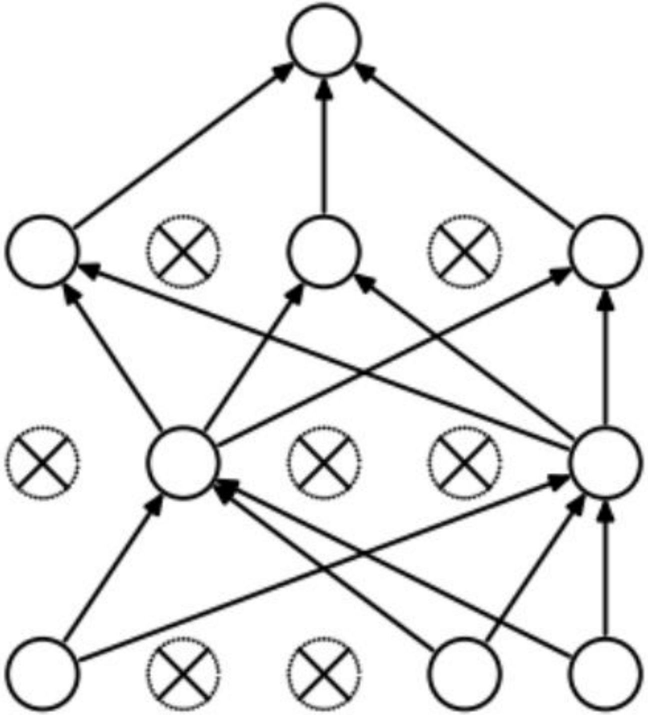


	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Dropout

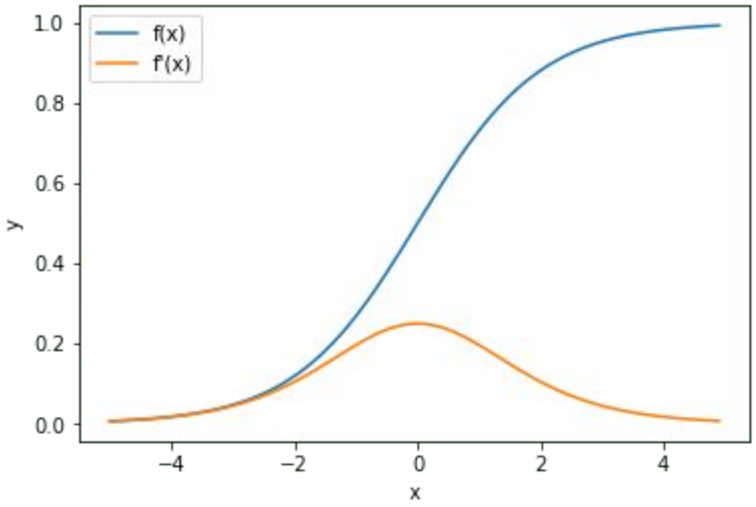


(a) Standard Neural Net



(b) After applying dropout.

Vanishing Gradient Problem

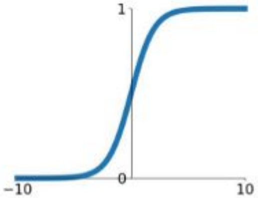


Activation Functions

Non Linearity!

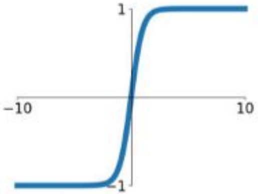
Sigmoid

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



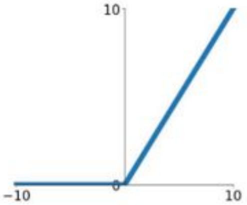
tanh

$\tanh(x)$



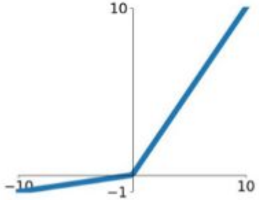
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

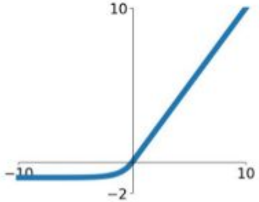


Maxout

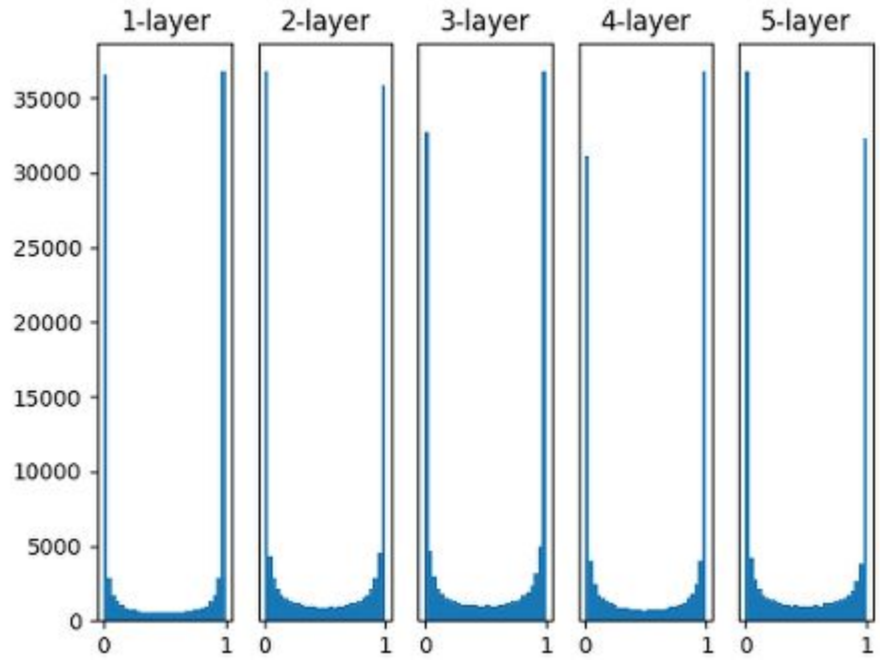
$\max(w_1^T x + b_1, w_2^T x + b_2)$

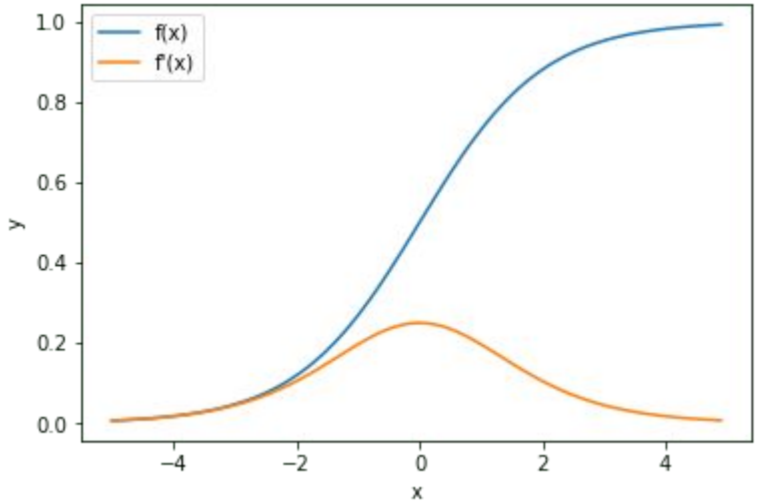
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

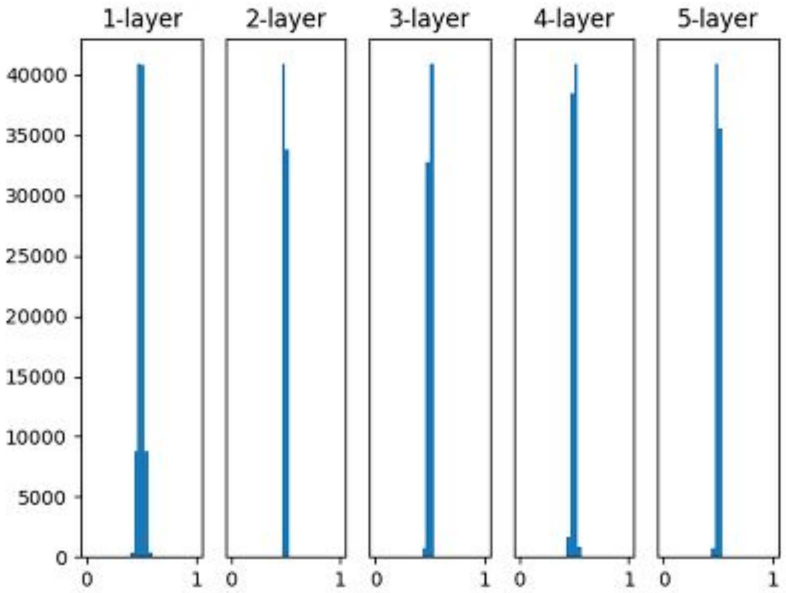


Weight Initialization

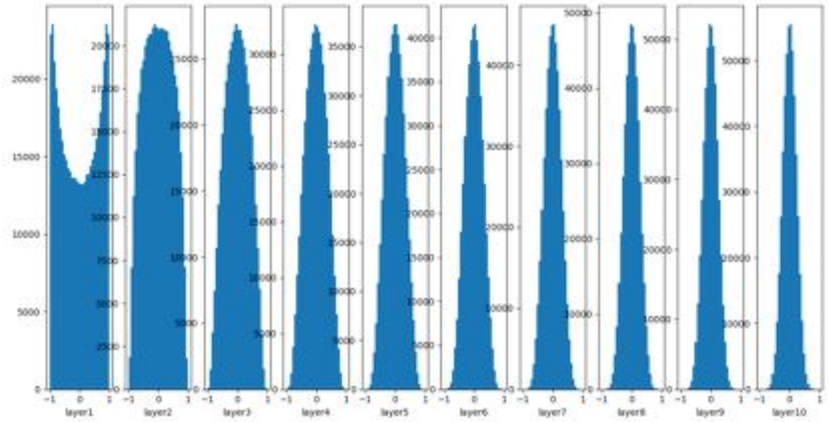




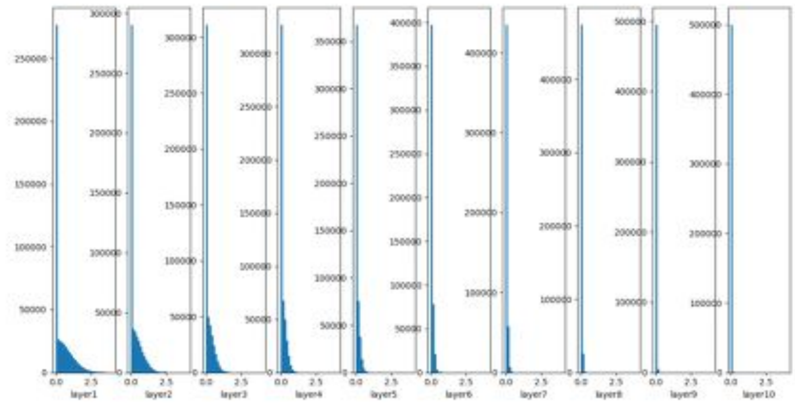
Weight Initialization - small std



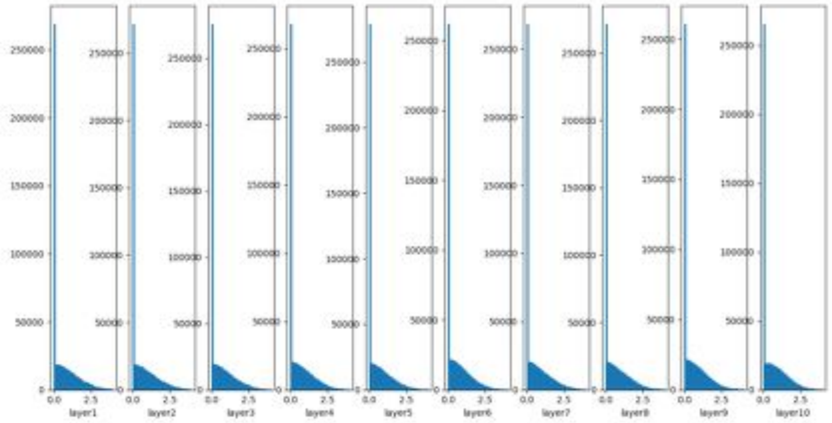
Weight Initialization - Xavier Initialization



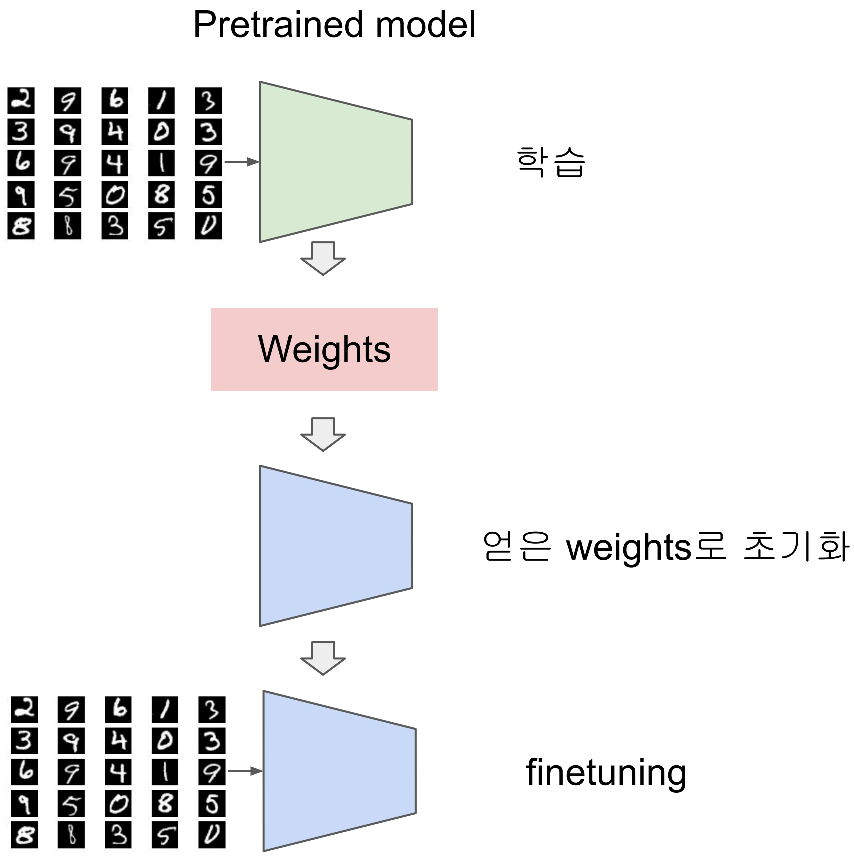
Weight Initialization - Xavier + Relu



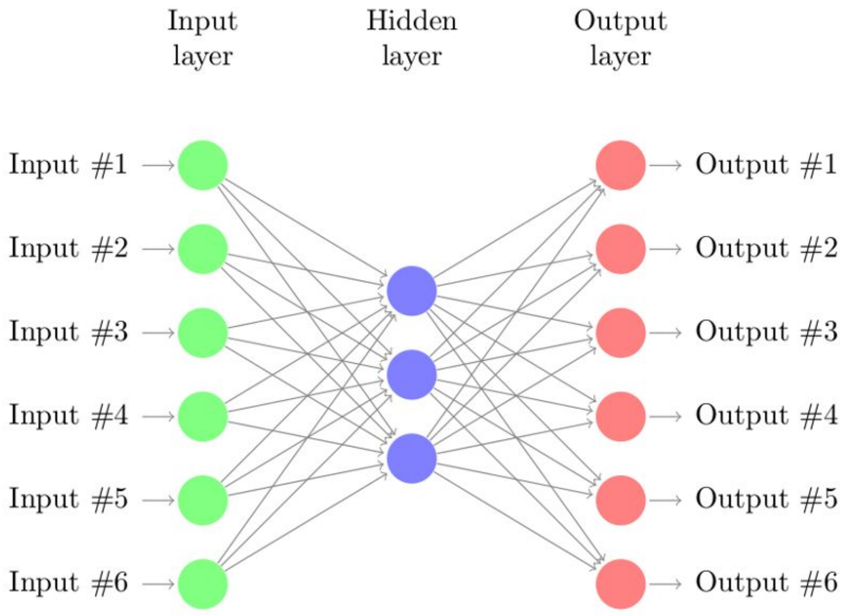
Weight Initialization - He Initialization + Relu



Fine - tuning



Weight Initialization - autoencoder



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

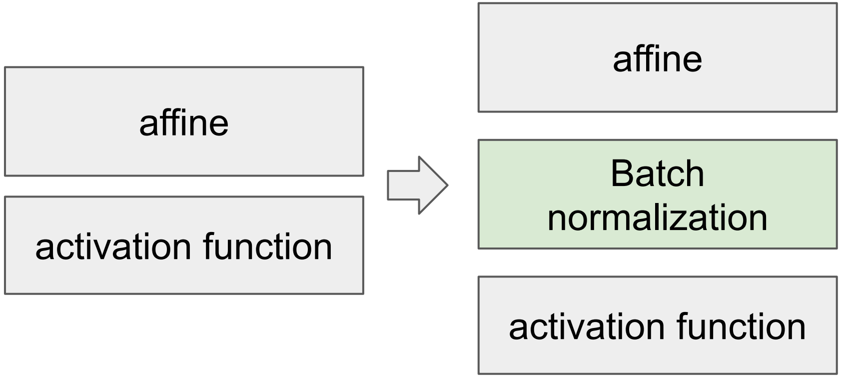
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

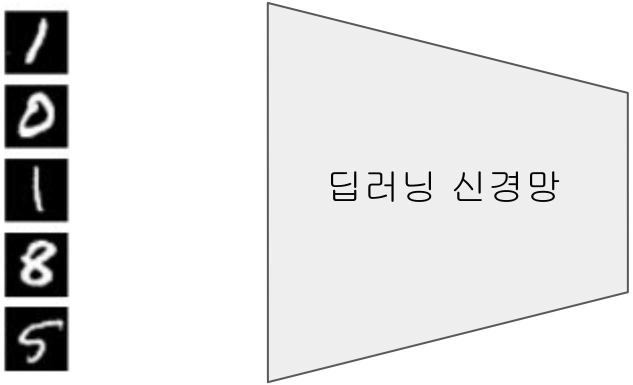


학습 속도 개선
초깃값 의존도 감소
오버피팅 억제

실습 7

MNIST Classifier 개선하기

입력



출력

[0,1,0,0,0,0,0,0,0,0]
[1,0,0,0,0,0,0,0,0,0]
[0,1,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,1,0]
[0,0,0,0,0,1,0,0,0,0]

실습 7

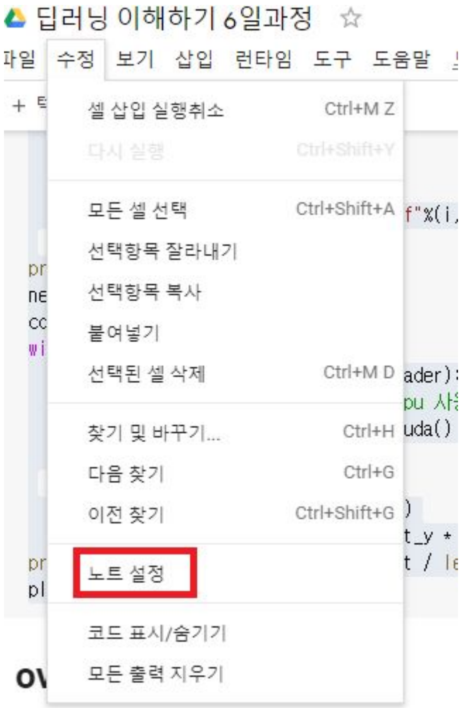
MNIST Classifier 개선하기

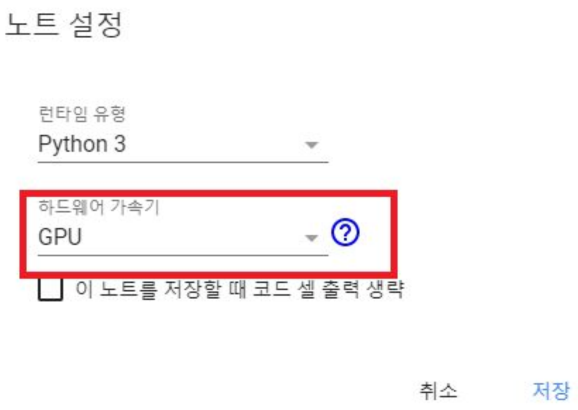
1. **Sigmoid => ReLU**
2. **Batch Normalization 추가(맨 마지막 레이어는 사용x)**
3. **SGD -> ADAM(학습속도 개선)**
4. **GPU 사용(cuda. 계산속도 개선)**

검색해보셔도 좋습니다.

실습 7

MNIST Classifier 개선하기
GPU 사용(cuda. 계산속도 개선)





실습 7

MNIST Classifier 개선하기

GPU 사용(cuda. 계산속도 개선)

1. model => model.cuda()
2. 계산 그래프를 시작하는 텐서 전부에 대하여 A => A.cuda()

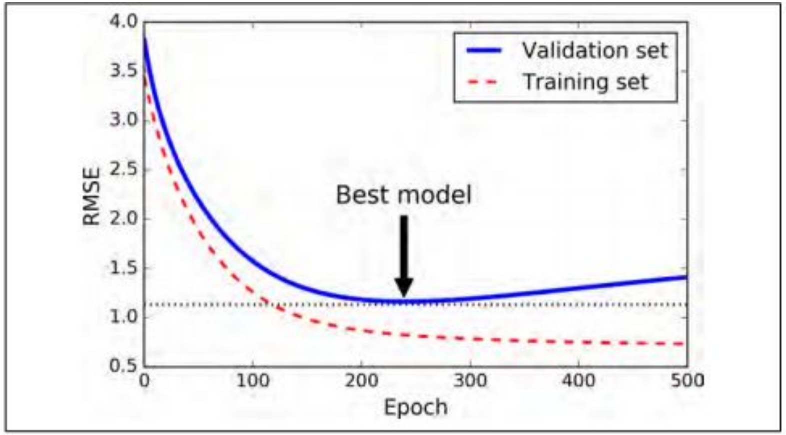
```
net = TwoLayerNet_pytorch(input_size=784, hidden_size=50, output_size=10).cuda() # gpu 사용.(뒤에 .cuda())
```

```
X = X.view(-1, 784).cuda() # gpu 사용.(뒤에 .cuda())
t = one_hot_embedding(t, 10).cuda() # gpu 사용.(뒤에 .cuda())
```

Validation

Validation

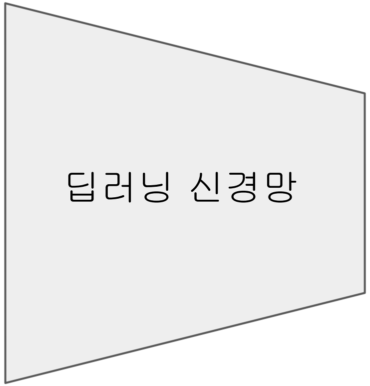
finding overfitting



실습 8

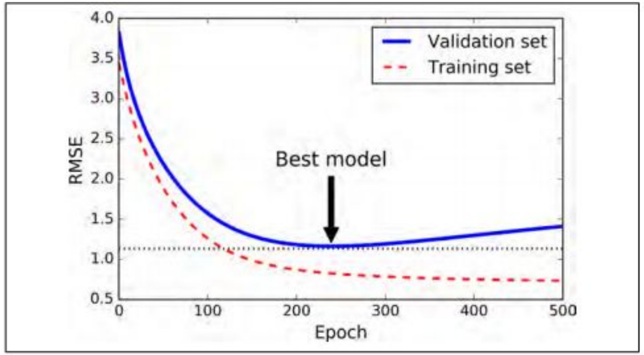
overfitting 검사하기

입력



출력

[0,1,0,0,0,0,0,0,0,0]
[1,0,0,0,0,0,0,0,0,0]
[0,1,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,1,0]
[0,0,0,0,0,1,0,0,0,0]



실습 8

```python
from torch.utils.data.sampler import SubsetRandomSampler
# load the dataset
dataset = datasets.MNIST('../data', train=True,
                         download=True, transform=transforms.Compose([
        transforms.ToTensor()
        , transforms.Normalize((0.1307,), (0.3081,))
    ]))
num_train = len(dataset)
valid_size = 500

indices = list(range(num_train))
split = num_train-valid_size
np.random.shuffle(indices)
train_idx, valid_idx = indices[:split], indices[split:]
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

train_loader = torch.utils.data.DataLoader(dataset,
                                           batch_size=batch_size, sampler=train_sampler)

valid_loader = torch.utils.data.DataLoader(dataset,
                                           batch_size=batch_size, sampler=valid_sampler)

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

실습 8

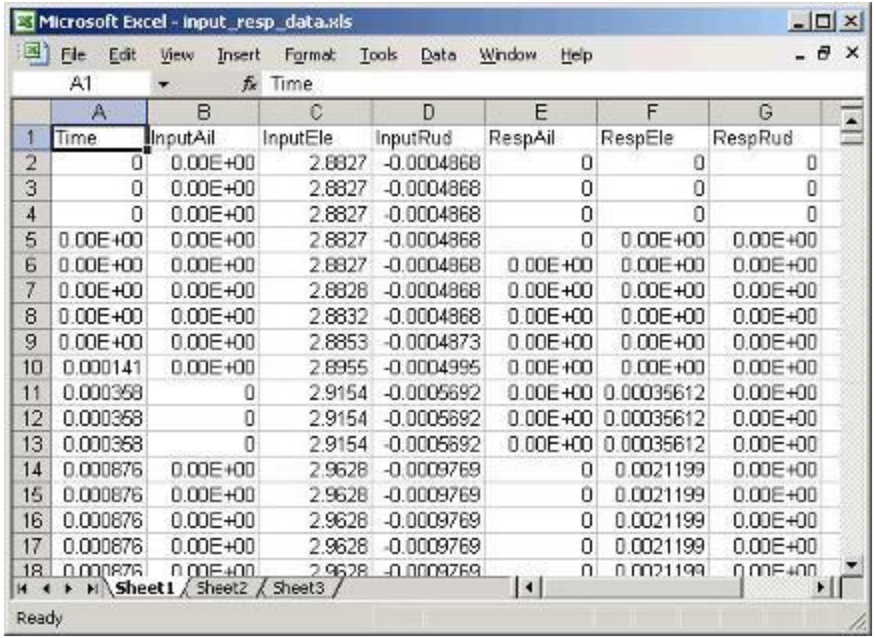
```
train_loss_list = []
val_loss_list = []
net.train()
for epoch in range(epochs):
    for i, (X, t) in enumerate(train_loader):
        X = X.view(-1, 784).cuda() # gpu 사용.(뒤에 .cuda())
        t = one_hot_embedding(t, 10).cuda() # gpu 사용.(뒤에 .cuda())

        Y = net(X)
        loss = loss_function(Y, t)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        #validation loss 계산. 계산이 무거우니 몇백 iteration혹은 몇 epoch마다 한번 수행하는것이 적당합니다. 예제는 매 100 iteration마다 수행합니다.
        if i % 100 == 0:
          with torch.no_grad():
              val_100_loss = []
              for (X, t) in valid_loader:
                X = X.view(-1, 784).cuda() # gpu 사용.(뒤에 .cuda())
                t = one_hot_embedding(t, 10).cuda() # gpu 사용.(뒤에 .cuda())

                Y = net(X)
                loss = loss_function(Y, t)
                val_100_loss.append(loss)

              # append loss
              train_loss_list.append(loss)
              val_loss_list.append(np.asarray(val_100_loss).sum()/len(valid_loader))
        print("[%d/%d][%d/%d] loss : %f"%(i,len(train_loader),epoch,epochs, loss))
```

실습 8

```
print("calculating accuracy...")
net.eval()
correct = 0
with torch.no_grad():
    for i, (X, t) in enumerate(test_loader):
        X = X.view(-1, 784).cuda() # gpu 사용.(뒤에 .cuda())
        t = one_hot_embedding(t, 10).cuda() # gpu 사용.(뒤에 .cuda())
        Y = net(X)

        onehot_y= softmax_to_one_hot(Y)
        correct += int(torch.sum(onehot_y * t))
print("Accuracy : %f" % (100. * correct / len(test_loader.dataset)))
plt.plot(np.column_stack((train_loss_list,val_loss_list)))
```

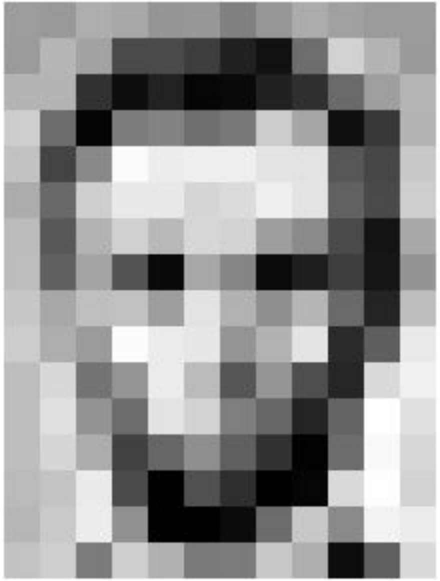
Image data

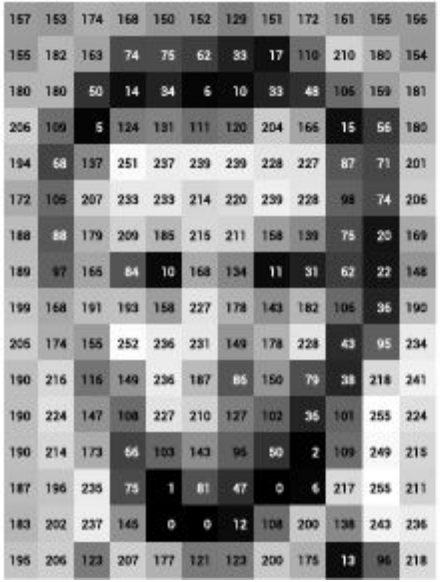
Data



	A	B	C	D	E	F	G
1	Time	InputAil	InputEle	InputRud	RespAil	RespEle	RespRud
2	0	0.00E+00	2.8827	-0.0004868	0	0	0
3	0	0.00E+00	2.8827	-0.0004868	0	0	0
4	0	0.00E+00	2.8827	-0.0004868	0	0	0
5	0.00E+00	0.00E+00	2.8827	-0.0004868	0	0.00E+00	0.00E+00
6	0.00E+00	0.00E+00	2.8827	-0.0004868	0.00E+00	0.00E+00	0.00E+00
7	0.00E+00	0.00E+00	2.8828	-0.0004868	0.00E+00	0.00E+00	0.00E+00
8	0.00E+00	0.00E+00	2.8832	-0.0004868	0.00E+00	0.00E+00	0.00E+00
9	0.00E+00	0.00E+00	2.8853	-0.0004873	0.00E+00	0.00E+00	0.00E+00
10	0.000141	0.00E+00	2.8955	-0.0004995	0.00E+00	0.00E+00	0.00E+00
11	0.000358	0	2.9154	-0.0005692	0.00E+00	0.00035612	0.00E+00
12	0.000358	0	2.9154	-0.0005692	0.00E+00	0.00035612	0.00E+00
13	0.000358	0	2.9154	-0.0005692	0.00E+00	0.00035612	0.00E+00
14	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00
15	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00
16	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00
17	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00
18	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00

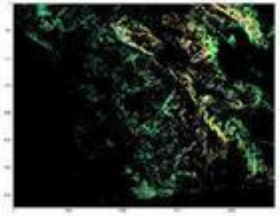
image data

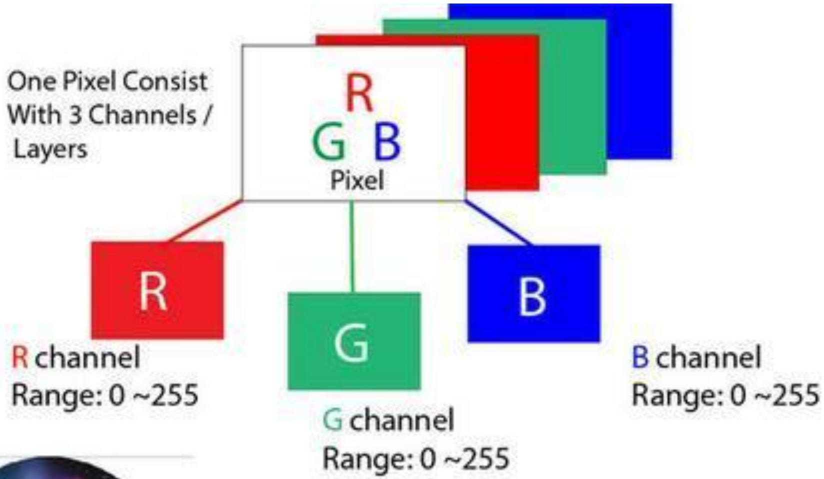




157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

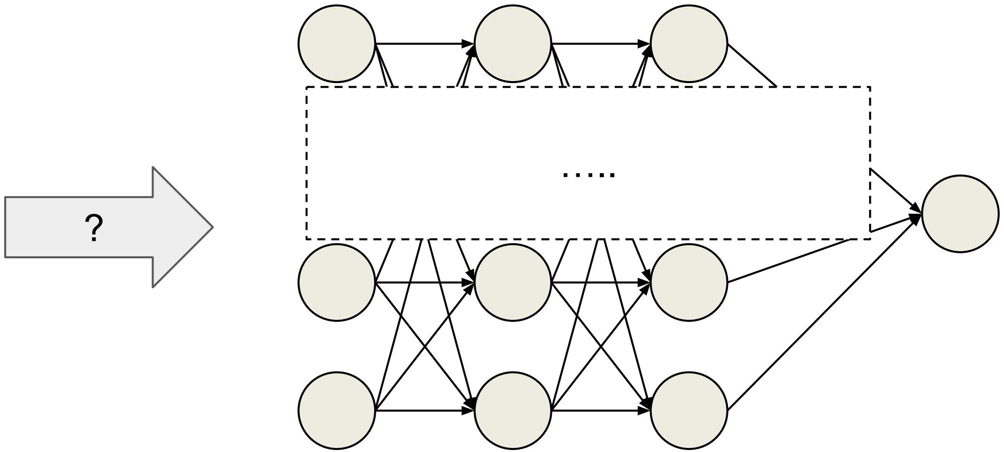
color image




One Pixel Consist
With 3 Channels /
Layers
R
G B
Pixel
R
G
B
R channel
Range: 0 ~255
G channel
Range: 0 ~255
B channel
Range: 0 ~255
@ITPUB博客

color image

4	5	9	10	47	0
0	74	5	78	5	6
12	34	0	8	1	8
2	5	8	1	8	7
84	87	48	87	85	3
1	22	45	43	21	44



color image

4	5	9	10	47	0
0	74	5	78	5	6
12	34	0	8	1	8
2	5	8	1	8	7
84	87	48	87	85	3
1	22	45	43	21	44

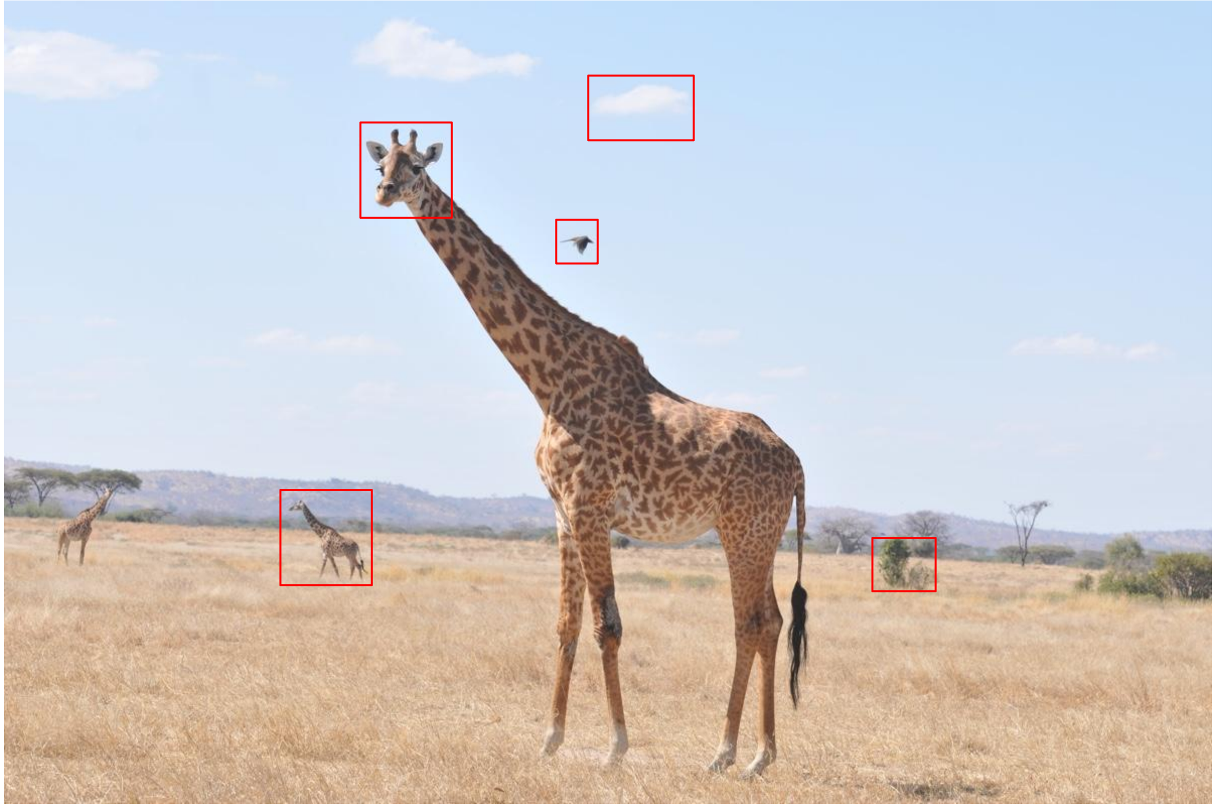
4	5	9	10	47	0	...	1	22	45	43	21	44

```
X = X.view(-1, 784) # 1 x 28 x 28 형태임으로, 784 형태의 벡터로 바꿔준다.
```

Convolutional Neural Network

신경망과 이미지

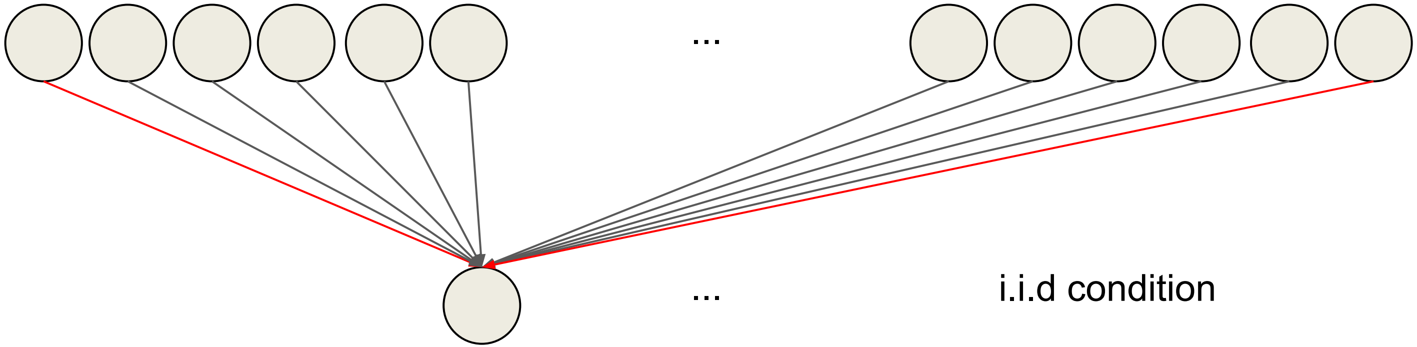
신경망과 이미지



vectorize(flatten)

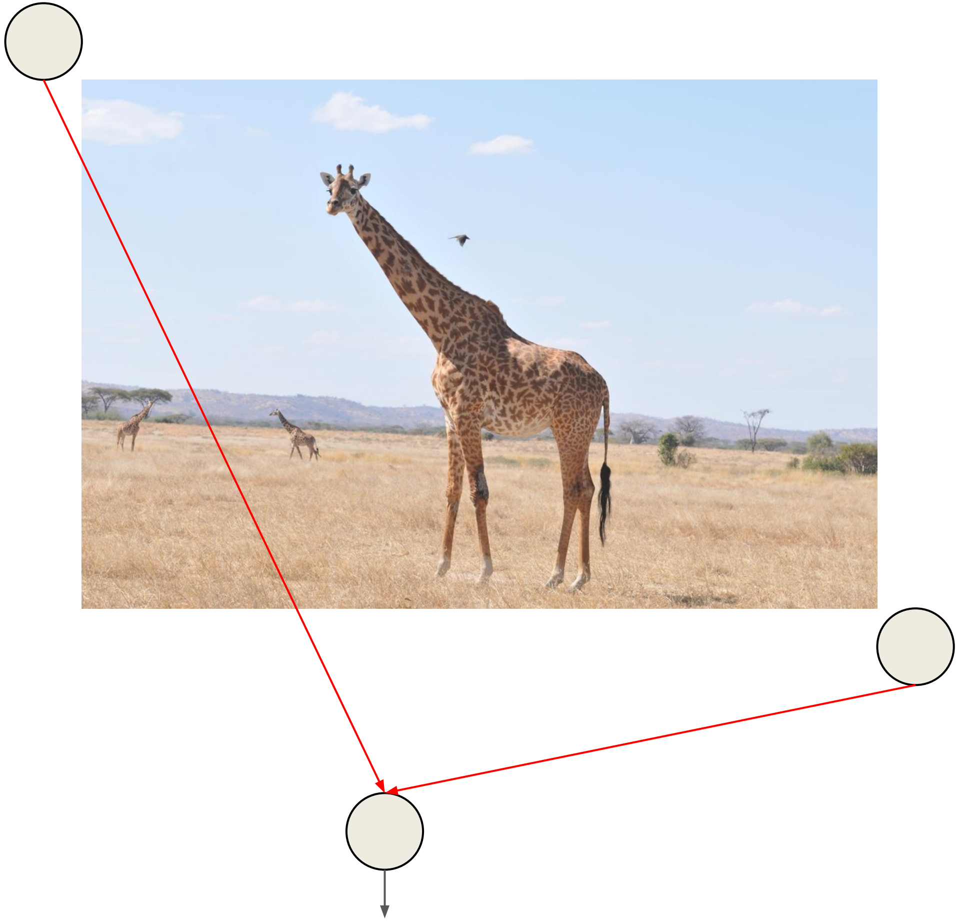
12	34	0	8	1	8
2	5	8	1	8	7
84	87	48	87	85	3
1	22	45	43	21	44

4	5	9	10	47	0	...	1	22	45	43	21	44

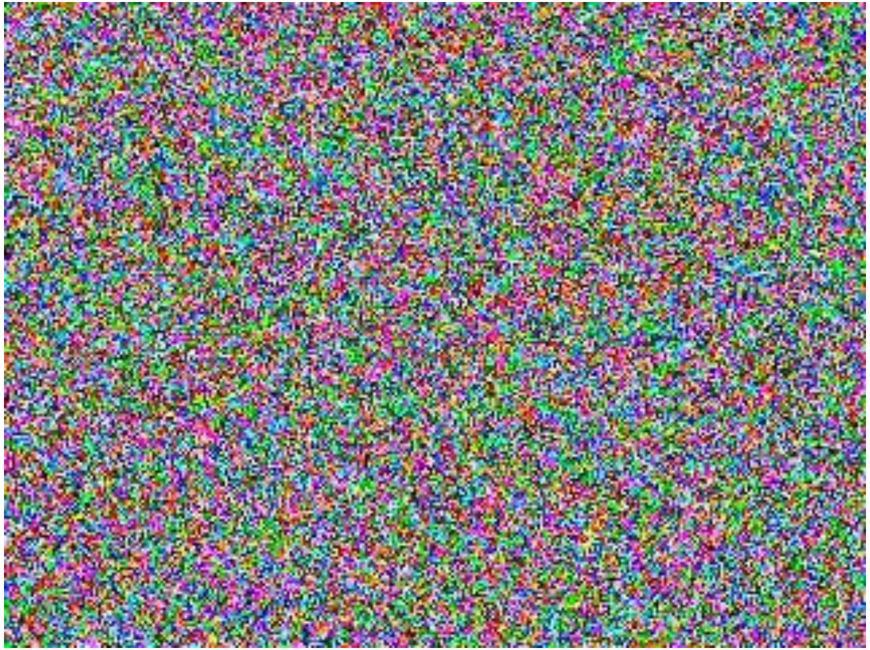


i.i.d condition

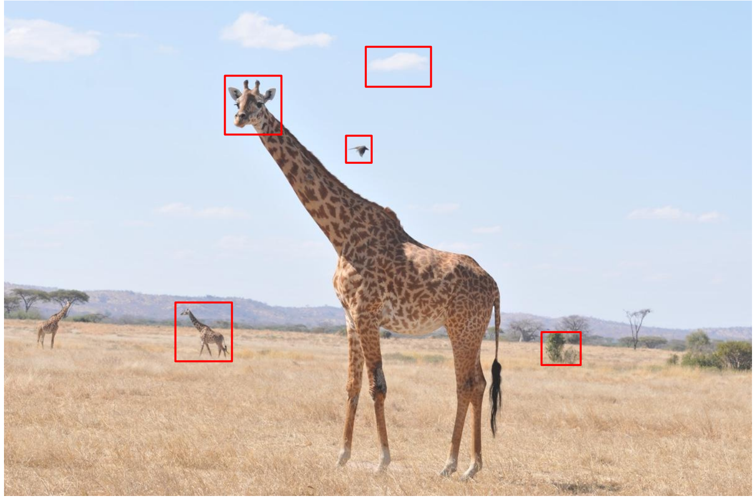
신경망과 이미지



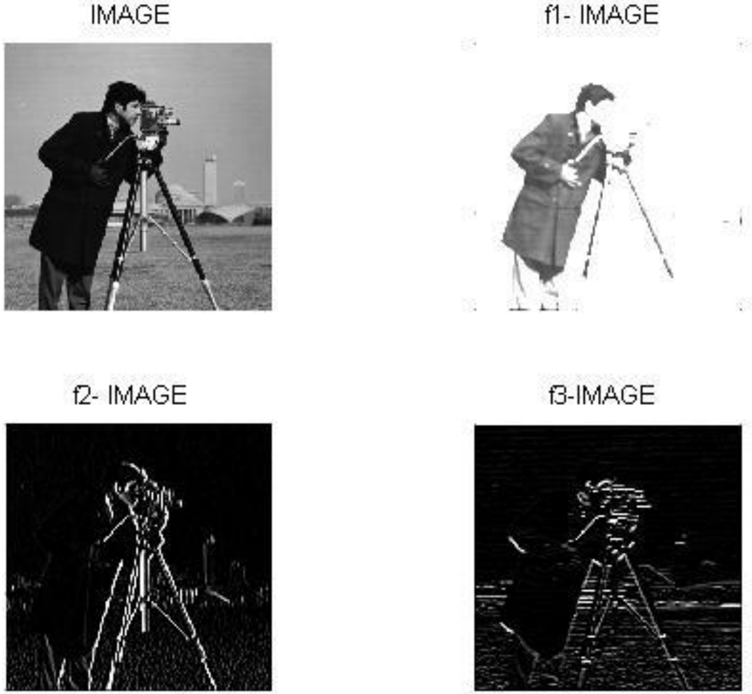
신경망과 이미지



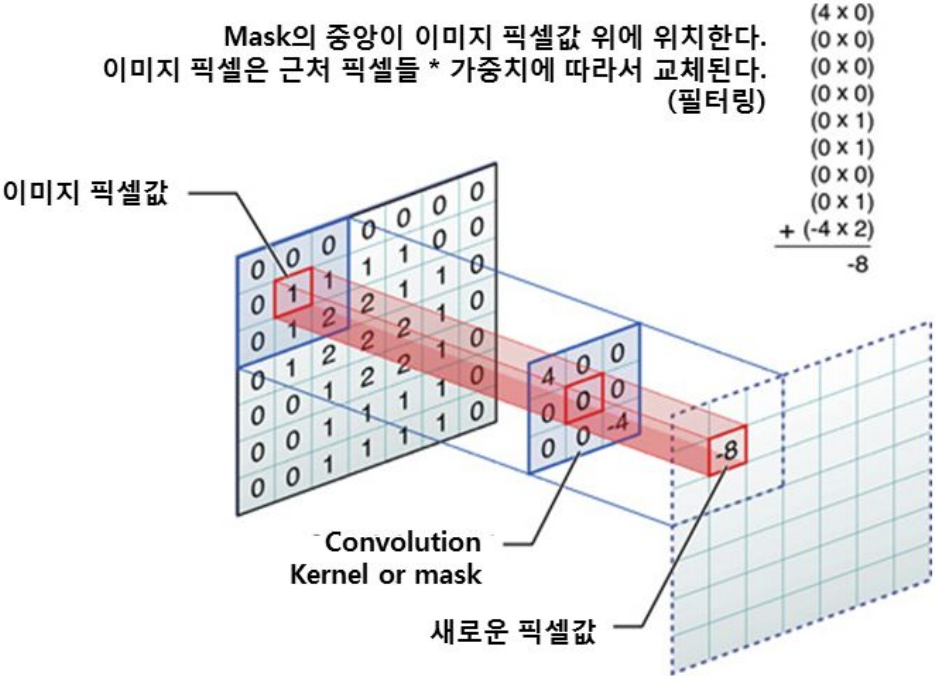
신경망과 이미지



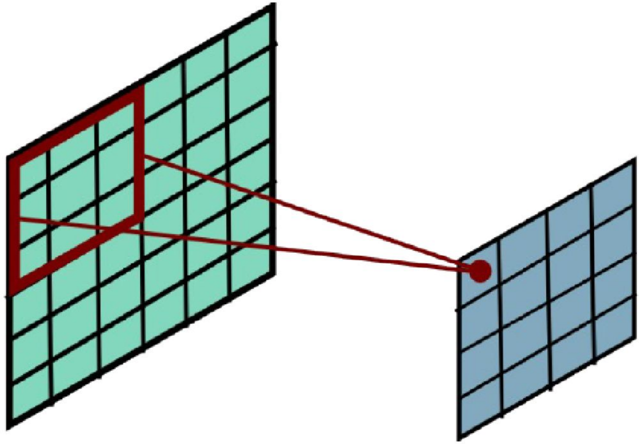
이미지 필터링



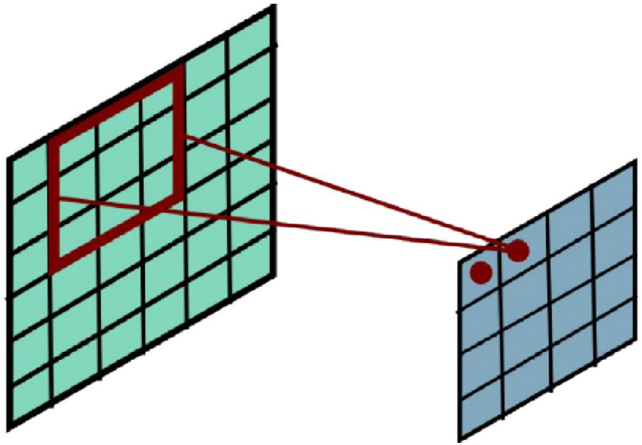
convolution



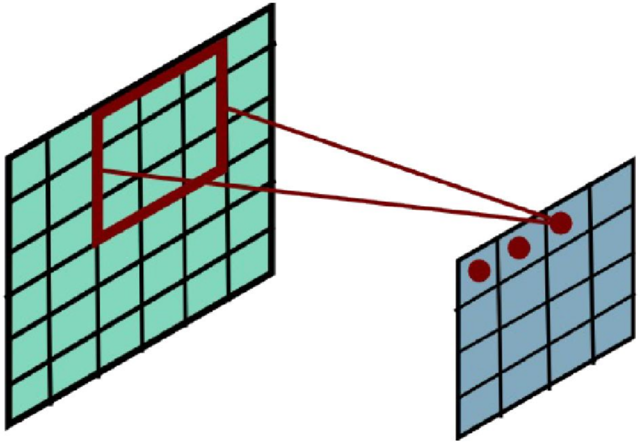
convolution



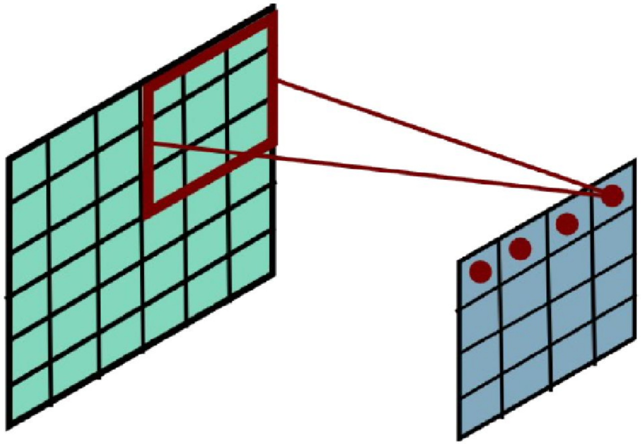
convolution



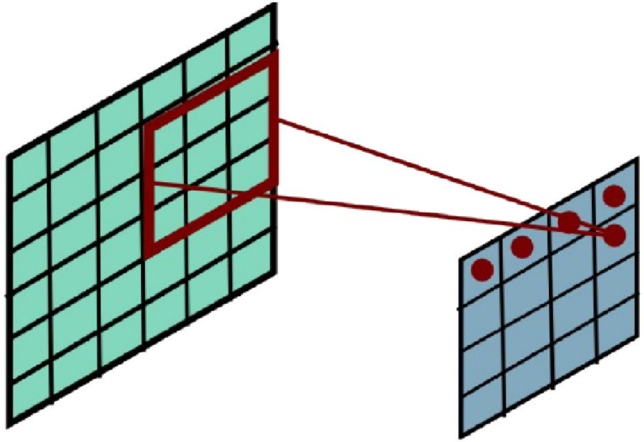
convolution



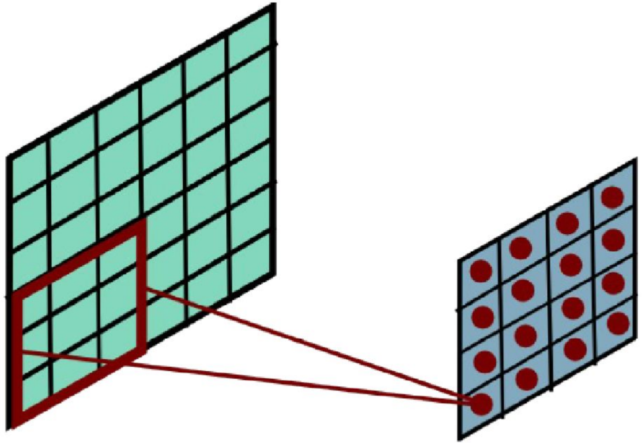
convolution



convolution



convolution



zero padding

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 15 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 6 & 8 \\ 6 & 15 & 14 \\ 12 & 14 & 9 \end{bmatrix}$$

stride

stride 1

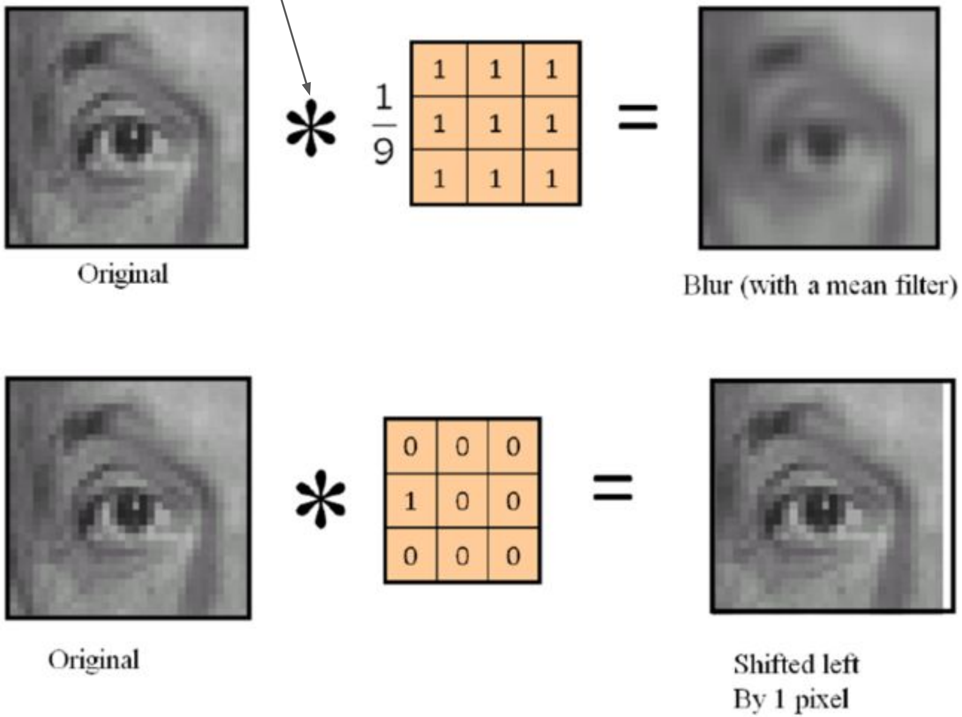
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 6 & 8 \\ 6 & 15 & 14 \\ 12 & 14 & 9 \end{bmatrix}$$

stride 2

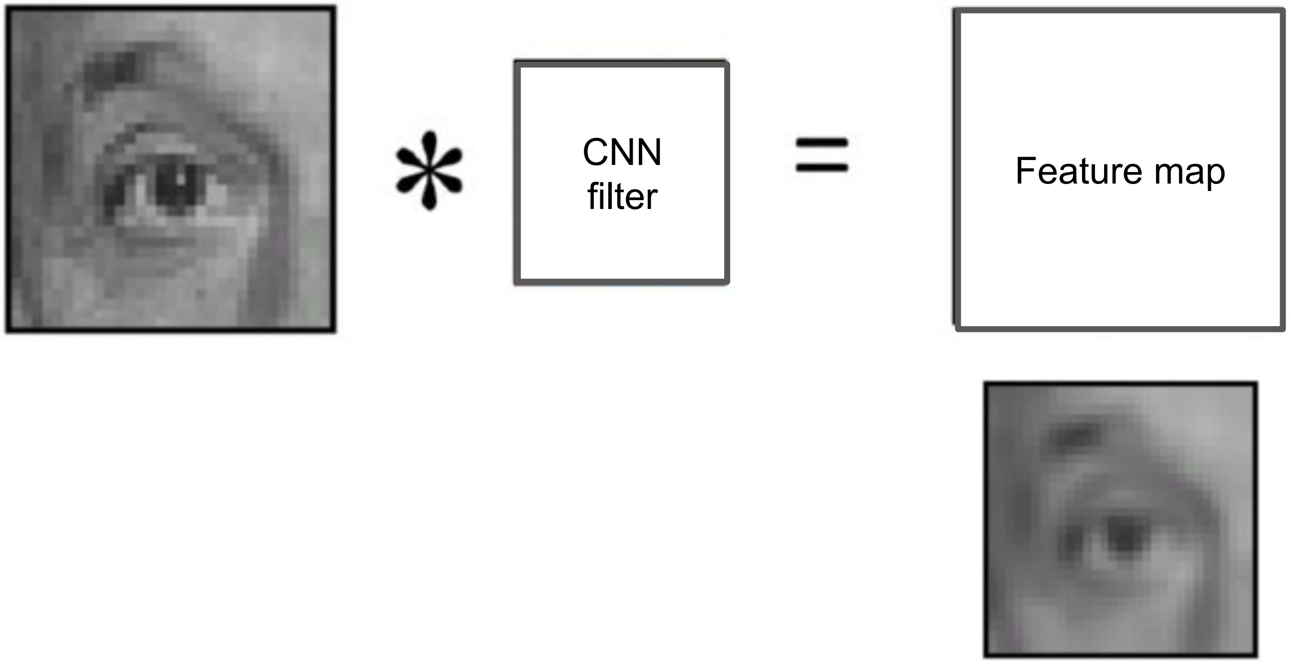
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 8 \\ 12 & 9 \end{bmatrix}$$

convolution

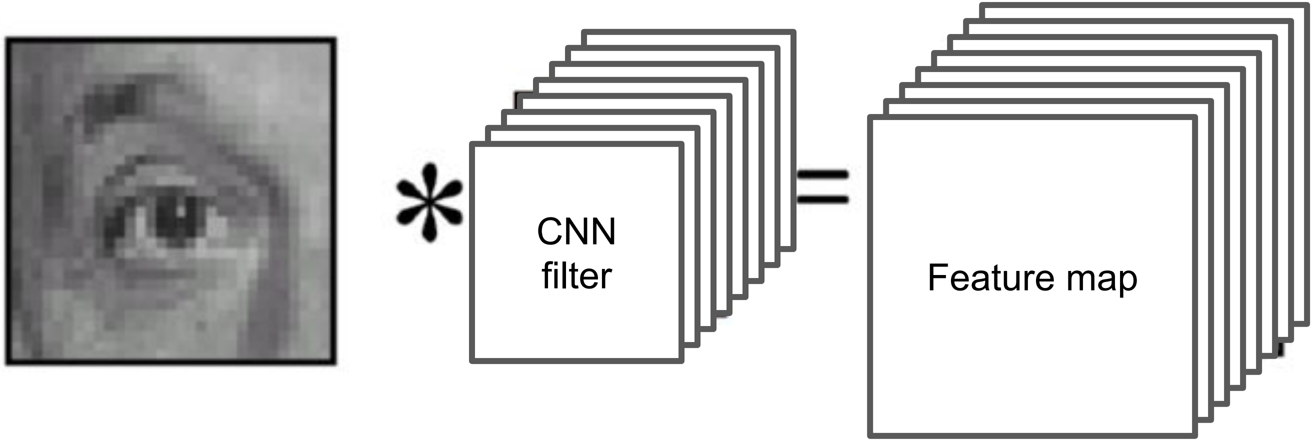
convolution 연산



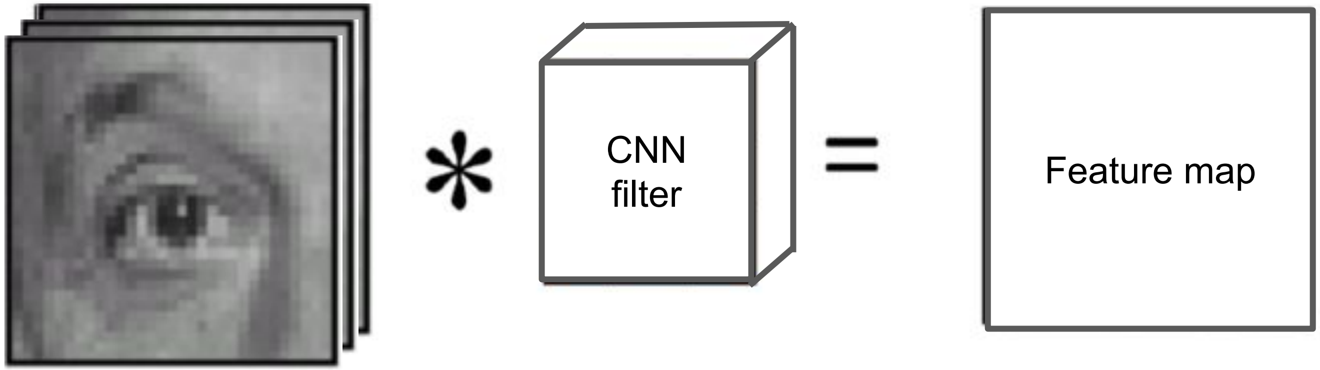
CNN



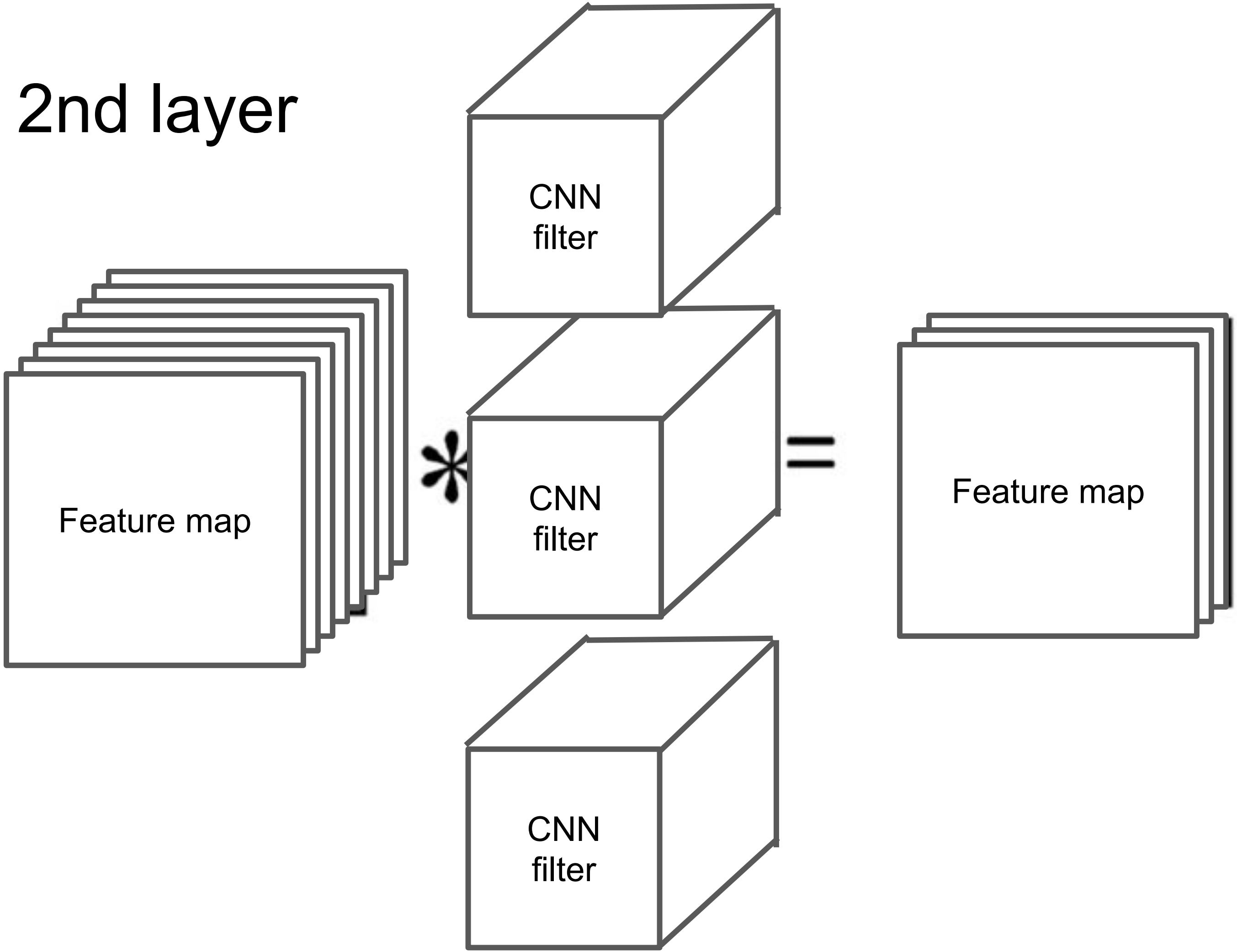
CNN



CNN - channel



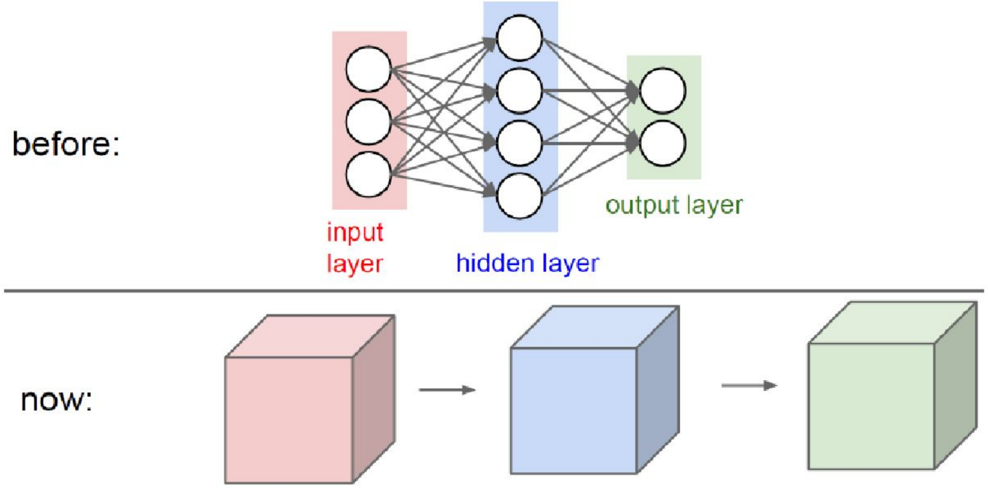
CNN 2nd layer



3 dim weight

- Number of filters (neurons) is considered as a new dimension (depth)

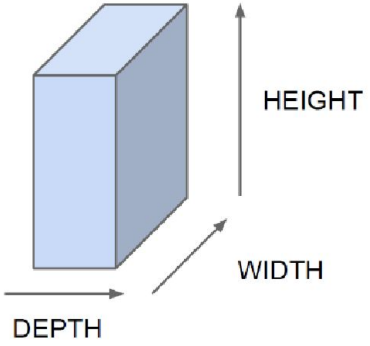
  ⇒ Volumetric representation



depth

- Number of filters (neurons) is considered as a new dimension (depth)
  $\Rightarrow$ Volumetric representation

All Neural Net activations arranged in **3 dimensions:**



For example, a CIFAR-10 image is a 32x32x3 volume
32 width, 32 height, 3 depth (RGB channels)

실습 9

CNN 연산 구현하기

0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

*

0	0	1
0	1	0
1	0	0

=

1	6	8
6	15	14
12	14	9

실습 9

CNN 연산 구현하기
convolution 연산의
행렬 연산을 위한
평탄화 코드

```python
import numpy as np
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).

    Parameters
    ----------
    input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비)
    filter_h : 필터의 높이
    filter_w : 필터의 너비
    stride : 스트라이드
    pad : 패딩

    Returns
    -------
    col : 2차원 배열
    """
    N, C, H, W = input_data.shape
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1

    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride*out_h
        for x in range(filter_w):
            x_max = x + stride*out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
    return col
```

실습 9

CNN 연산 구현하기
Convolution layer

```python
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

        # 중간 데이터 (backward 시 사용)
        self.x = None
        self.col = None
        self.col_W = None

        # 가중치와 편향 매개변수의 기울기
        self.dW = None
        self.db = None

    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
        out_w = 1 + int((W + 2*self.pad - FW) / self.stride)

        col = im2col(x, FH, FW, self.stride, self.pad)
        print("input data -> im2col is")
        print(col)
        col_W = self.W.reshape(FN, -1).T
        print("Weight = filter ... -> im2col is")
        print(col_W)

        out = np.dot(col, col_W) + self.b
        print("affine 연산 수행 결과")
        print(out)

        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W

        return out
```

print들은 실습을 위한 것으로, 실제 계산그래프에서는 모두 지웁니다.

실습 9

CNN 연산 구현하기
Convolution layer의 backward
(오늘 실습에선 안씁니다)

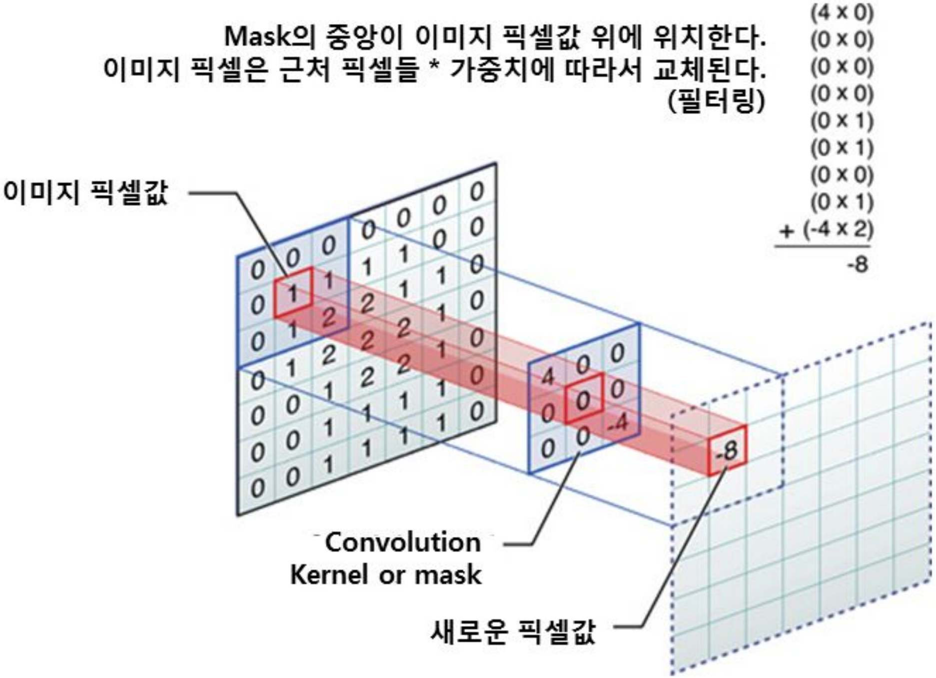
```
'''
def backward(self, dout):
    FN, C, FH, FW = self.W.shape
    dout = dout.transpose(0,2,3,1).reshape(-1, FN)

    self.db = np.sum(dout, axis=0)
    self.dW = np.dot(self.col.T, dout)
    self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

    dcol = np.dot(dout, self.col_W.T)
    dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

    return dx
'''
```

실습 9

CNN 연산 구현하기

```
filter_num = 1
input_channels = 1

# 입력데이터 만들기
x1 = np.array([[0,0,0,0,0],[0,1,2,3,0],[0,4,5,6,0],[0,7,8,9,0],[0,0,0,0,0]]).reshape(1, input_channels, 5, 5)
print("input data is")
print(x1)

# weight = convolution filter 만들기
W1 = np.array([[0,0,1],[0,1,0],[1,0,0]]).reshape([filter_num, input_channels, 3, 3])
b1 = np.zeros(filter_num) # bias는 0으로...
'''
W1 = np.array([[[0,0,1],[0,1,0],[1,0,0]],[[1,0,1],[0,1,0],[1,0,1]]]).reshape([2, input_channels, 3, 3])
b1 = np.zeros(2)
'''

print("weight = filter = kernel = mask is")
print(W1)

conv1 = Convolution(W1, b1) # convolution layer 정의
y=conv1.forward(x1) # convlution 연산 수행

print("convolution 수행 결과")
print(y)
```

실습 9



Mask의 중앙이 이미지 픽셀값 위에 위치한다.
이미지 픽셀은 근처 픽셀들 * 가중치에 따라서 교체된다.
(필터링)
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
이미지 픽셀값
Convolution
Kernel or mask
새로운 픽셀값

실습 9

```
input data is
[[[[0 0 0 0 0]
   [0 1 2 3 0]
   [0 4 5 6 0]
   [0 7 8 9 0]
   [0 0 0 0 0]]]]
```

```
weight = filter = kernel = mask is
[[[[0 0 1]
   [0 1 0]
   [1 0 0]]]]
```

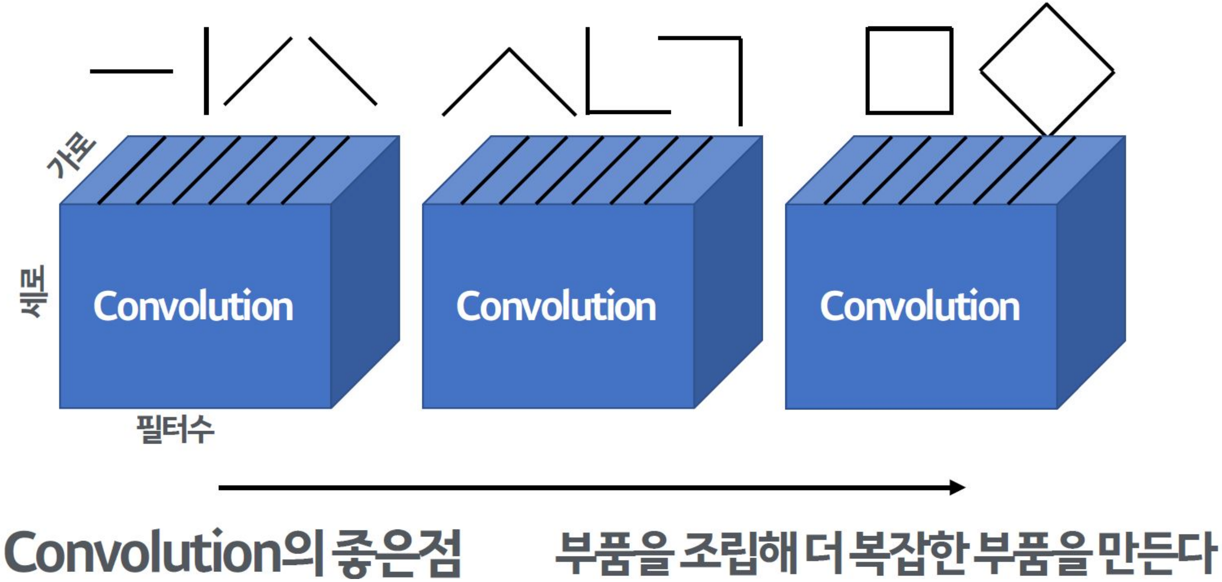
```
input data -> im2col is
[[0. 0. 0. 0. 1. 2. 0. 4. 5.]
 [0. 0. 0. 1. 2. 3. 4. 5. 6.]
 [0. 0. 0. 2. 3. 0. 5. 6. 0.]
 [0. 1. 2. 0. 4. 5. 0. 7. 8.]
 [1. 2. 3. 4. 5. 6. 7. 8. 9.]
 [2. 3. 0. 5. 6. 0. 8. 9. 0.]
 [0. 4. 5. 0. 7. 8. 0. 0. 0.]
 [4. 5. 6. 7. 8. 9. 0. 0. 0.]
 [5. 6. 0. 8. 9. 0. 0. 0. 0.]]
Weight = filter ... -> im2col is
```

```
eight = filter ... -> im2col is
[[0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
 [0]
 [0]]
```

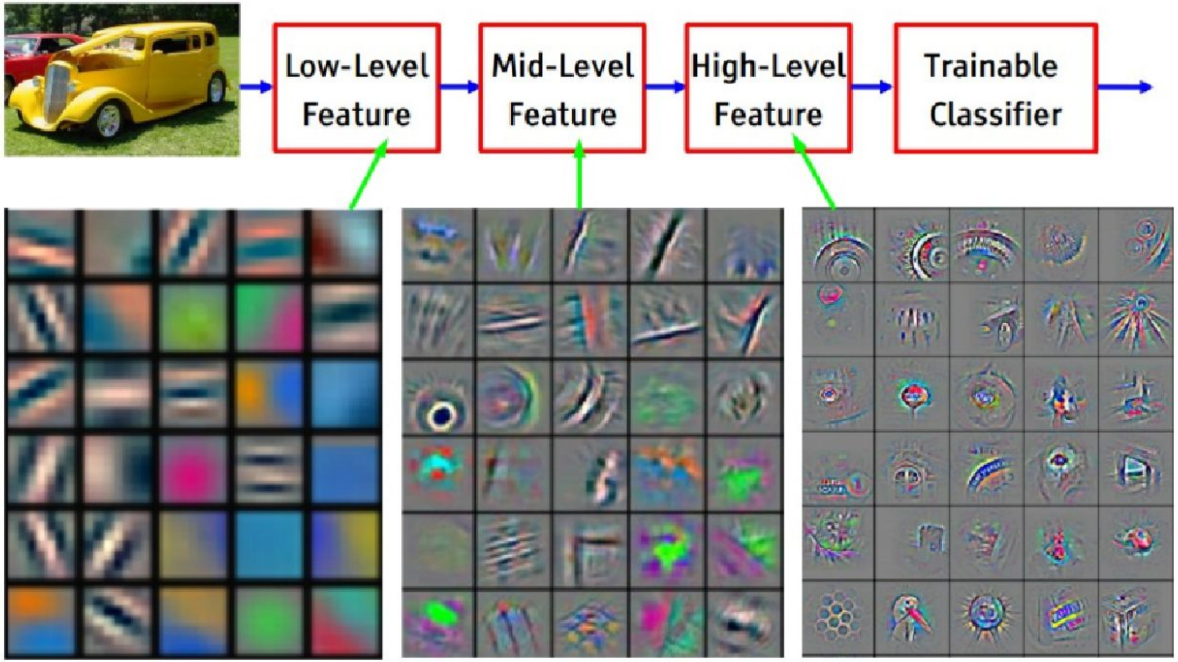
```
affine 연산 수행 결과
[[ 1.]
 [ 6.]
 [ 8.]
 [ 6.]
 [15.]
 [14.]
 [12.]
 [14.]
 [ 9.]]
```

```
convolution 수행 결과
[[[[ 1.  6.  8.]
   [ 6. 15. 14.]
   [12. 14.  9.]]]]
```

feature

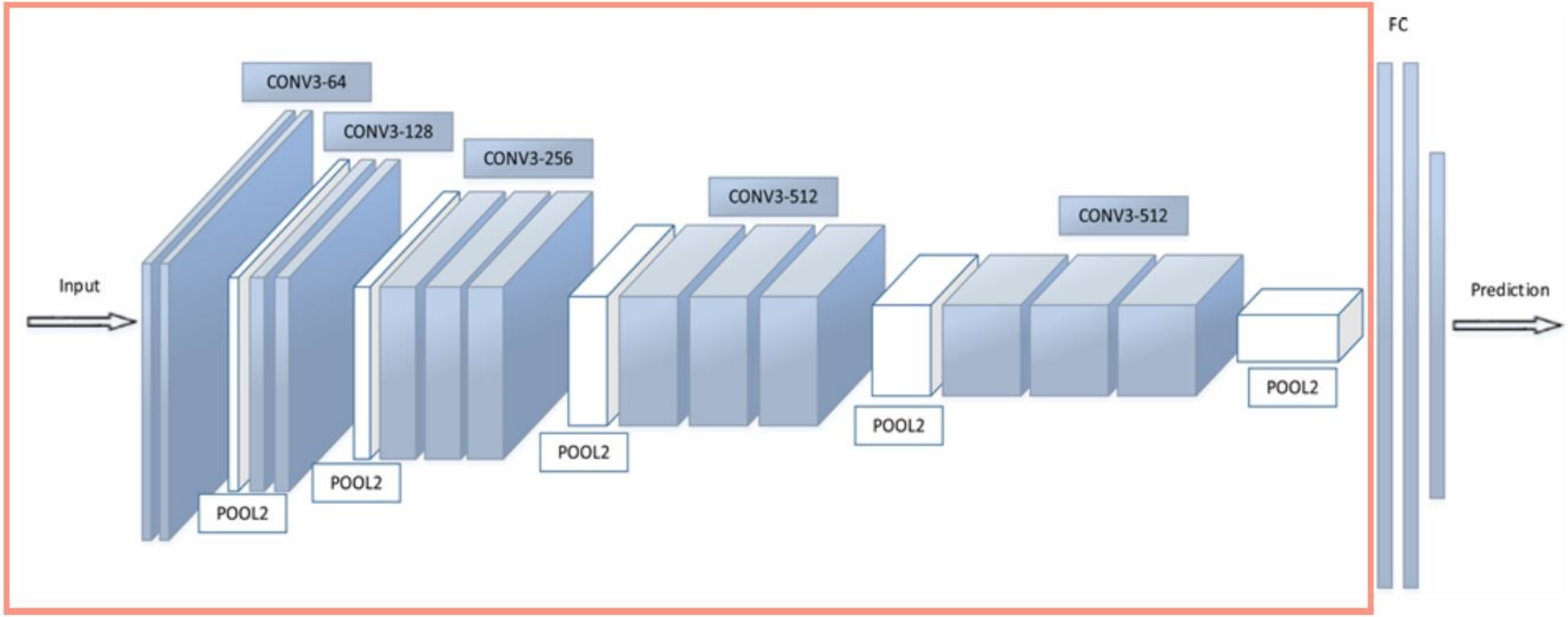

가로
세로
Convolution
Convolution
Convolution
필터수
Convolution의 좋은점
부품을 조립해 더 복잡한 부품을 만든다

feature map



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN classifier



Pooling

Activation Map

12	20	30	0
8	12	2	0
34	70	37	7
112	100	22	12

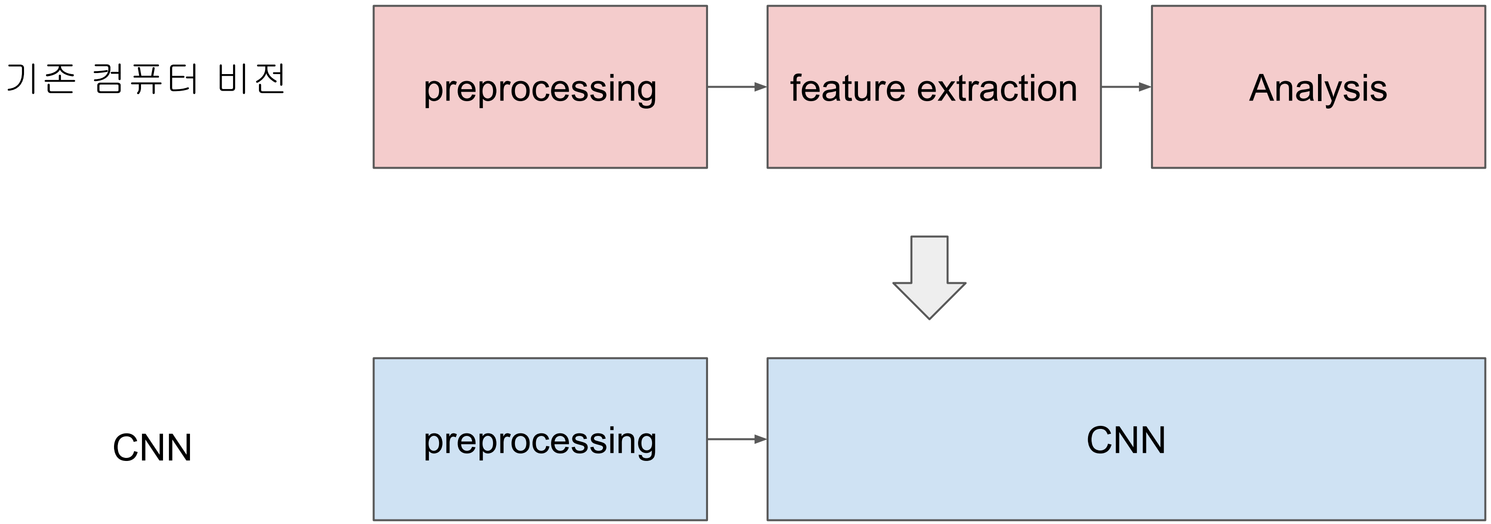
Max Pooling

20	30
112	37

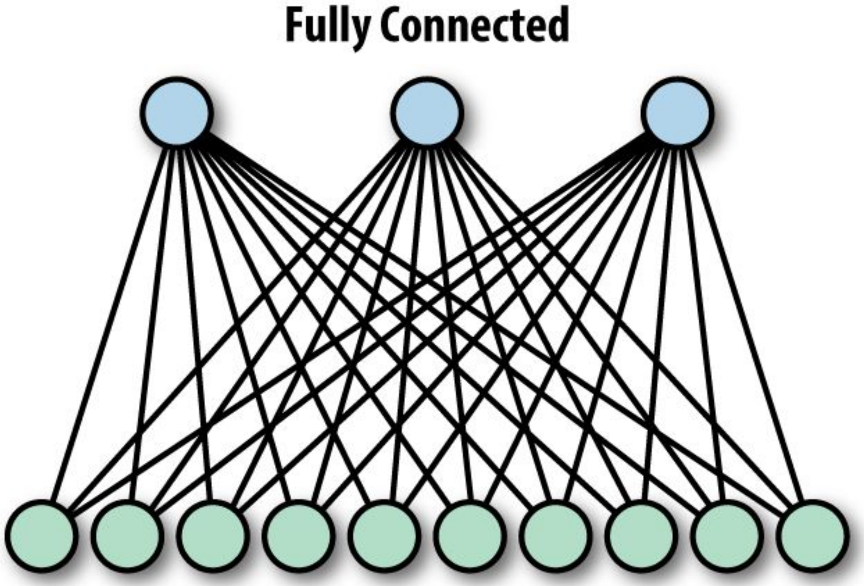
Average Pooling

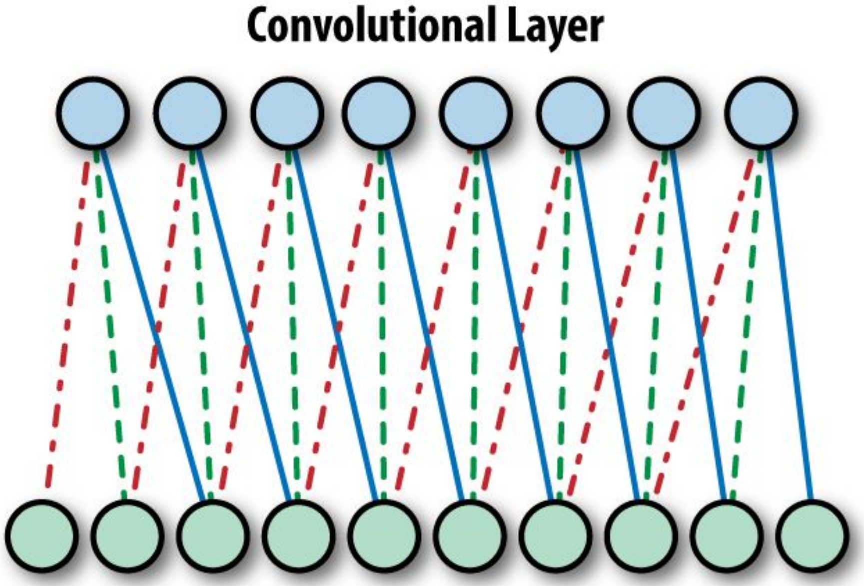
13	8
79	18

CNN


기존 컴퓨터 비전
preprocessing
feature extraction
Analysis
CNN
preprocessing
CNN

wasted weights



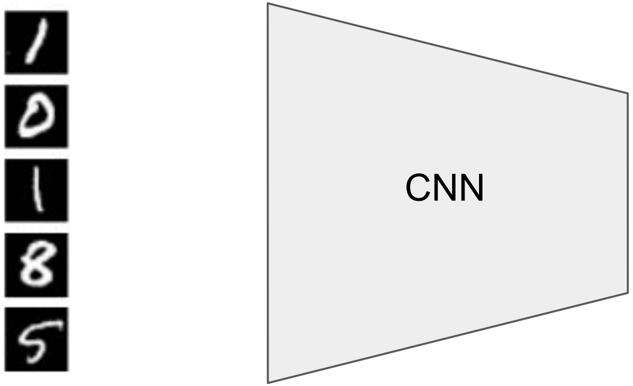


실습 10

CNN classifier 만들기

입력

출력



[0,1,0,0,0,0,0,0,0,0]
[1,0,0,0,0,0,0,0,0,0]
[0,1,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,1,0]
[0,0,0,0,0,1,0,0,0,0]

실습 10

```python
class MNIST_classifier_CNN(nn.Module):
  def __init__(self, class_num):
    super().__init__()
    self.class_num = class_num

    self.conv_net = nn.Sequential(
        nn.Conv2d(in_channels=1, out_channels=10, kernel_size=5),
        nn.BatchNorm2d(10),
        nn.MaxPool2d(2),
        nn.ReLU(),

        nn.Conv2d(in_channels=10, out_channels=20, kernel_size=5),
        nn.BatchNorm2d(20),
        nn.MaxPool2d(2),
        nn.ReLU()
    )
    self.fc_net = nn.Sequential(
        nn.Linear(320,50),
        nn.BatchNorm1d(50),
        nn.ReLU(),
        nn.Linear(50,self.class_num),
        nn.Softmax()
    )
  def forward(self, x):
    feature = self.conv_net(x)
    feature = feature.view(-1,320)
    y = self.fc_net(feature)
    return y
```

```python
net = MNIST_classifier_CNN(class_num=10).cuda() # gpu 사용.(뒤에 .cuda())
net.apply(weight_init)
```

실습 10

```python
def weight_init(m):
    # Conv layer와 batchnorm layer를 위한 가중치 초기화를 추가함.
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)
    elif classname.find('Linear')!=-1:
        m.weight.data.normal_(0.0, 0.02)
        m.bias.data.fill_(0)
```

실습 10

```
train_loss_list = []
val_loss_list = []
net.train()
for epoch in range(epochs):
    for i, (X, t) in enumerate(train_loader):
        X = X.cuda() # gpu 사용.(뒤에 .cuda()) => view를 이용해 vectorize하는 부분 사라짐
        t = one_hot_embedding(t, 10).cuda() # gpu 사용.(뒤에 .cuda())

        Y = net(X)
```