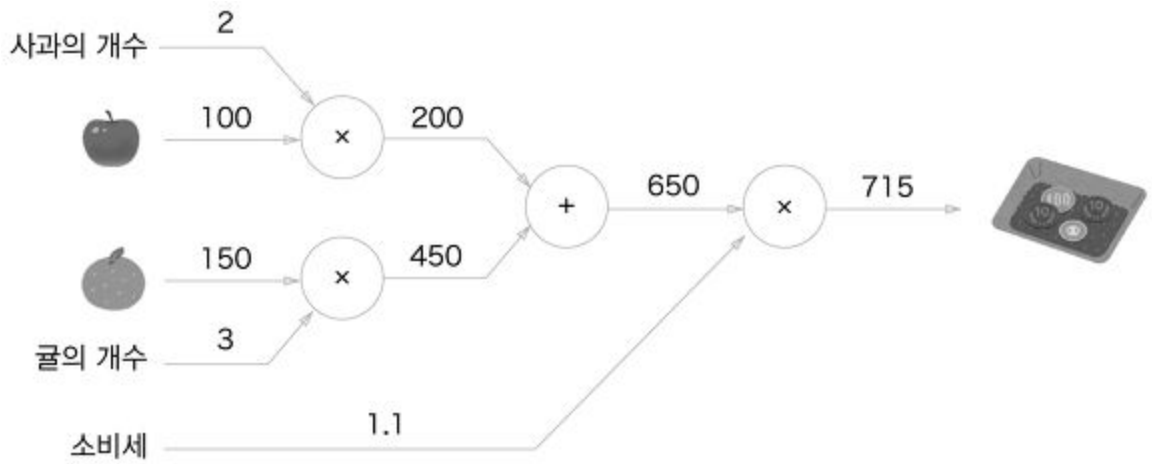
딥러닝 이해하기

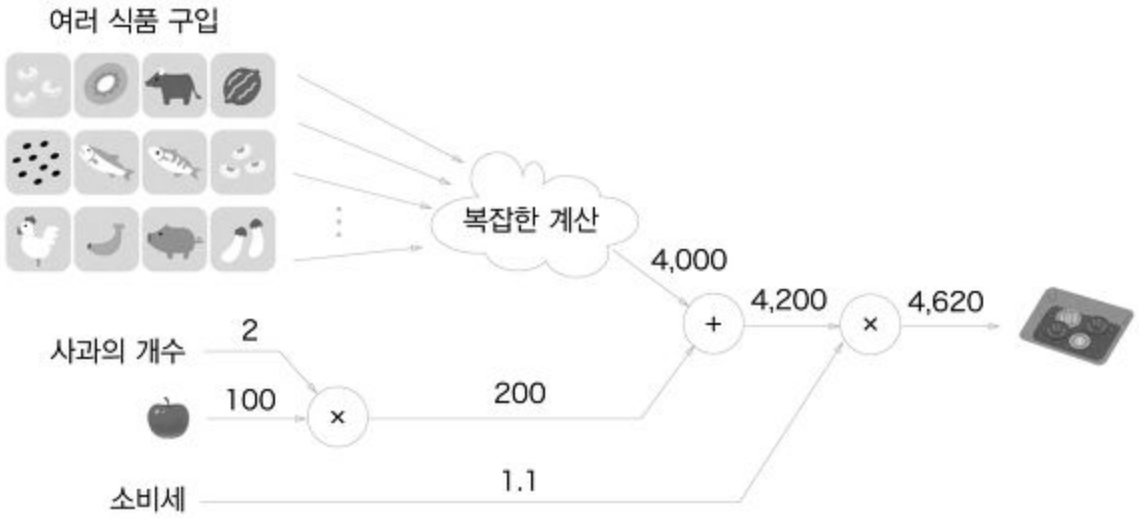
leejeyeol92@gmail.com

Computational Graph

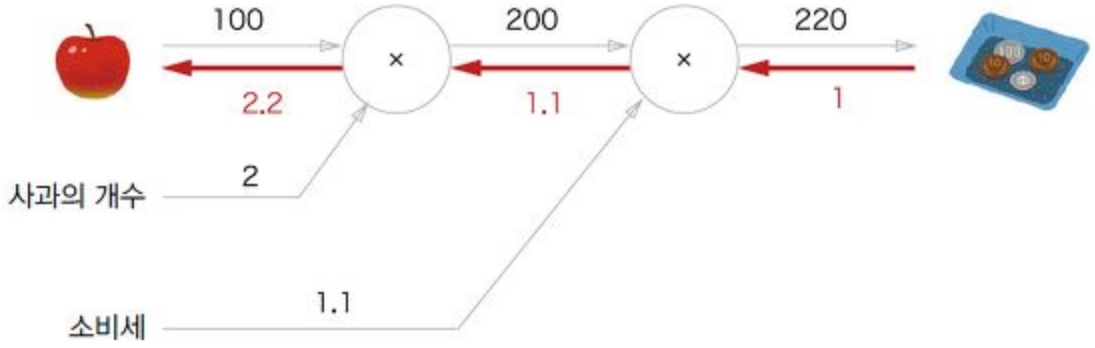
Computational Graph



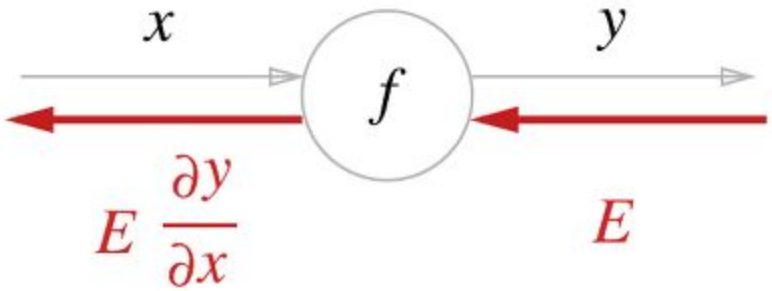
Computational Graph



Computational Graph



backward



chain rule

$$t = x + y$$

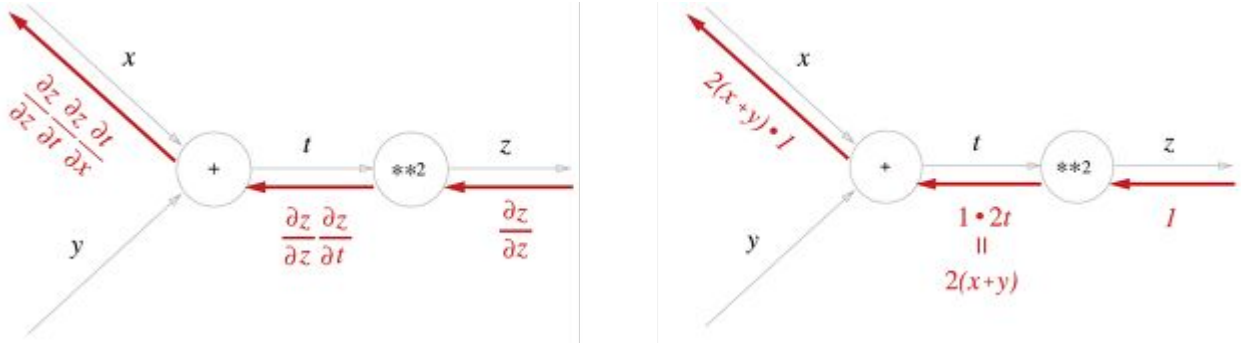
$$z = t^2$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

$$\frac{\partial z}{\partial t} = 2t$$

$$\frac{\partial t}{\partial x} = 1$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x + y)$$

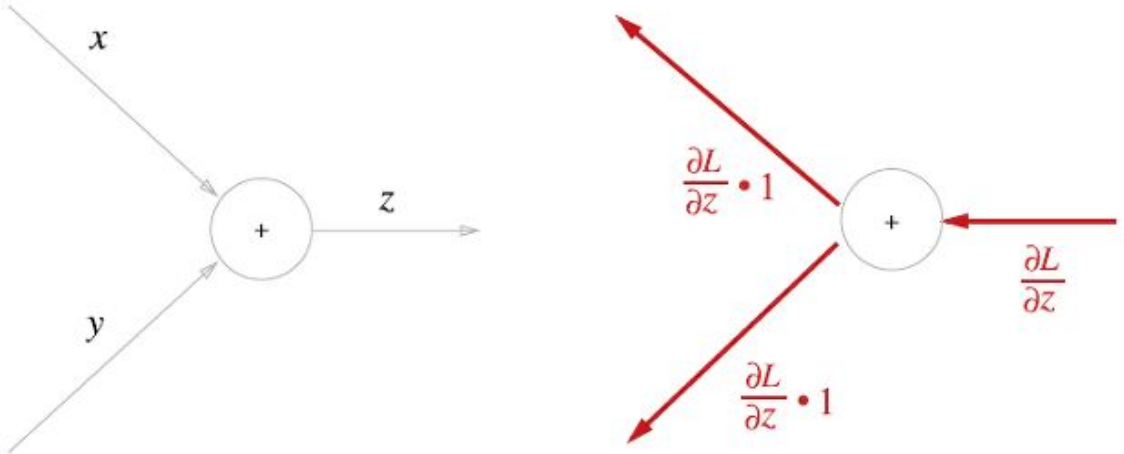


덧셈 그래프

$$z = x + y$$

$$\frac{\partial z}{\partial x} = 1$$

$$\frac{\partial z}{\partial y} = 1$$



덧셈 그래프 코드

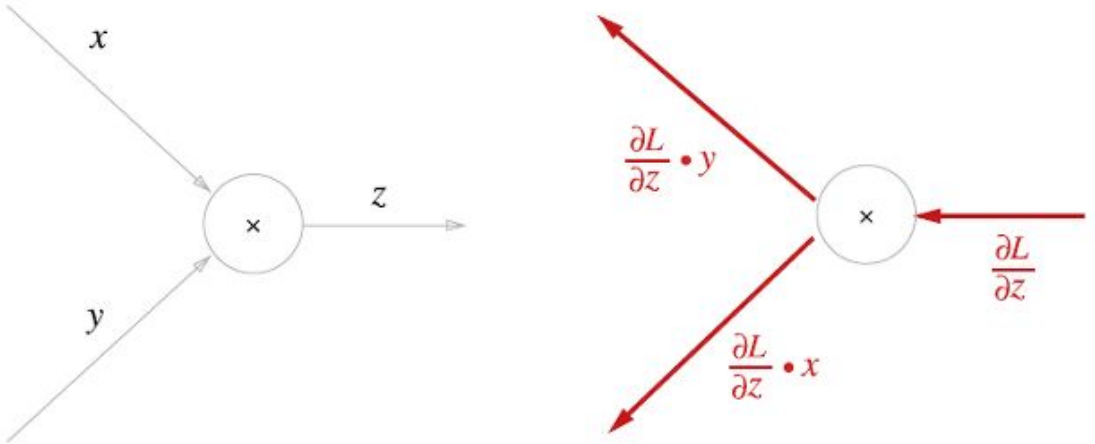
```python
#덧셈 그래프
class AddLayer:
    def __init__(self):
        pass

    def forward(self, x, y):
        out = x + y
        return out

    def backward(self, dout):
        dx = dout * 1
        dy = dout * 1
        return dx, dy
```

곱셈 그래프

$$z = xy$$

$$\frac{\partial z}{\partial x} = y$$

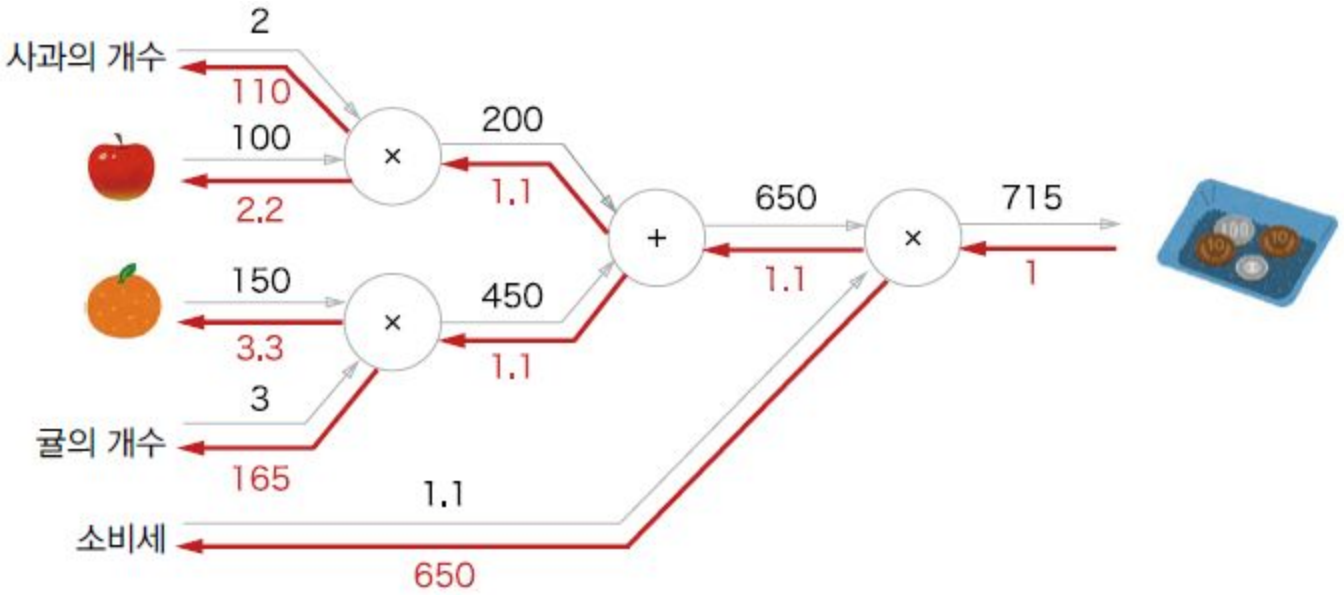
$$\frac{\partial z}{\partial y} = x$$



곱셈 그래프 코드

```python
#곱셈 그래프
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout):
        dx = dout * self.y  # x와 y를 바꾼다.
        dy = dout * self.x

        return dx, dy
```

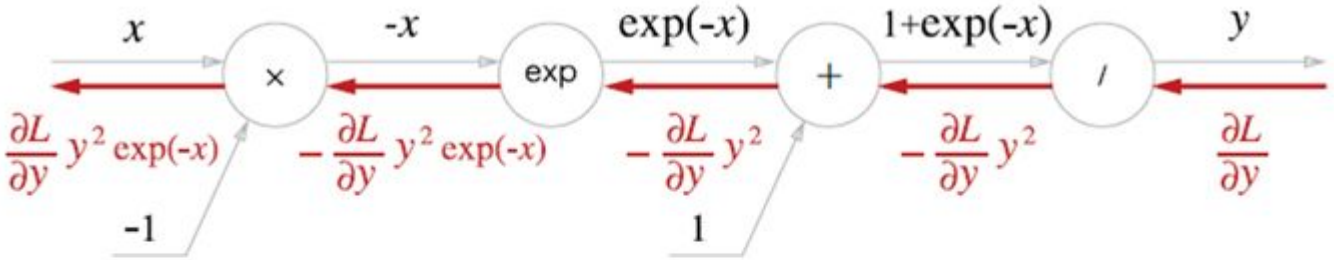
실습3

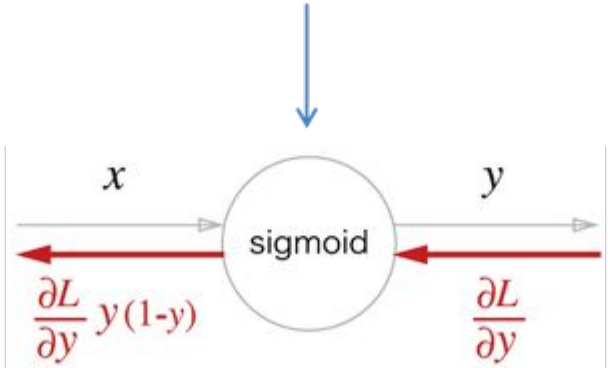

2
사과의 개수
110
100
2.2
×
200
1.1
150
3.3
×
450
1.1
3
귤의 개수
165
+
650
1.1
×
715
1
1.1
소비세
650

실습3

```
apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1

# layer
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

# forward
                                                    # (1)
                                                    # (2)
all_price = add_apple_orange_layer.forward(apple_price, orange_price)  # (3)
price = mul_tax_layer.forward(all_price, tax)  # (4)

# backward
dprice = 1
dall_price, dtax = mul_tax_layer.backward(dprice)  # (4)
dapple_price, dorange_price = add_apple_orange_layer.backward(dall_price)  # (3)
                                                    # (2)
                                                    (1)

print("price:", int(price))
print("dApple:", dapple)
print("dApple_num:", int(dapple_num))
print("dOrange:", dorange)
print("dOrange_num:", int(dorange_num))
print("dTax:", dtax)
```

Sigmoid 그래프





$$\frac{\partial L}{\partial y} y^2 exp(-x) = \frac{\partial L}{\partial y} \frac{1}{(1+exp(-x))^2} \exp(-x)$$
$$= \frac{\partial L}{\partial y} \frac{1}{1+exp(-x)} \frac{exp(-x)}{1+exp(-x)}$$
$$= \frac{\partial L}{\partial y} y(1-y)$$

Sigmoid 그래프 코드

```python
# Sigmoid 계산 그래프
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = 1 / (1 + np.exp(-x))
        self.out = out

        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx
```

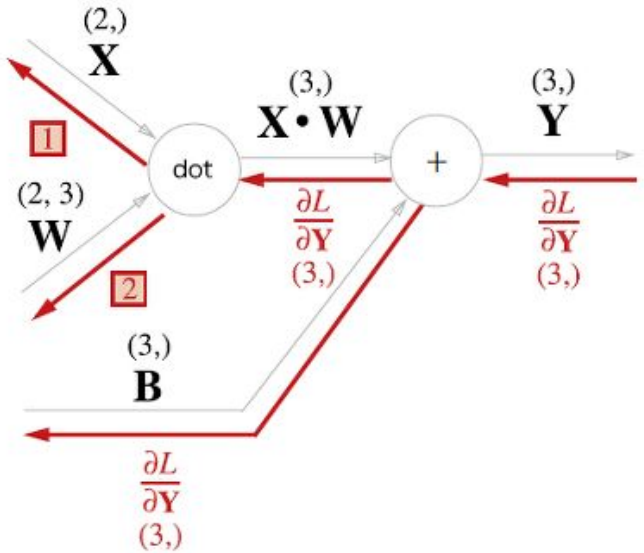
행렬곱 그래프

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^T$$

$$\frac{\partial L}{\partial \mathbf{X}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

1 $\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \mathbf{W}^T$
(2,) (3,) (3, 2)

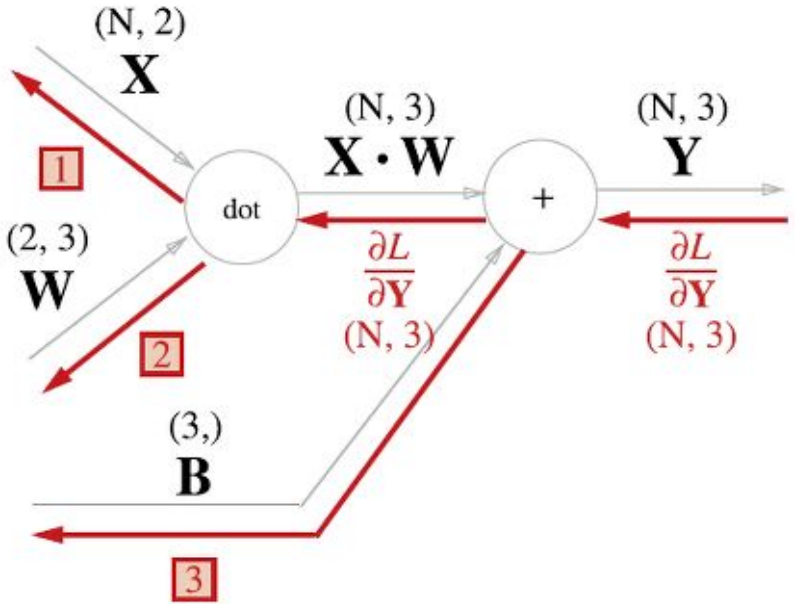
2 $\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \frac{\partial L}{\partial \mathbf{Y}}$
(2, 3) (2, 1) (1, 3)



행렬곱 그래프

1 $\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{\mathrm{T}}$
(N, 2) (N, 3) (3, 2)

2 $\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$
(2, 3) (2, N) (N, 3)



3 $\frac{\partial L}{\partial \mathbf{B}} = \frac{\partial L}{\partial \mathbf{Y}}$ 의 첫 번째 축(0축, 열방향)의 합
(3) (N, 3)

행렬곱 그래프 코드

```python
# 행렬곱 연산 계산 그래프
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self, x):
        self.x = x
        out = np.dot(x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        return dx
```

MSE 그래프 코드

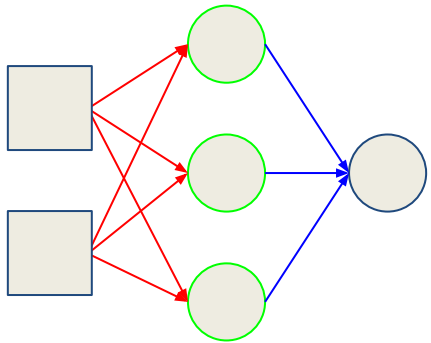
```python
class MSE:
    def __init__(self):
        self.loss = None # 손실함수
        self.y = None    # MSE 출력
        self.t = None    # 정답

    def forward(self, x, t):
        self.t = t
        self.y = x
        self.loss = 1 / 2 * np.square(self.t - self.y).sum()

        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        dx = (self.y - self.t) / batch_size
        return dx
```

실습 4

인공 신경망 학습 ver computational graph
(xor gate)

입력

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



출력

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

실습 4

TwoLayerNet class

```python
# coding: utf-8
import numpy as np
from collections import OrderedDict
import matplotlib.pyplot as plt

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = np.random.uniform(size=(input_size, hidden_size))
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = np.random.uniform(size=(hidden_size, output_size))
        self.params['b2'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Sigmoid1'] = Sigmoid()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['Sigmoid2'] = Sigmoid()
        self.lastLayer = MSE()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)
```

실습 4

TwoLayerNet class
이어서..

```
# x : 입력 데이터, t : 정답 레이블
def loss(self, x, t):
    y = self.predict(x)
    return self.lastLayer.forward(y, t)

def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    if t.ndim != 1 : t = np.argmax(t, axis=1)

    accuracy = np.sum(y == t) / float(x.shape[0])
    return accuracy

# x : 입력 데이터, t : 정답 레이블
def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

    return grads
```

실습 4

실행코드

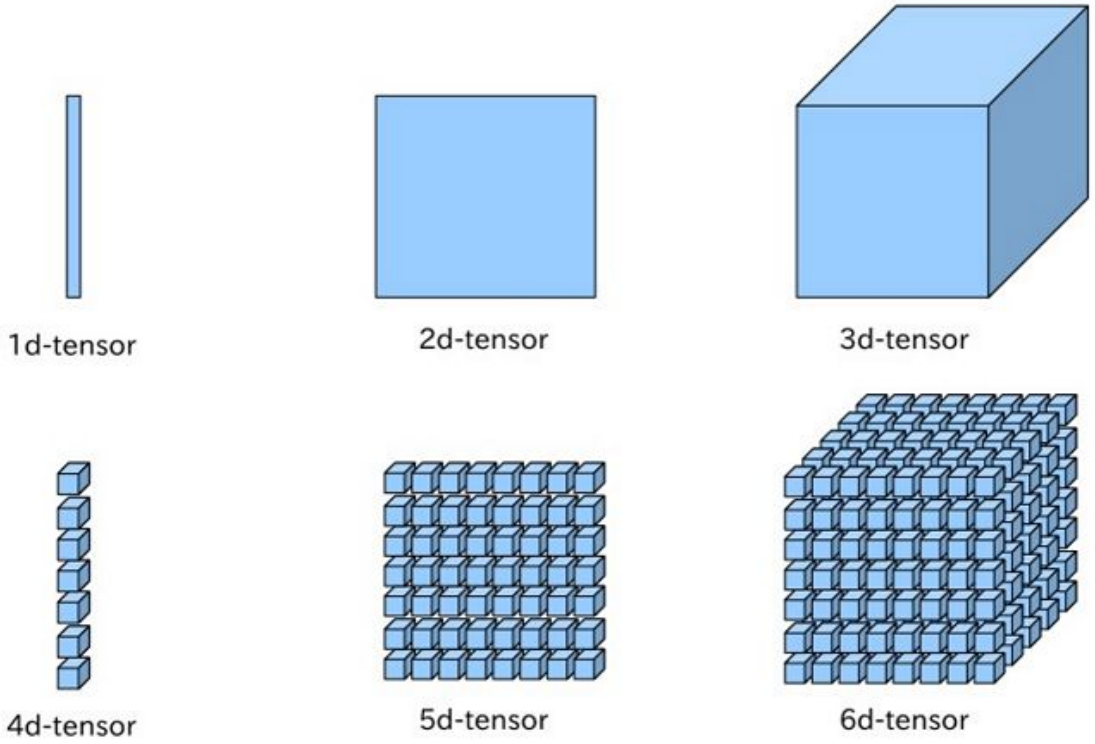
```
# 인공적으로 입력과 출력 만들기 X : 입력 t : 정답(label)
# xor 데이터입니다.
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
t = np.array([[0], [1], [1], [0]])

network = TwoLayerNet(input_size=2, hidden_size=3, output_size=1)

epoch = 20000

learning_rate = 0.1

train_loss_list = [] # 결과 출력을 위한 코드
for i in range(epoch):
    # network의 기울기 계산 메서드인 gradient 호출
    grad = 

    # 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(X, t)
    train_loss_list.append(loss) #결과 출력을 위한 코드
plt.plot(train_loss_list) # 결과 출력
```

Deep Learning Library

딥러닝 라이브러리

딥러닝 라이브러리

딥러닝 라이브러리

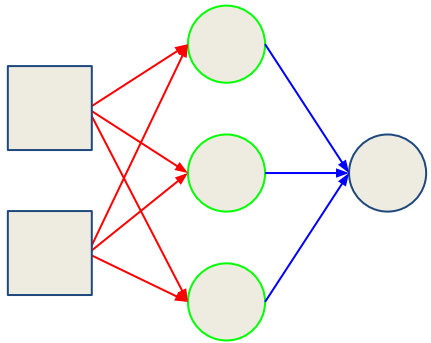


실습 5

인공 신경망 학습 ver pytorch
(xor gate)

입력

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



출력

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

실습 5

```
import torch
import torch.utils.data
import torch.nn as nn
import torch.optim as optim

# 가중치 초기화 함수
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)

# Xor data를 만들고 모델에 넣어줄 데이터로더. torch.utils.data.Dataset를 상속받고 __len__과 __getitem__을 선언해준다.
class xor_dataloader(torch.utils.data.Dataset):
    # 신경망 학습 중 신경망에 데이터를 공급해주는 dataloader 정의
    def __init__(self):
        super().__init__()
        # 외부 파일을 이용할 경우 여기서 파일을 읽거나 파일들의 경로를 찾음
        self.input = torch.Tensor([[0, 0], [0, 1], [1, 0], [1, 1]])
        self.target = torch.Tensor([[0], [1], [1], [0]])

    def __len__(self):
        # dataloder로 기능하기위해 선언 필요.
        # 데이터의 총 개수가 출력되도록 함
        return len(self.input)

    def __getitem__(self, item):
        # dataloader에 데이터를 요청하였을 때 다음 데이터를 제공.
        X = self.input[item]
        t = self.target[item]
        return X, t
```

실습 5

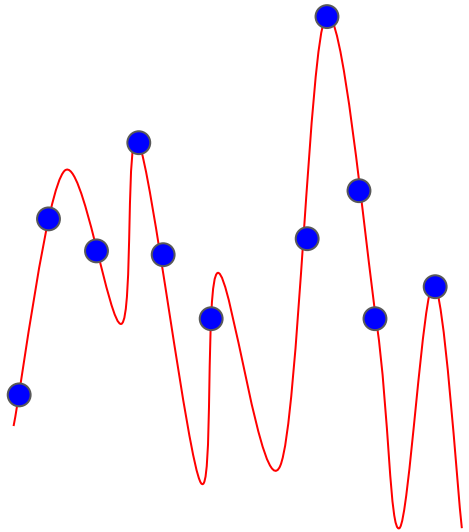
```
# pytorch의 네트워크 클래스. nn.Module을 상속받고 모델 구조를 정의한 후 forward 메서드를 정의해준다.
class TwoLayerNet_pytorch(nn.Module):
  def __init__(self, input_size, hidden_size, output_size):
    super().__init__()
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.output_size = output_size

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

실습 5

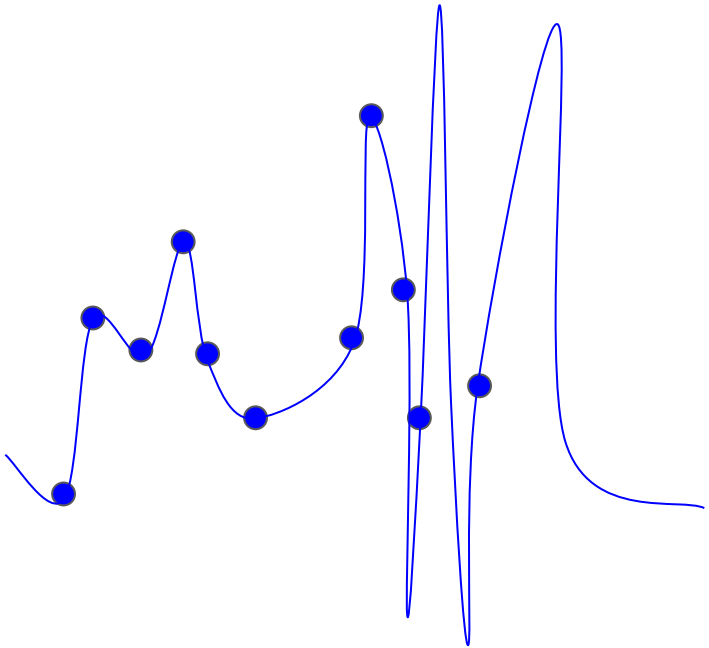
```
# hyperparameter 선언
epochs = 20000
learning_rate = 0.1

net = TwoLayerNet_pytorch(input_size=2, hidden_size=3, output_size=1) # 모델을 인스턴스로 만듦.
net.apply(weight_init) # 만들어진 모델 인스턴스에 weight 초기화 함수를 수행
optimizer = optim.SGD(net.parameters(), lr=learning_rate) # optimizer 선언.
dataloader = torch.utils.data.DataLoader(xor_dataloader(), batch_size=4) # dataloader 선언.

MSE = nn.MSELoss() # loss function 선언.

train_loss_list = [] # 결과 출력을 위한 코드
for epoch in range(epochs):
    for i, (X, t) in enumerate(dataloader,0):
        # 순전파
        Y = net(X)
        loss = MSE(Y, t)

        train_loss_list.append(loss) # 결과 출력을 위한 코드

        optimizer.zero_grad()  # optimizer의 gradient 텐서들을 초기화
        loss.backward() # backpropagation 수행. 각 weight에 대한 gradient 계산
        optimizer.step() # weight 갱신

# 학습종료 후 결과물 확인 (optional)
print("Input is")
print(X)
print("expected output is")
print(t)
print("actual output is ")
print(Y)
```

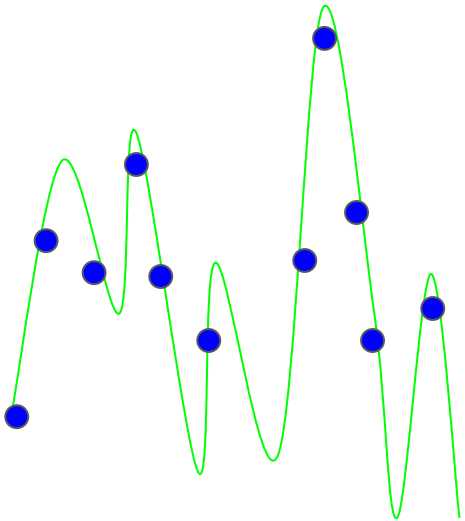
Deep Learning Project

Generalization



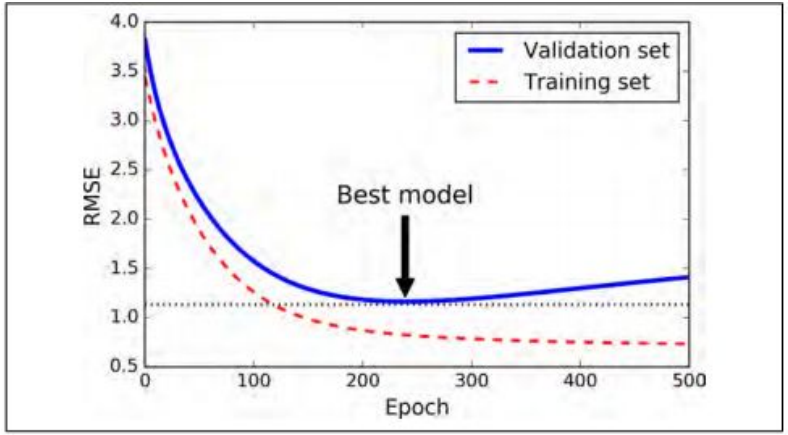
Overfitting problem



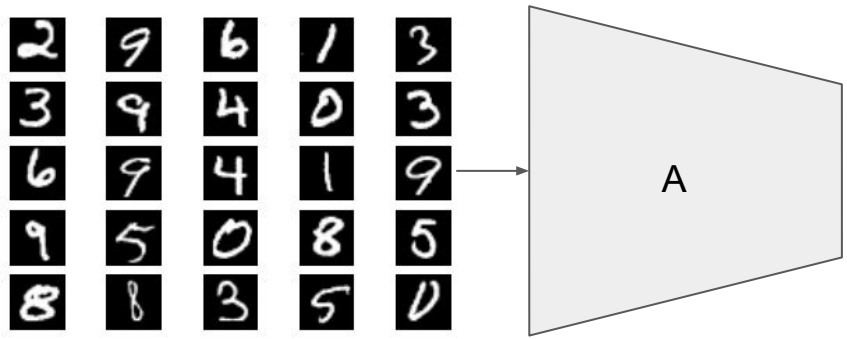


dataset 분리

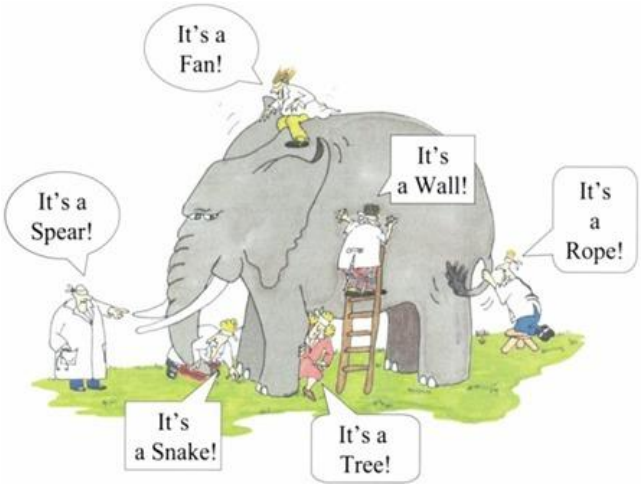

Train
Validation
Test


Validation set
Training set
Best model
RMSE
Epoch
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
100
200
300
400
500

데이터 로딩

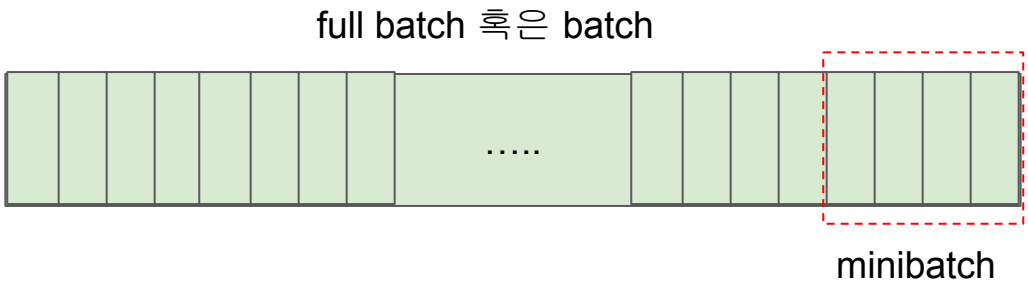


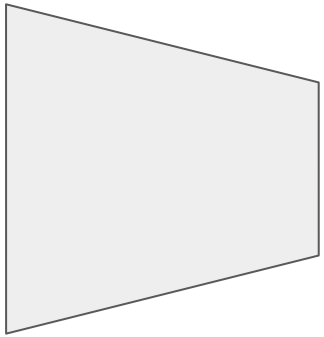
Stochastic Gradient Descent



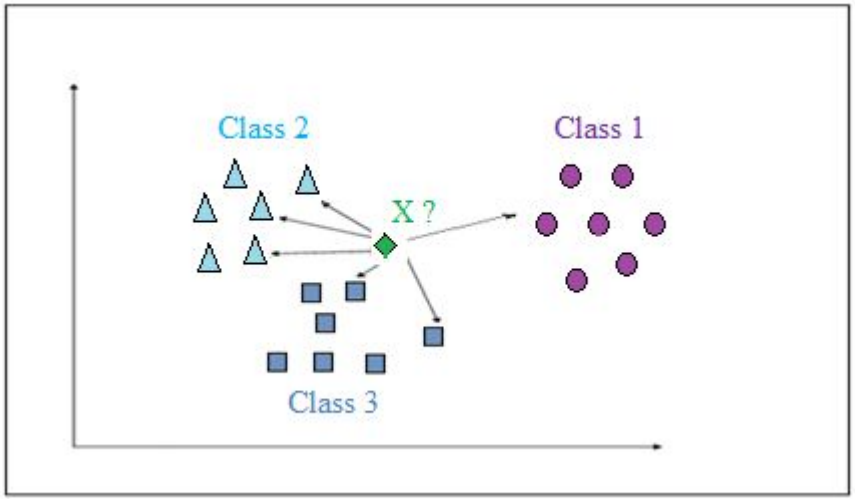
Stochastic Gradient Descent

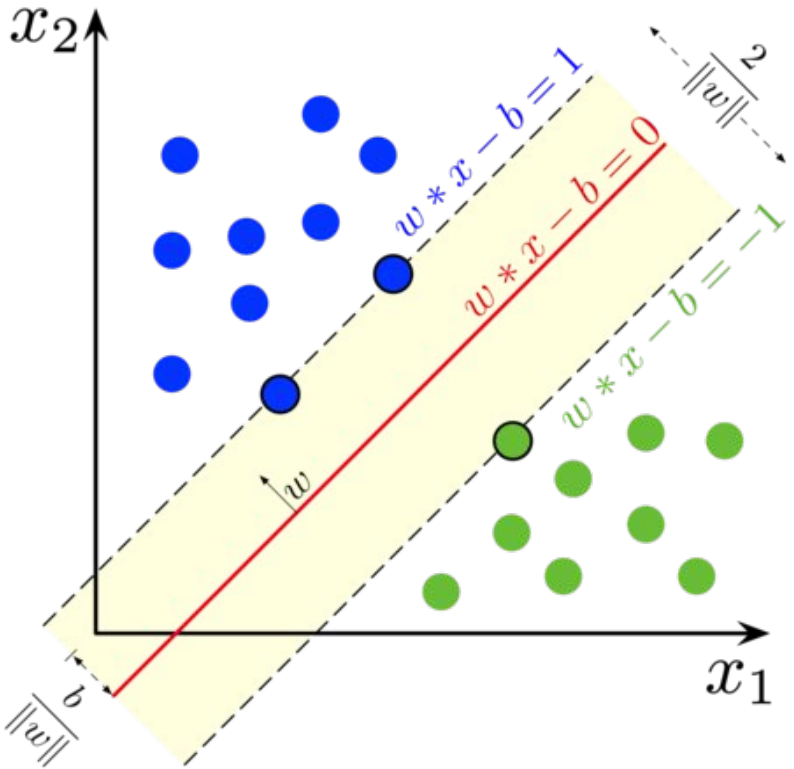
$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$$





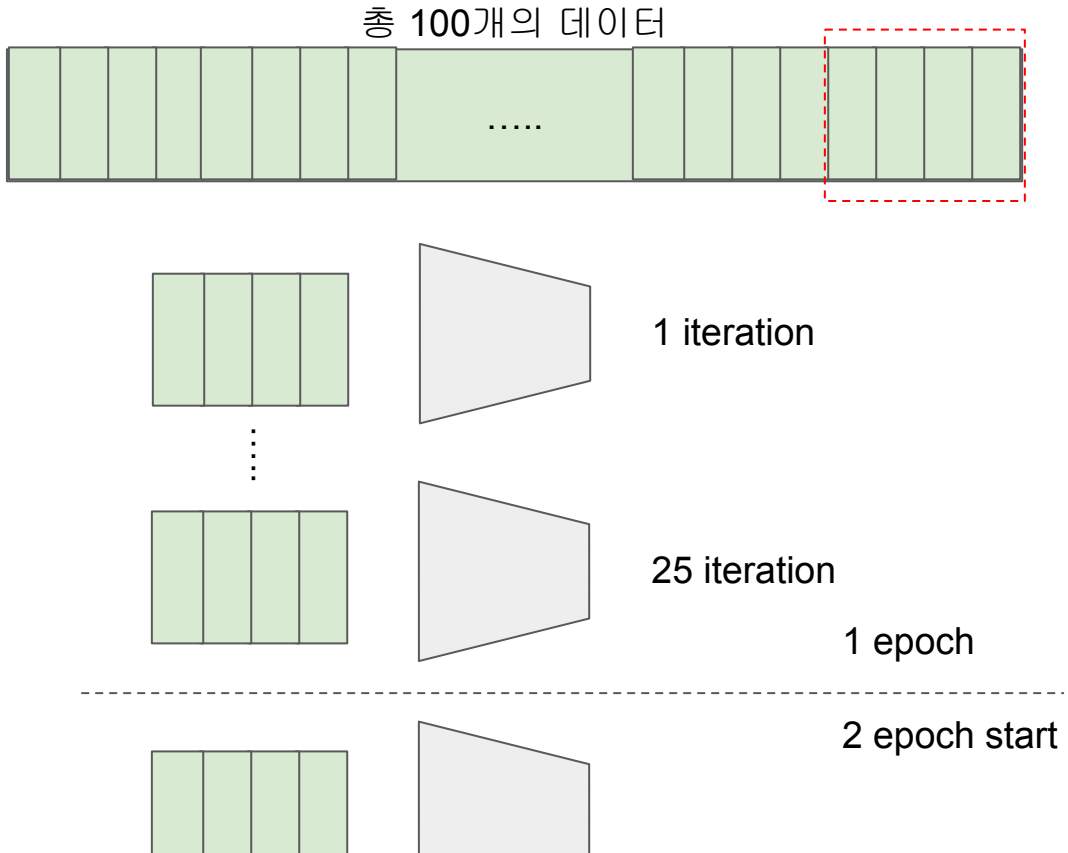
SGD가 중요한 이유?



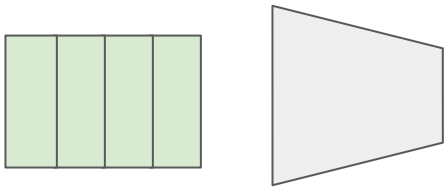


Epoch, Iteration

batch size = 4 일 경우

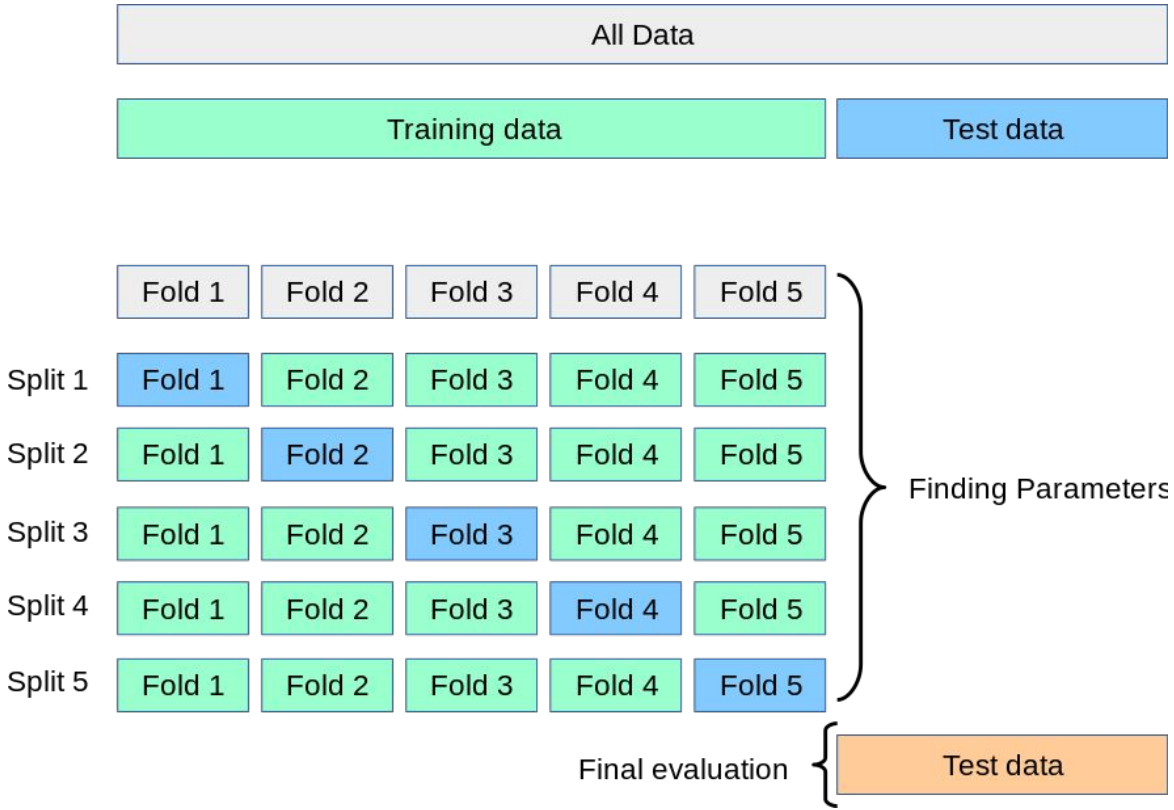


Training

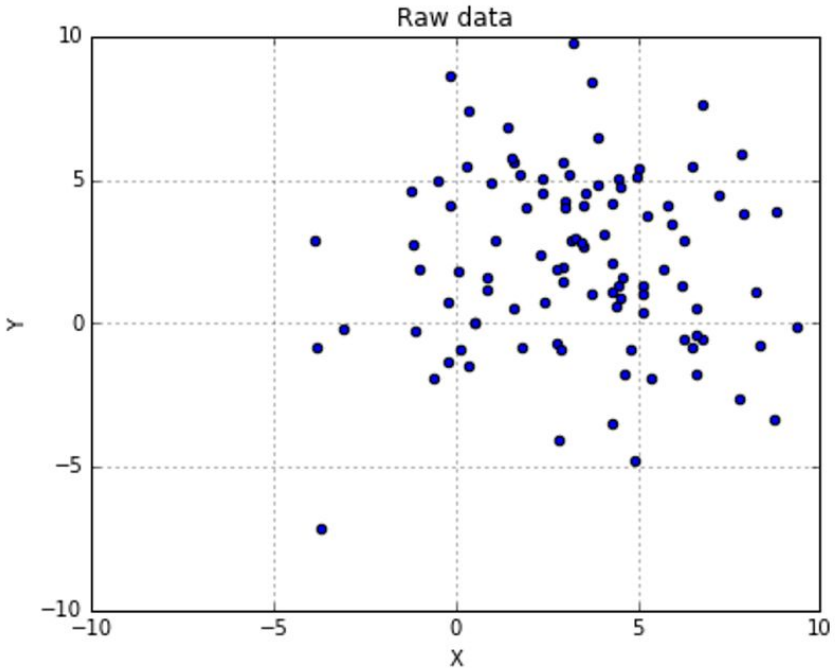


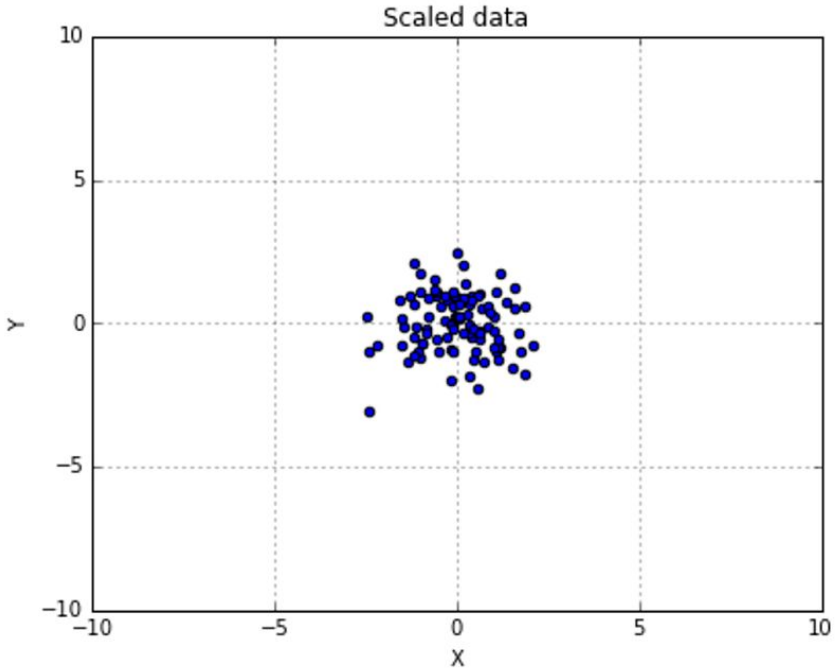
1 iteration = 1 weight update = 1 step training

Cross Validation

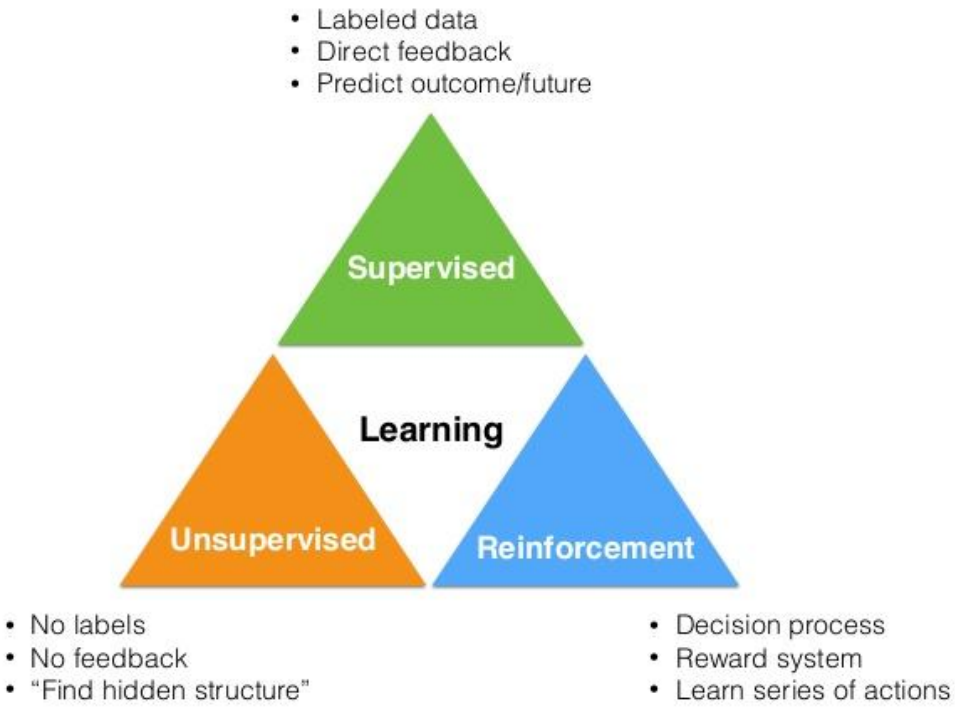


Data Preprocessing

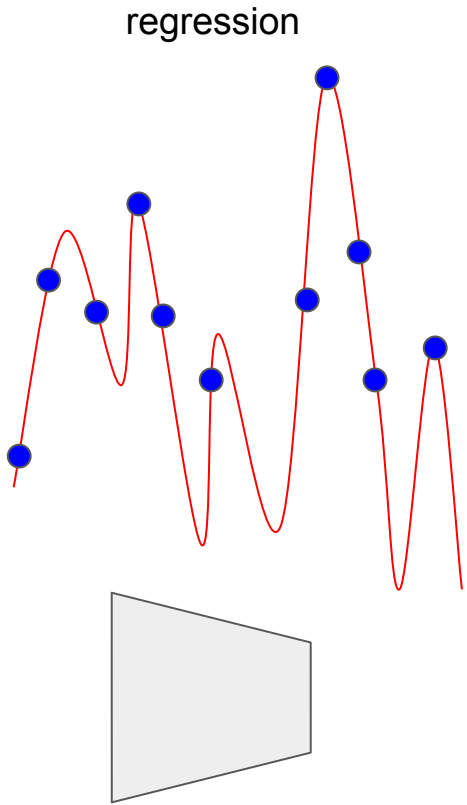


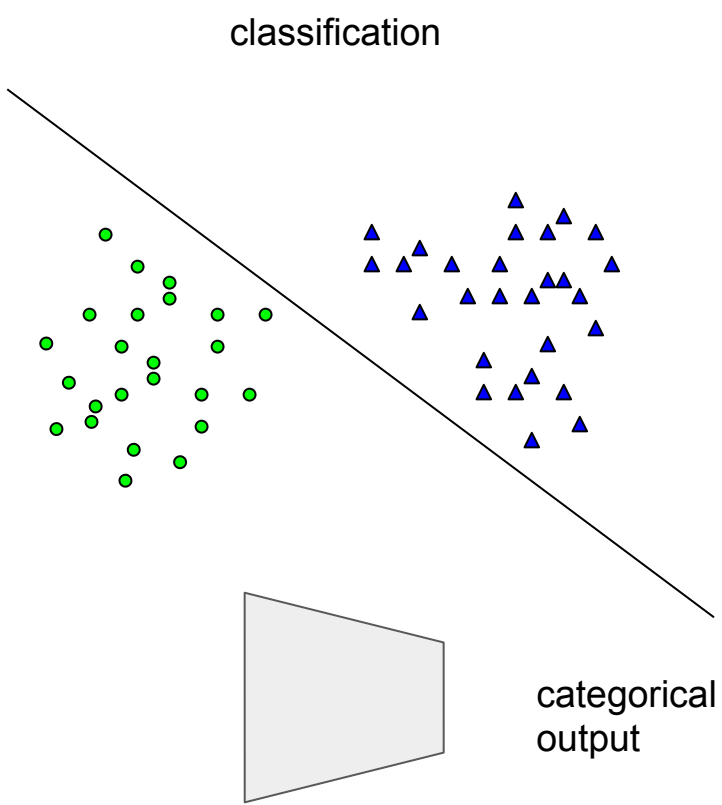


학습의 분류

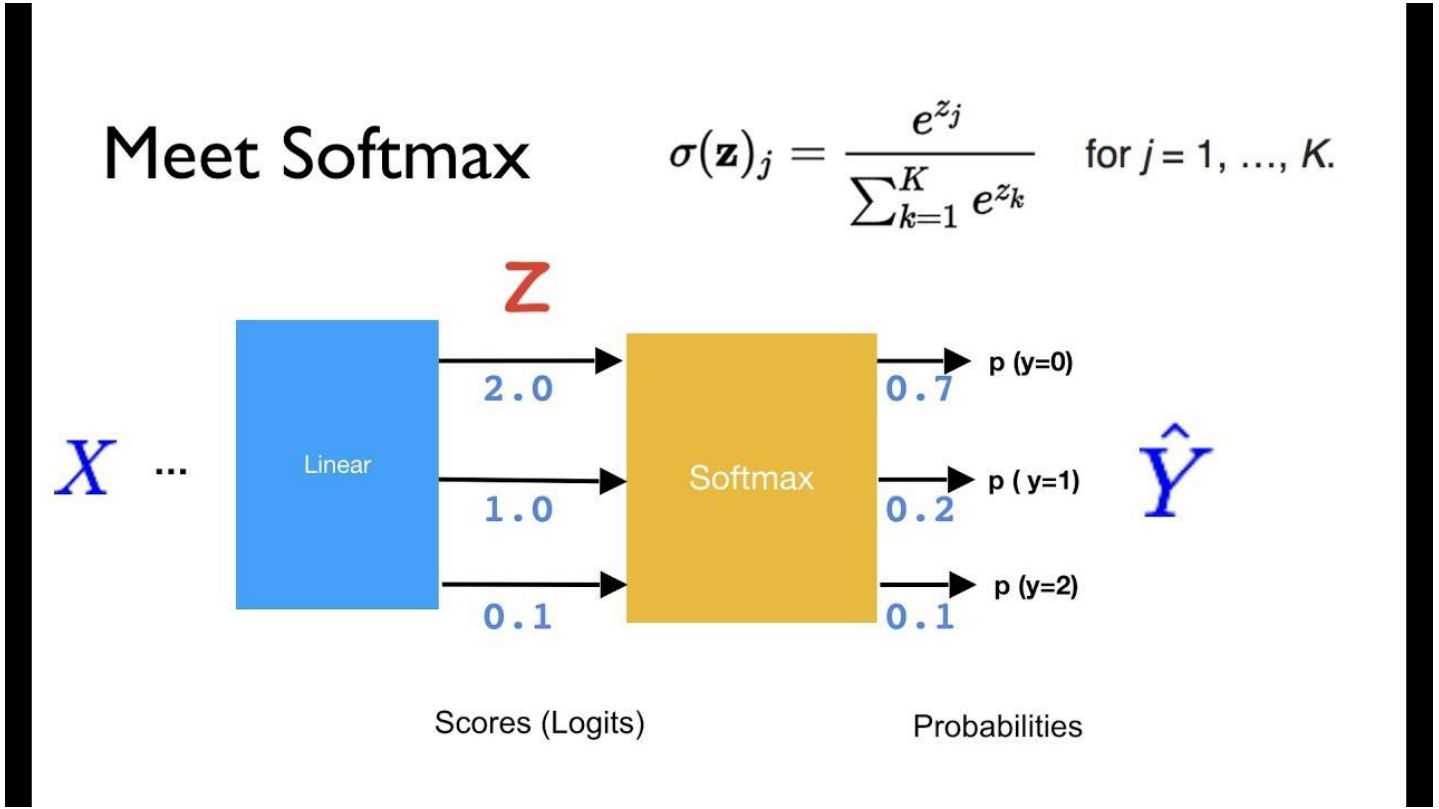

• Labeled data
• Direct feedback
• Predict outcome/future
Supervised
Learning
Unsupervised
Reinforcement
• No labels
• No feedback
• “Find hidden structure”
• Decision process
• Reward system
• Learn series of actions

Classification Problem



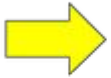


Softmax function



One Hot encoding

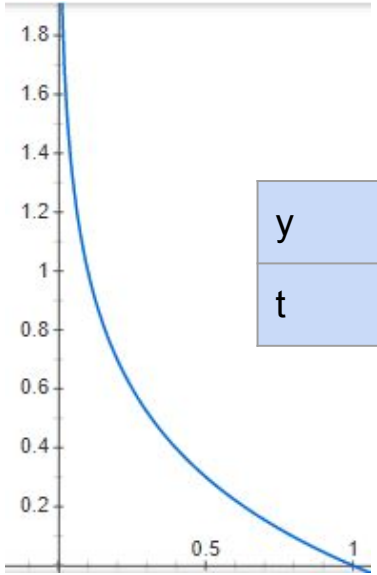
Color
Red
Red
Yellow
Green
Yellow



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

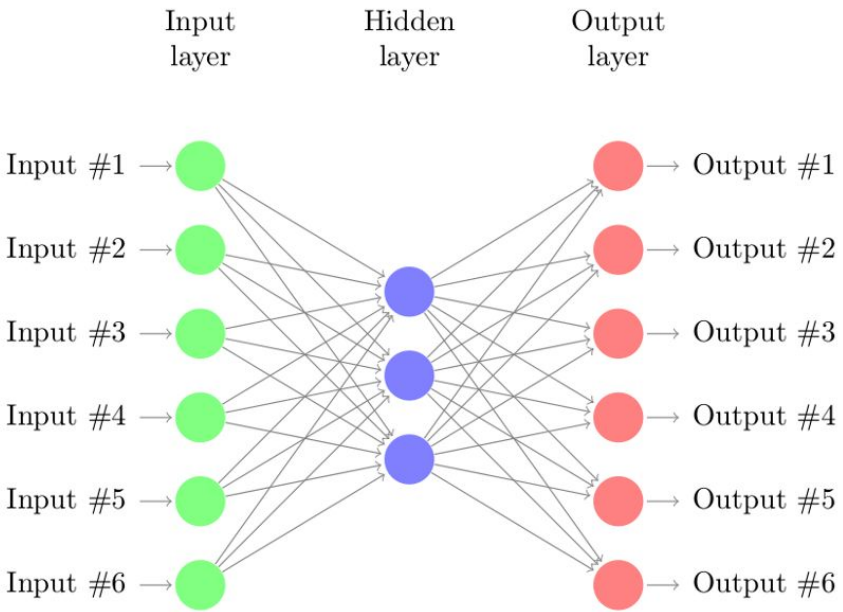
Cross Entropy loss function

$$-\sum_{k} t_k \log y_k$$

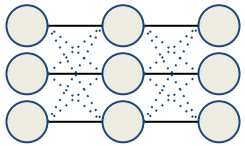


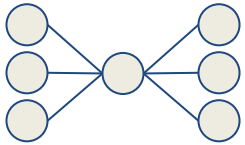
y	0.1	0.6	0.1	0.1	0.1
t	0	1	0	0	0

오토인코더

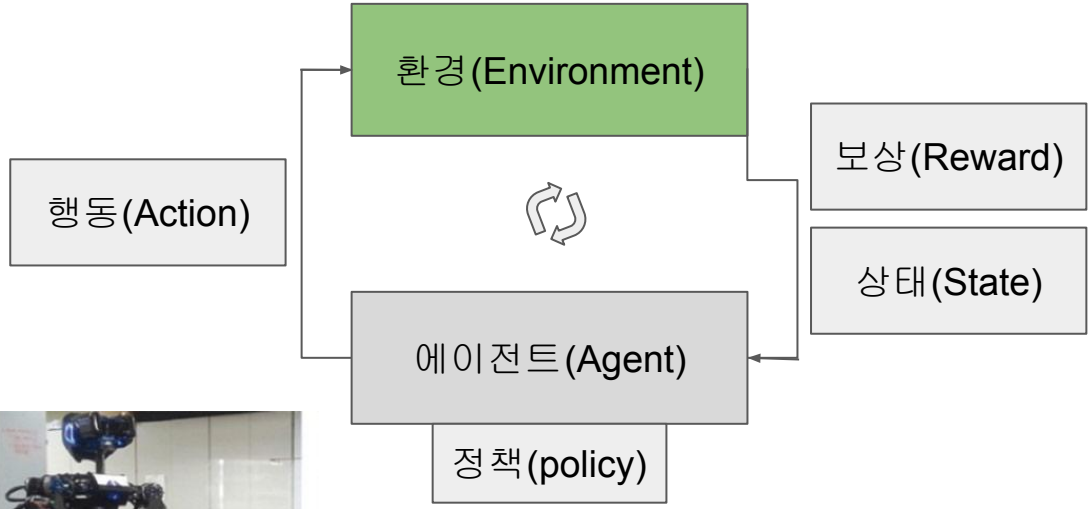

Input layer
Hidden layer
Output layer
Input #1
Input #2
Input #3
Input #4
Input #5
Input #6
Output #1
Output #2
Output #3
Output #4
Output #5
Output #6

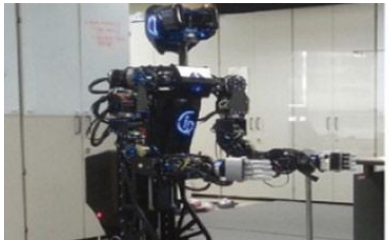
오토인코더





강화학습


환경(Environment)
보상(Reward)
행동(Action)
상태(State)
에이전트(Agent)
정책(policy)



실습 6

MNIST Classifier 구현 ver pytorch

28x28 크기의
숫자 손글씨
이미지 데이터

입력

딥러닝 신경망

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

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

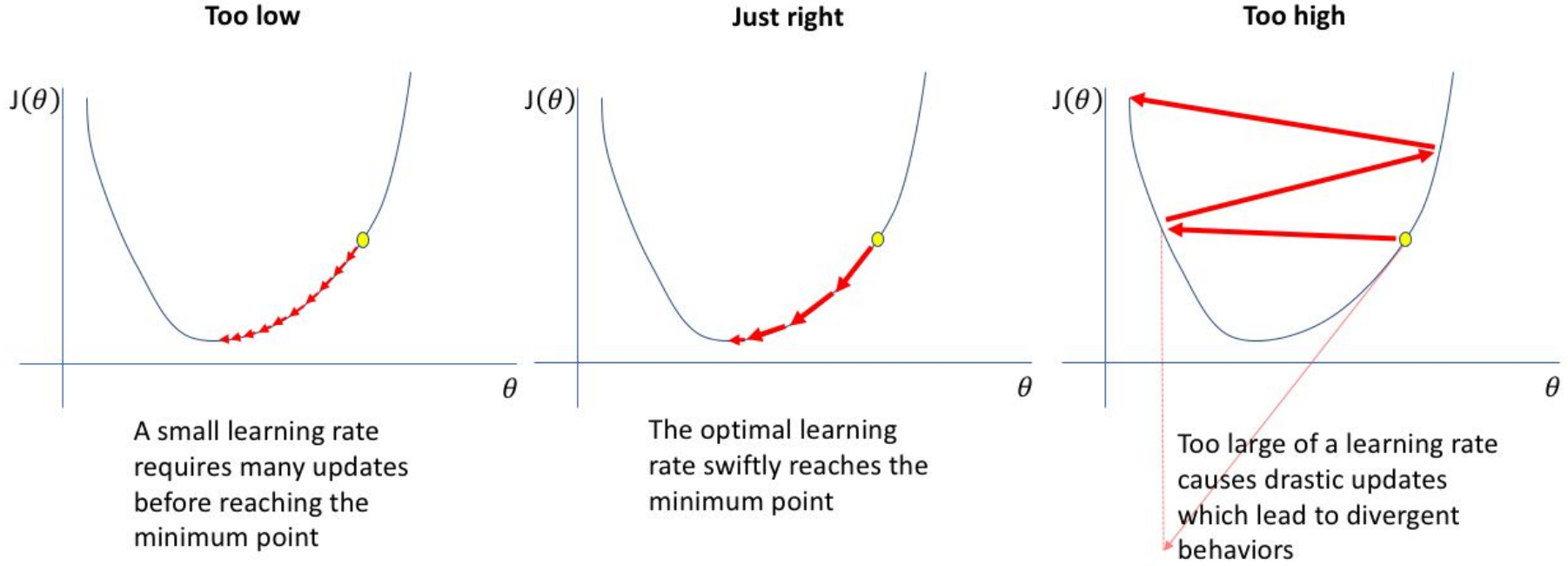
hyperparmater
선언과 학습

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

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))
```

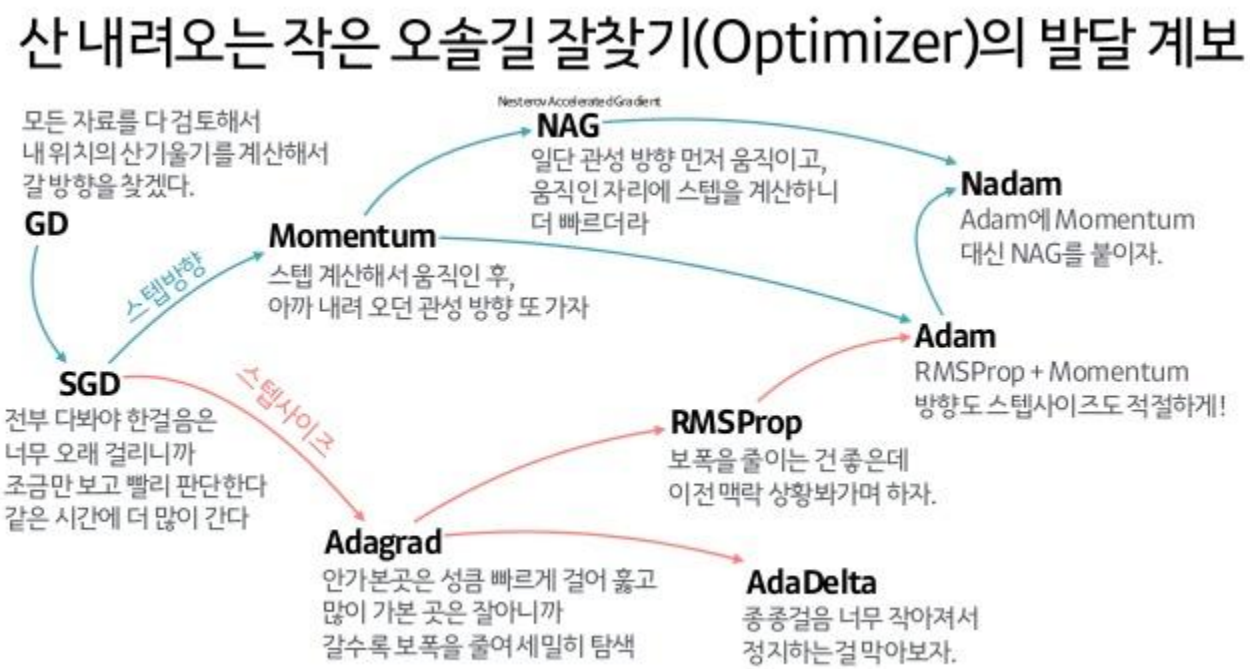
실습 6

```
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)
```

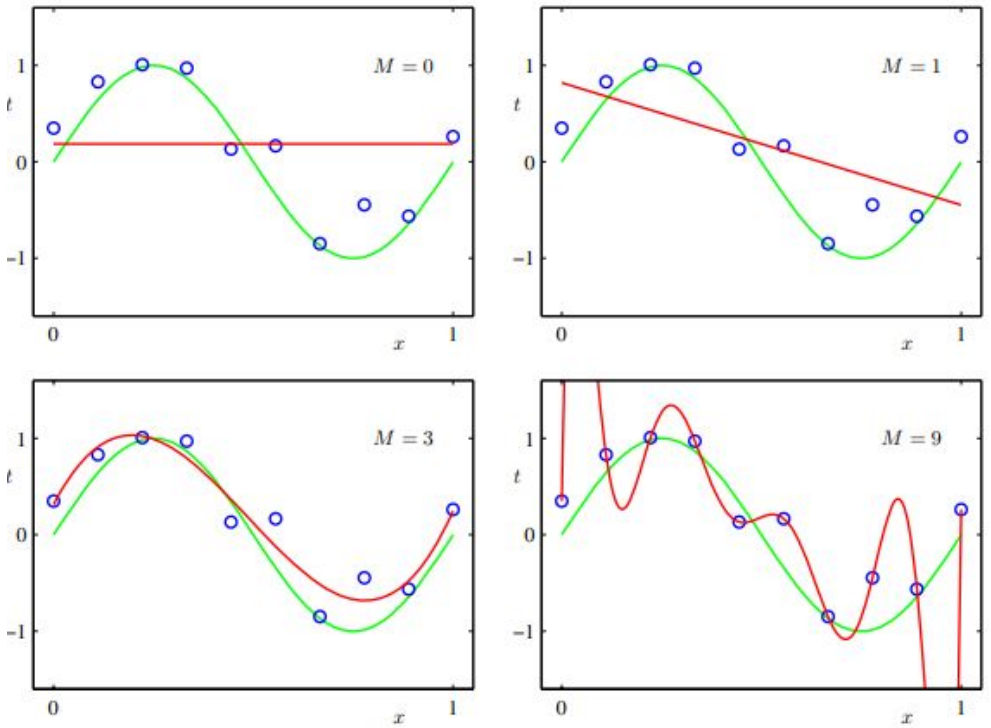
Learning rate



optimizers


산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보
모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
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
스텝사이즈
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.

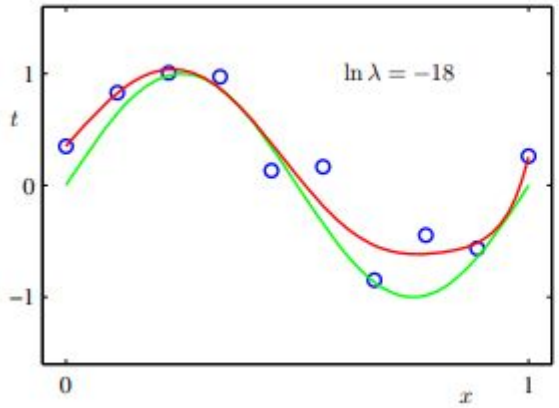
weight decay (L2 normalization)

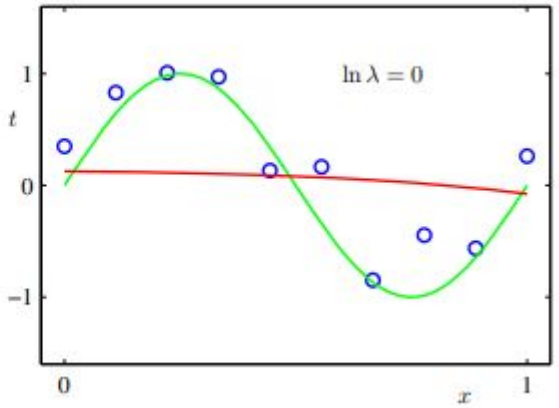


	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

weight decay (L2 normalization)

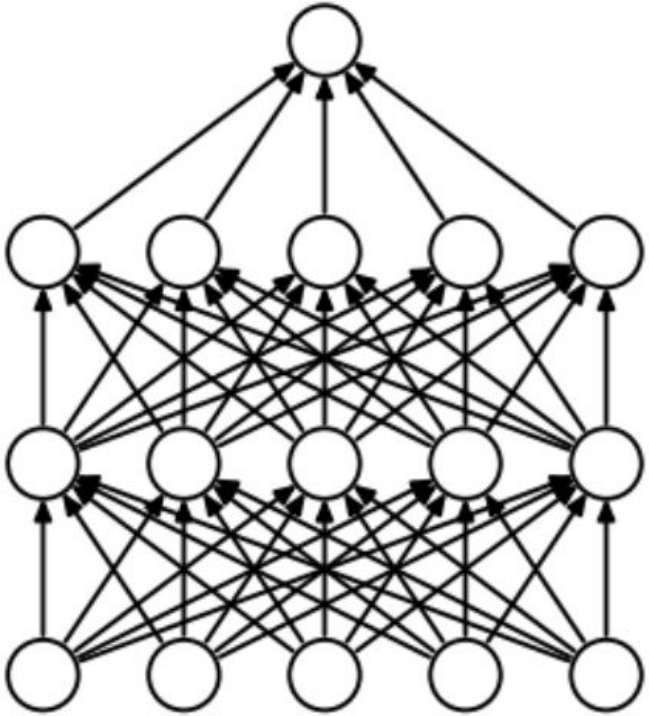
$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



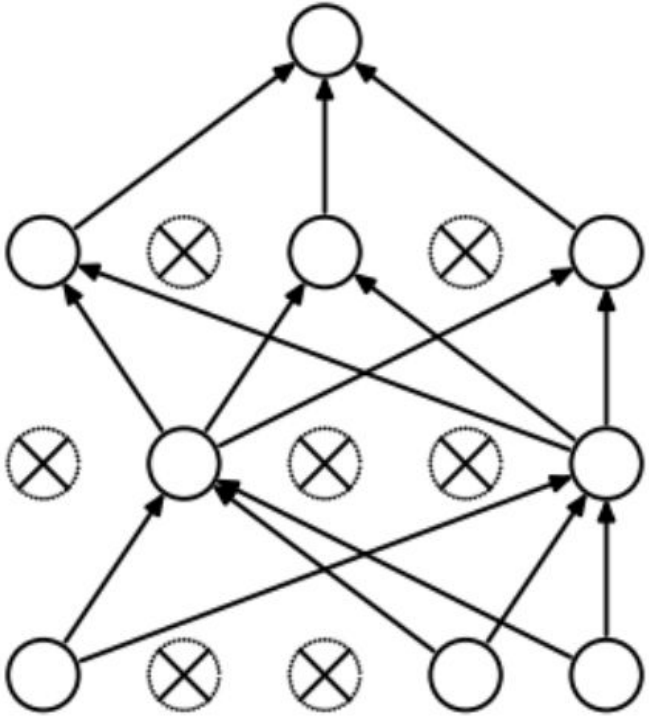


	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18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

Dropout

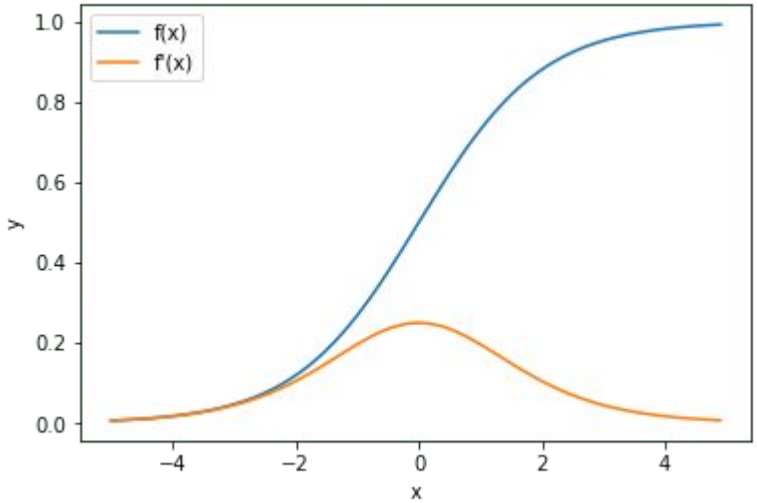


(a) Standard Neural Net



(b) After applying dropout.

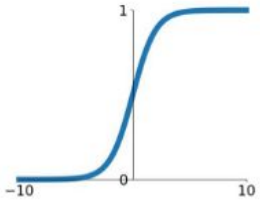
Vanishing Gradient Problem



Activation Functions

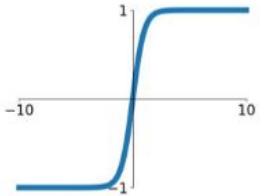
Sigmoid

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



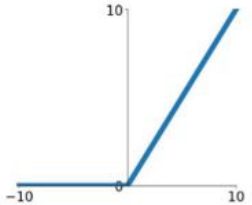
tanh

$$\tanh(x)$$



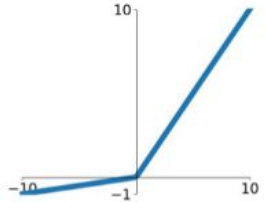
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

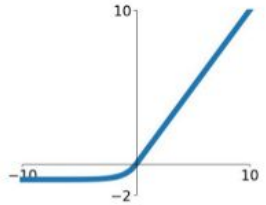


Maxout

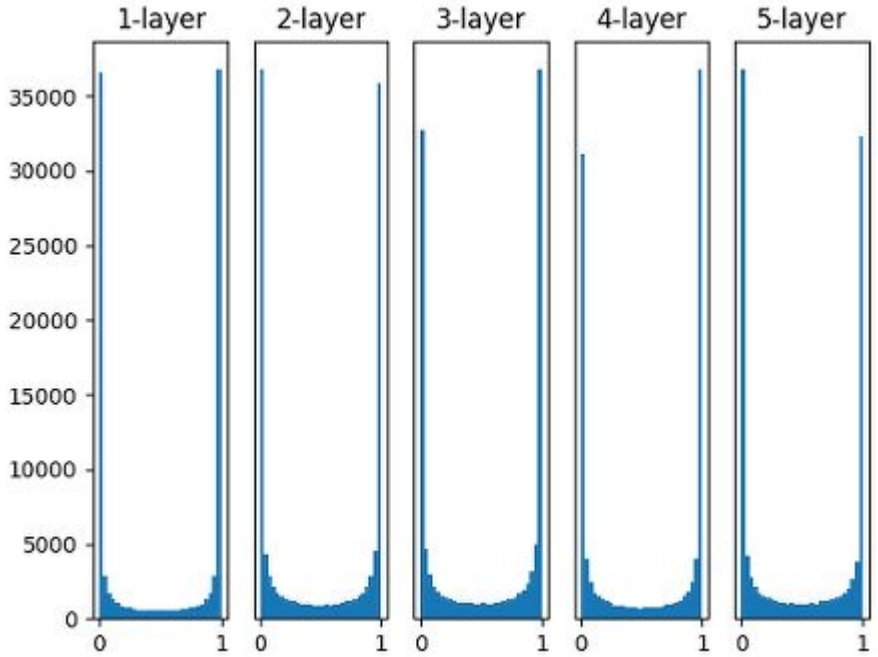
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

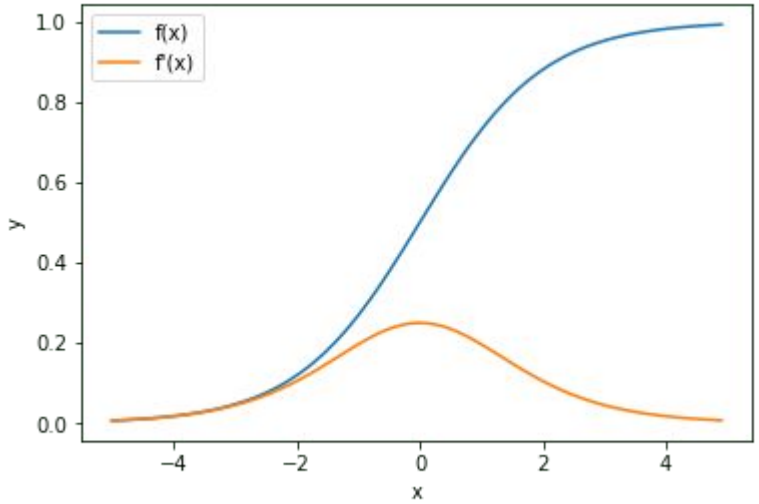
ELU

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

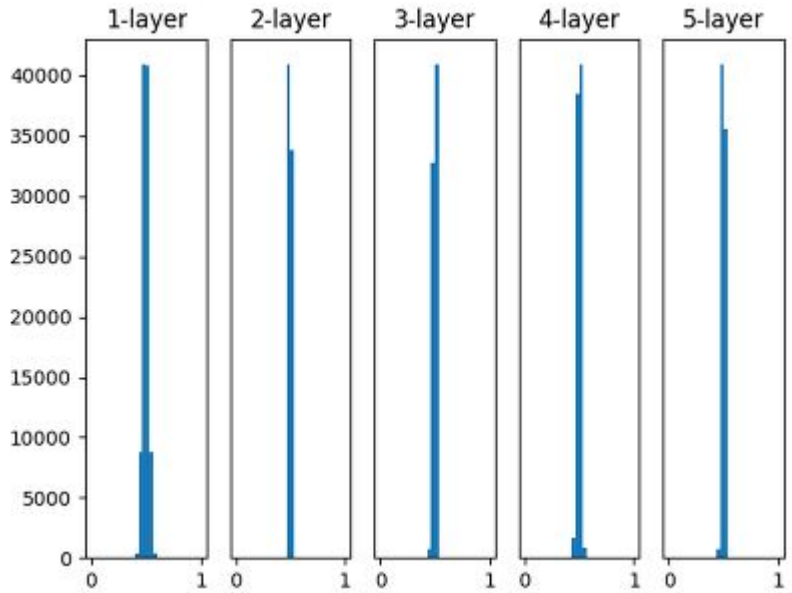


Weight Initialization

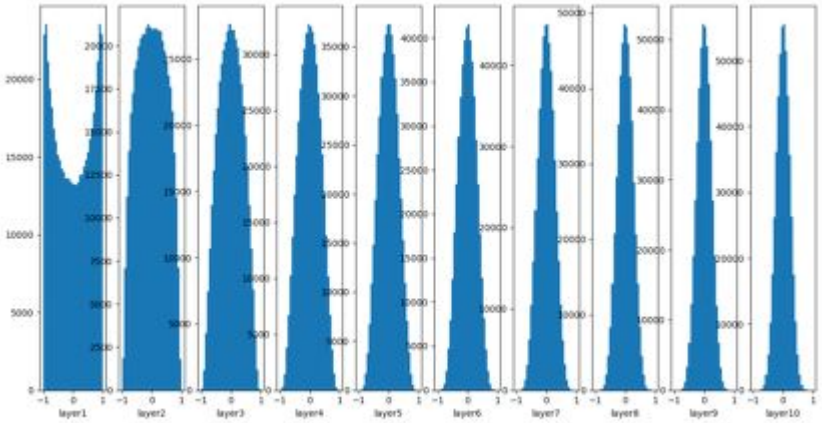




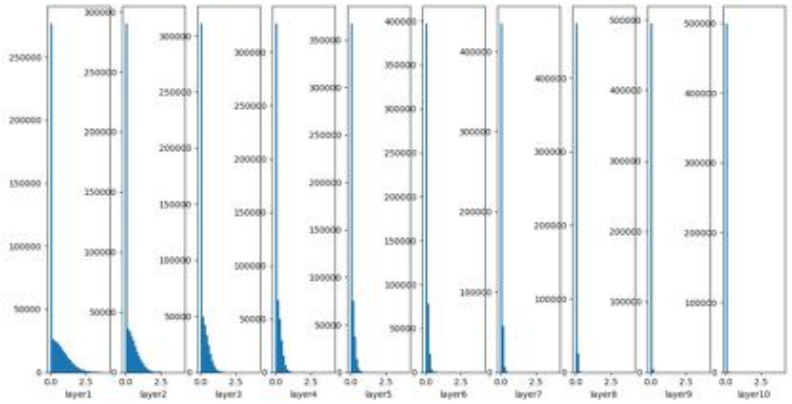
Weight Initialization - small std



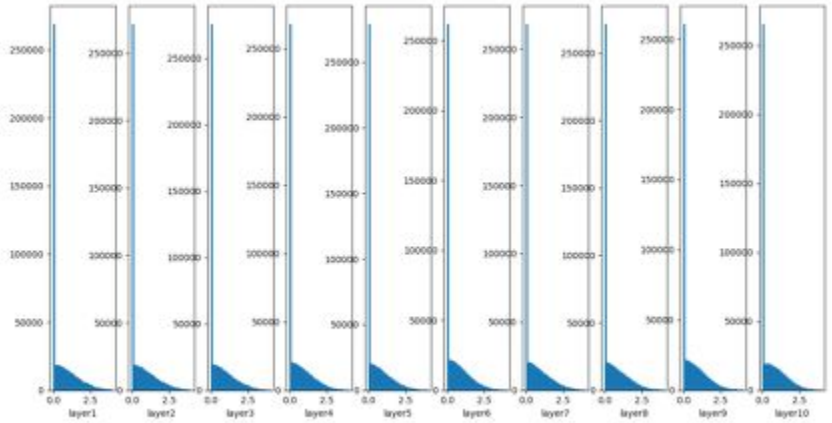
Weight Initialization - Xavier Initialization



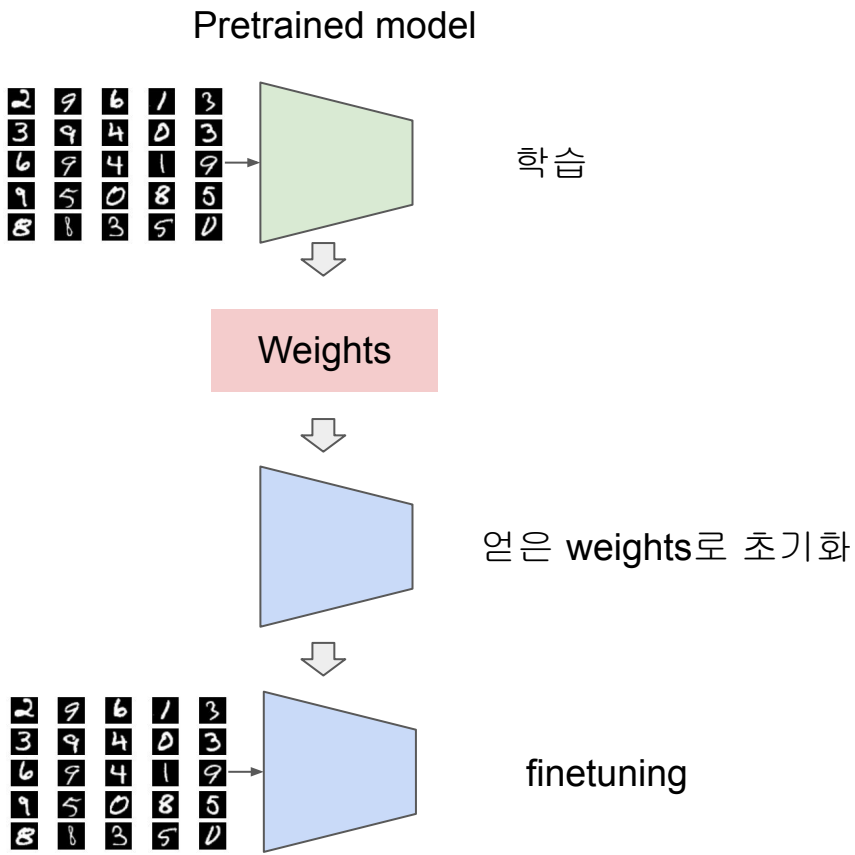
Weight Initialization - Xavier + Relu



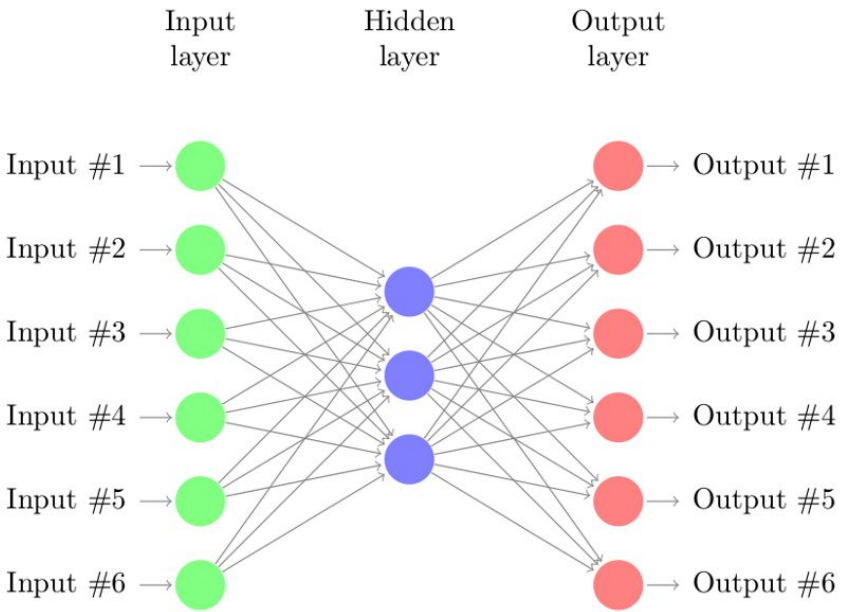
Weight Initialization - He Initialization + Relu



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

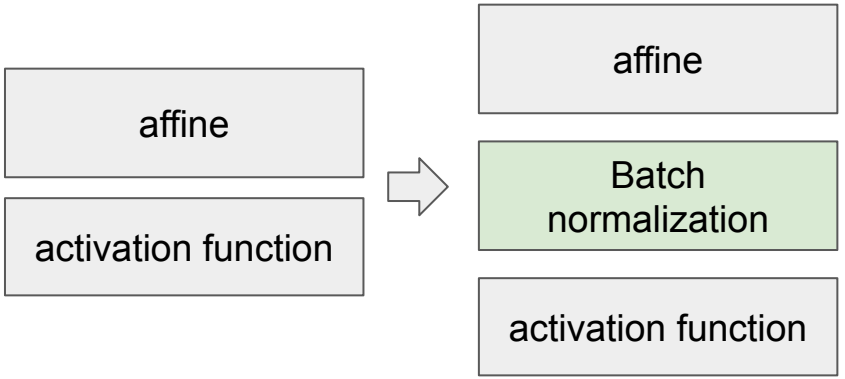
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

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



학습 속도 개선
초깃값 의존도 감소
오버피팅 억제

실습 6

MNIST Classifier 개선하기

입력

딥러닝 신경망

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

MNIST Classifier 개선하기

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

검색해보셔도 좋습니다.

실습 6

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())
```

다음 시간

Validation 실습
이미지 데이터
Convolutional Neural Network 이론
CNN 구현하기
CNN 프로젝트