MNIST

Dohyun Kim
dhkim1028@korea.ac.kr
Data Intelligence Lab, Korea University
2020.04.06.

Class Lab – 기초 과제 일정

1. XOR (~4/05)

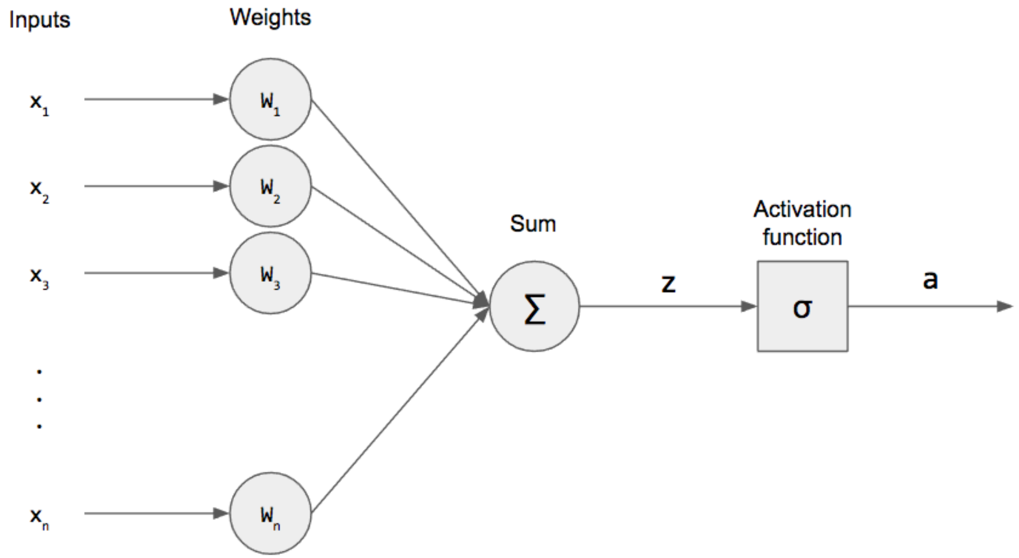
2. MNIST (~4/19)

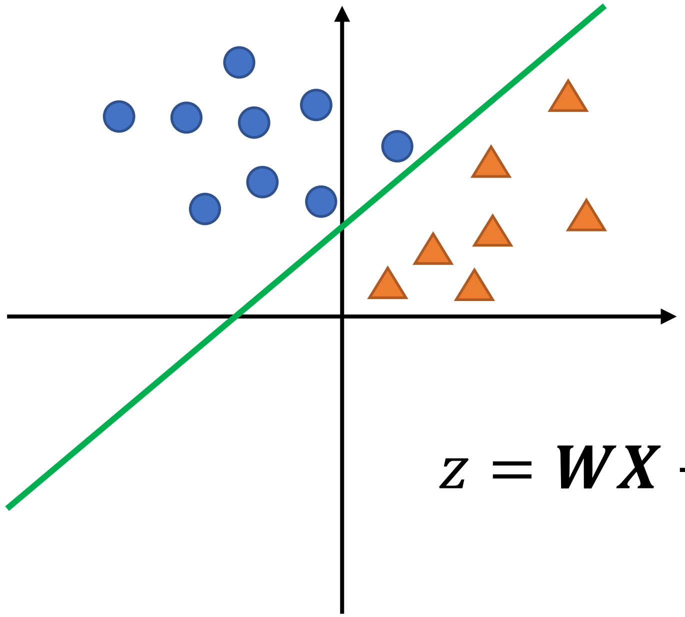
3. CIFAR10 (~5/03)

XOR Review

- **Perceptron**

Perceptron can deal with linearly separable problems.

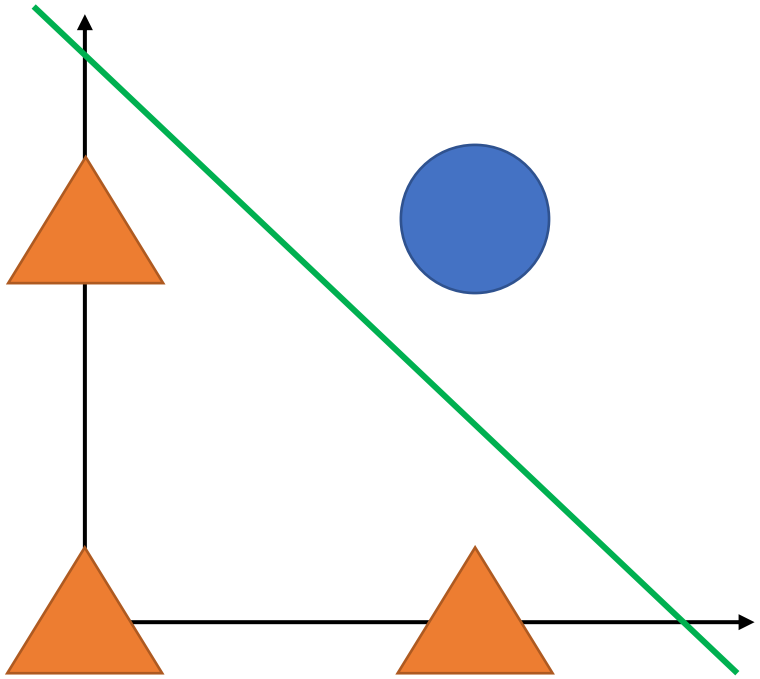




$$z = \boldsymbol{W}\boldsymbol{X} + w_0$$

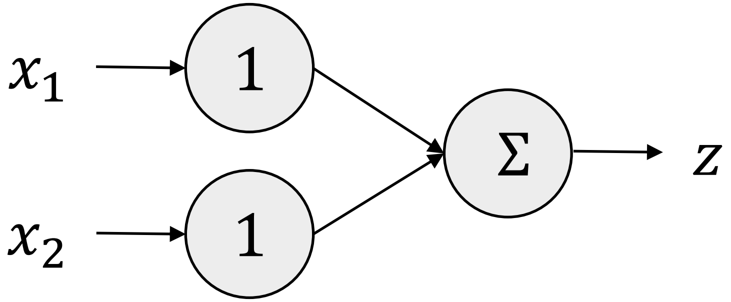
XOR Review

- **Single-layer Perceptron**



- AND Problem

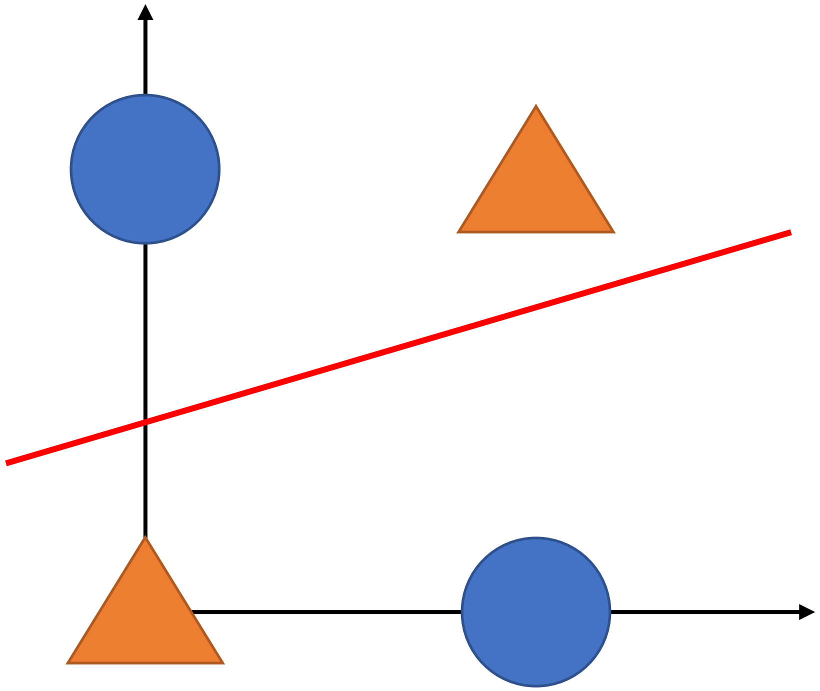
$$\boldsymbol{W} = [1 \quad 1] \qquad \boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad b = -1.5$$



- $z = x_1 + x_2 - 1.5$

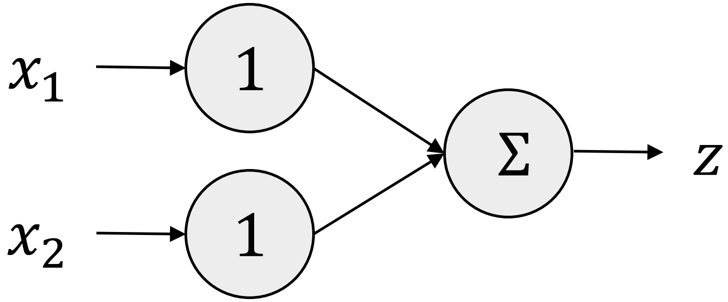
XOR Review

- **Problem of perceptron : cannot solve XOR.**



- AND Problem

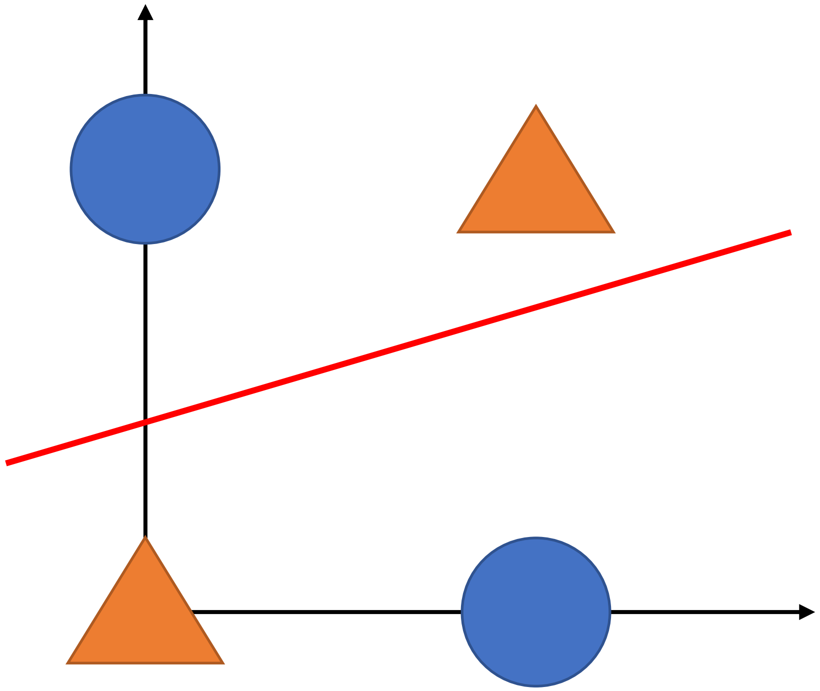
$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad b = -1.5$$



- $z = x_1 + x_2 - 1.5$

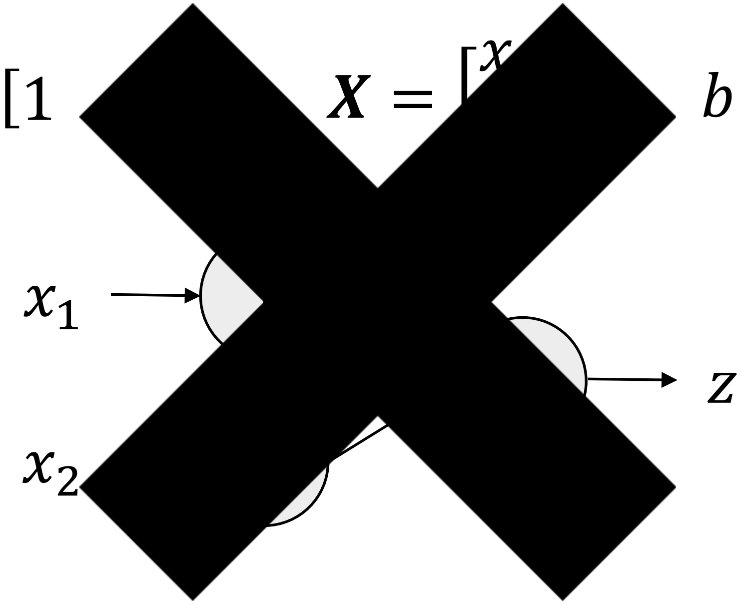
XOR Review

- **Problem of perceptron : cannot solve XOR.**



- AND Problem

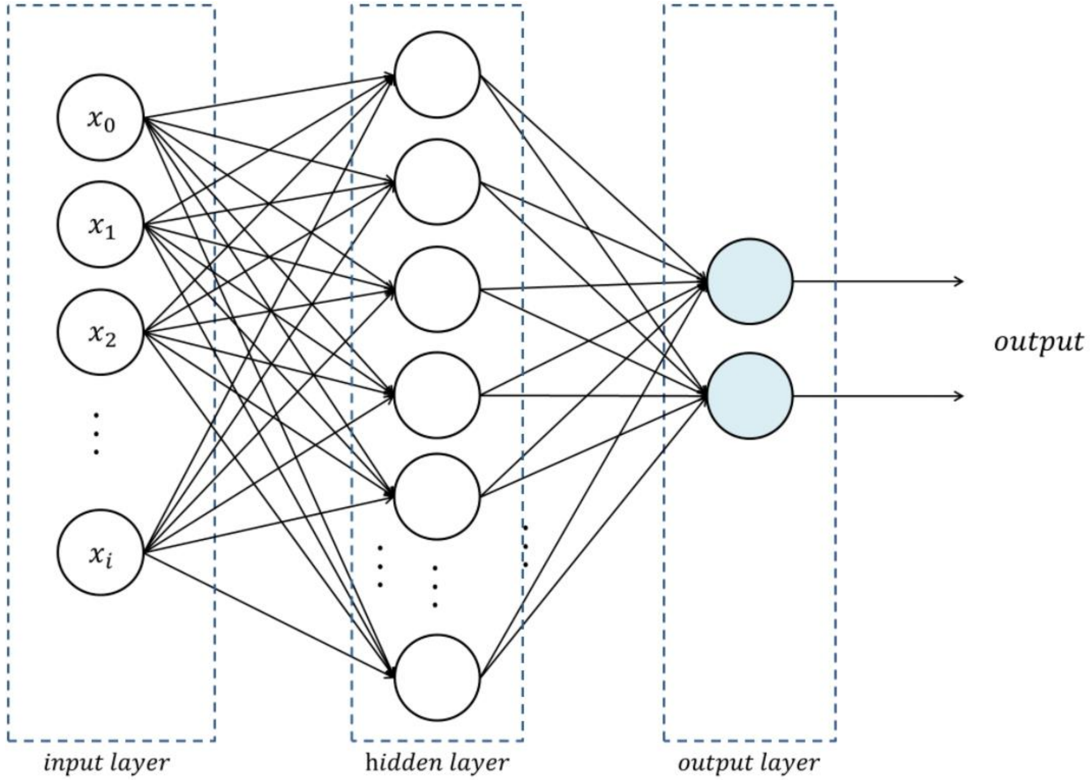
$\boldsymbol{W} = [1$ [redacted] $\boldsymbol{X} = [$ [redacted] $b = -1.5$



- $z = x_1 + x_2 - 1.5$

XOR Review

- **Multi-layer Perceptron**



X	**h**	**Y**
Size : i	**Size : j**	**Size : k**

- Stack of perceptron

$$\boldsymbol{h} = \sigma\left(\boldsymbol{W}^{(\mathbf{1})}\boldsymbol{X} + \boldsymbol{b}^{(\mathbf{1})}\right)$$
$$\boldsymbol{Y} = \sigma\left(\boldsymbol{W}^{(\mathbf{2})}\boldsymbol{h} + \boldsymbol{b}^{(\mathbf{2})}\right)$$

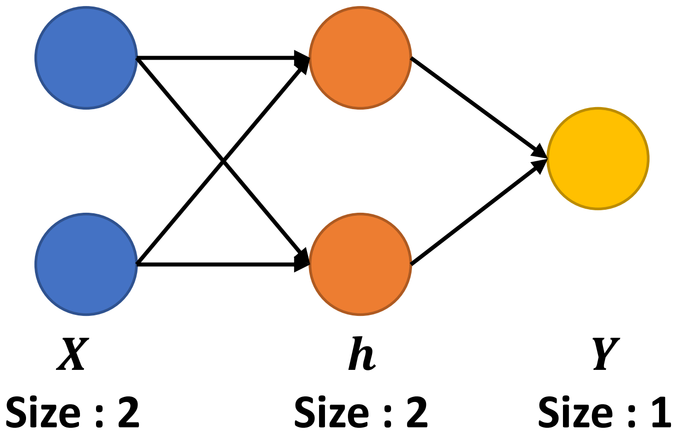
$$\boldsymbol{W}^{(\mathbf{1})} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \cdots & w_{1i}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \cdots & w_{2i}^{(1)} \\ & & \cdots & \\ w_{j1}^{(1)} & w_{j2}^{(1)} & \cdots & w_{ji}^{(1)} \end{bmatrix}$$

$$\boldsymbol{W}^{(\mathbf{2})} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & \cdots & w_{1j}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & \cdots & w_{2j}^{(2)} \\ & & \cdots & \\ w_{k1}^{(2)} & w_{k2}^{(2)} & \cdots & w_{kj}^{(2)} \end{bmatrix}$$

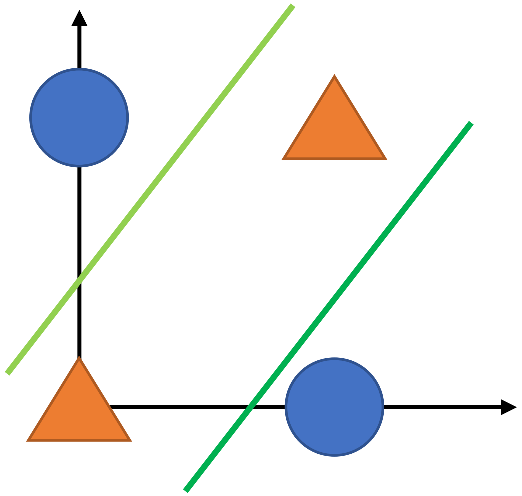
XOR Review

- **Multi-layer Perceptron**

$$h_1 = \sigma(x_1 - x_2 - 0.5)$$
$$h_2 = \sigma(x_1 - x_2 + 0.5)$$
$$\boldsymbol{Y} = \sigma(h_1 - h_2 + 0.5)$$

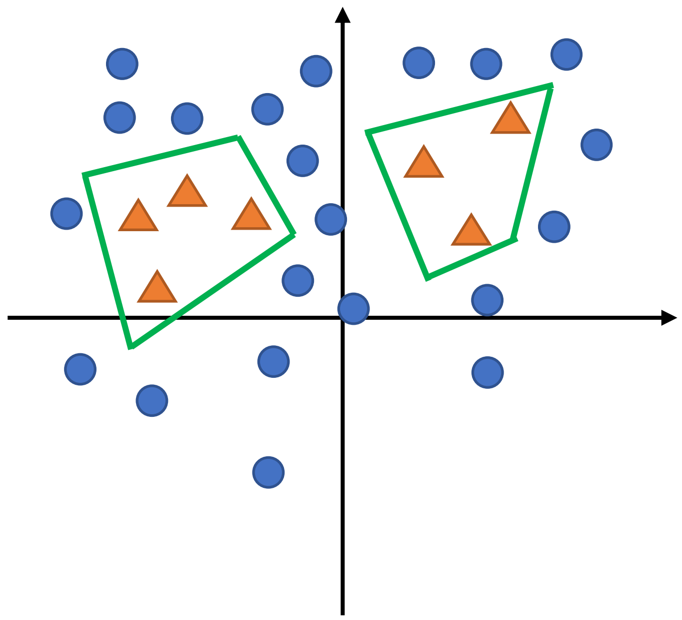


$\boldsymbol{h}$	$\boldsymbol{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$	$\boldsymbol{Y}$
(0, 1)	-0.5	0
(0, 0)	0.5	1
(1, 1)	0.5	1
(0, 1)	-0.5	0



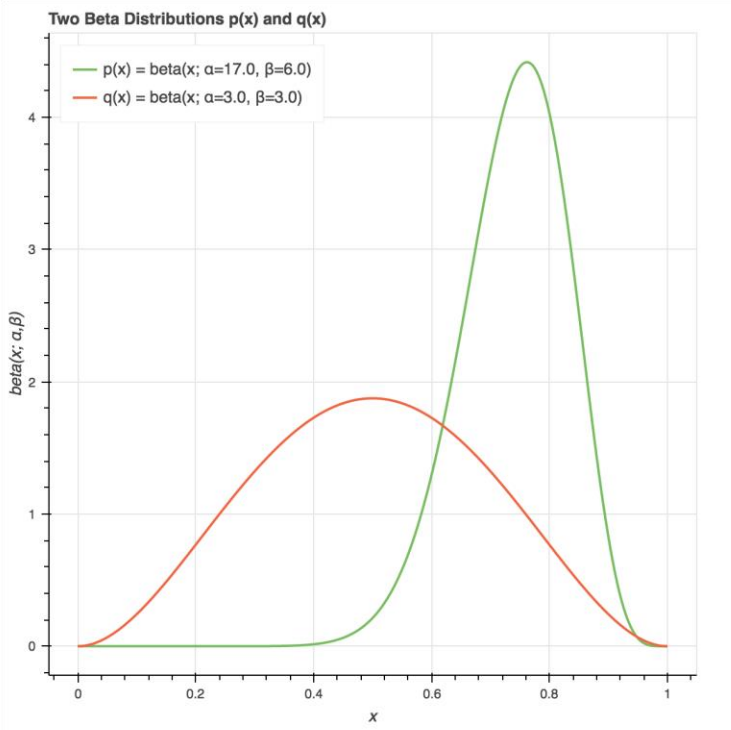
XOR Review

- Multi-layer Perceptron can deal with **more complex** problems.



XOR Review

- **Cross Entropy**



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

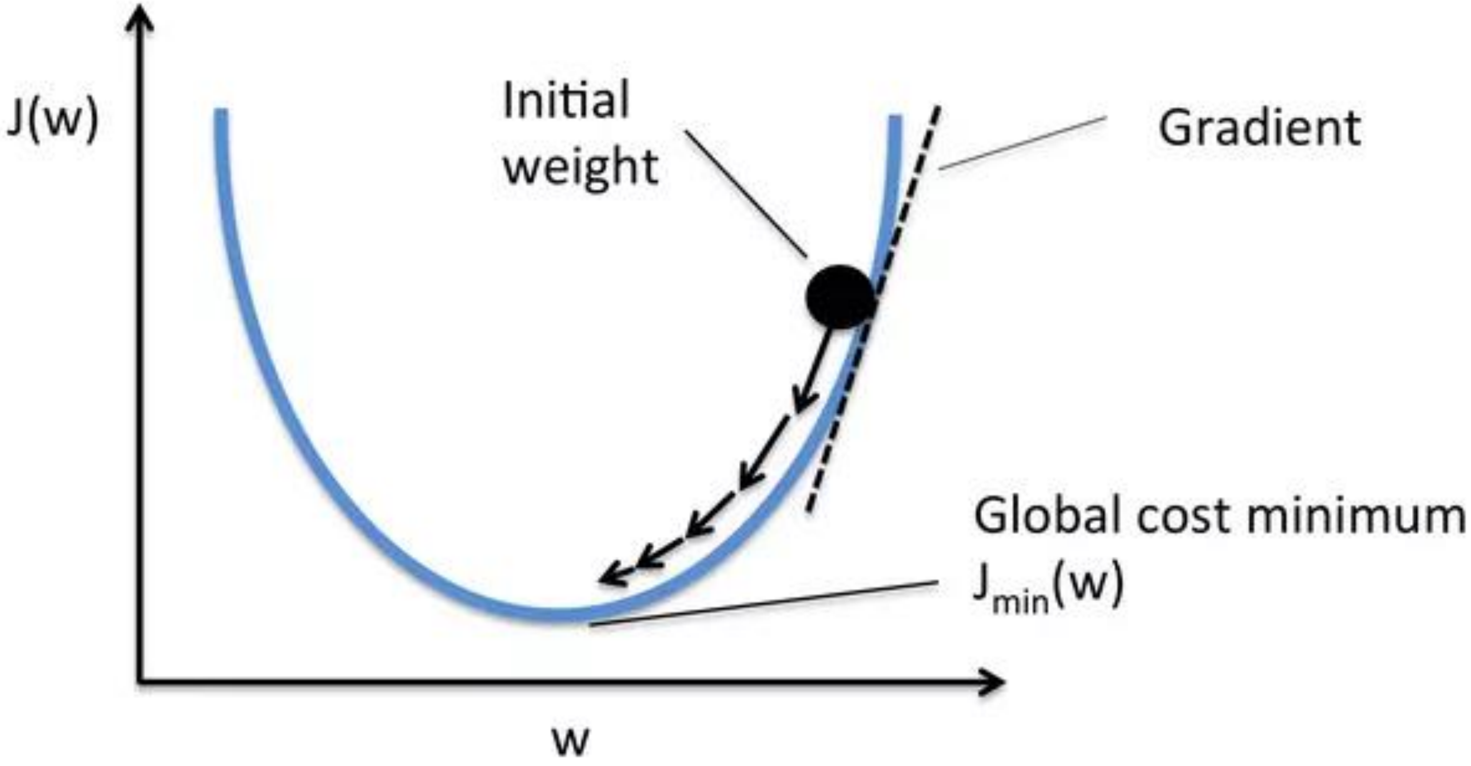
Measure distance between two probability distribution
Mostly used in classification problems.

Loss : 1.7

XOR Review

- **Gradient Descent**

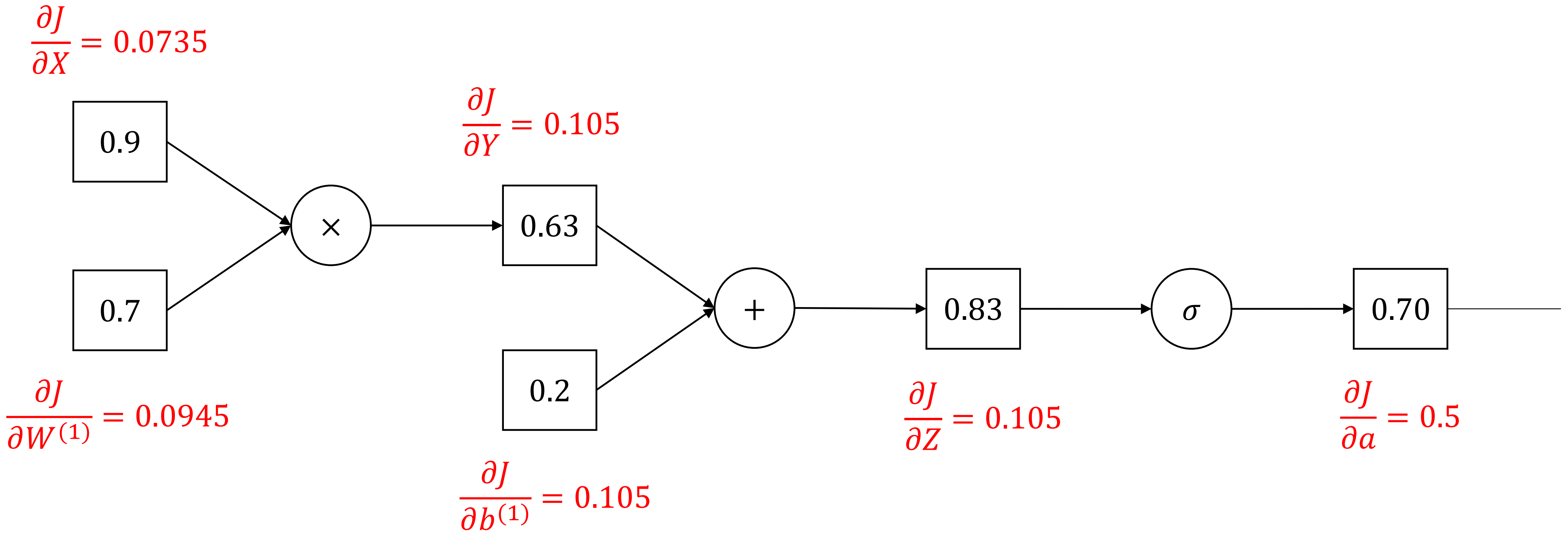
We should minimize the loss using differentiation.



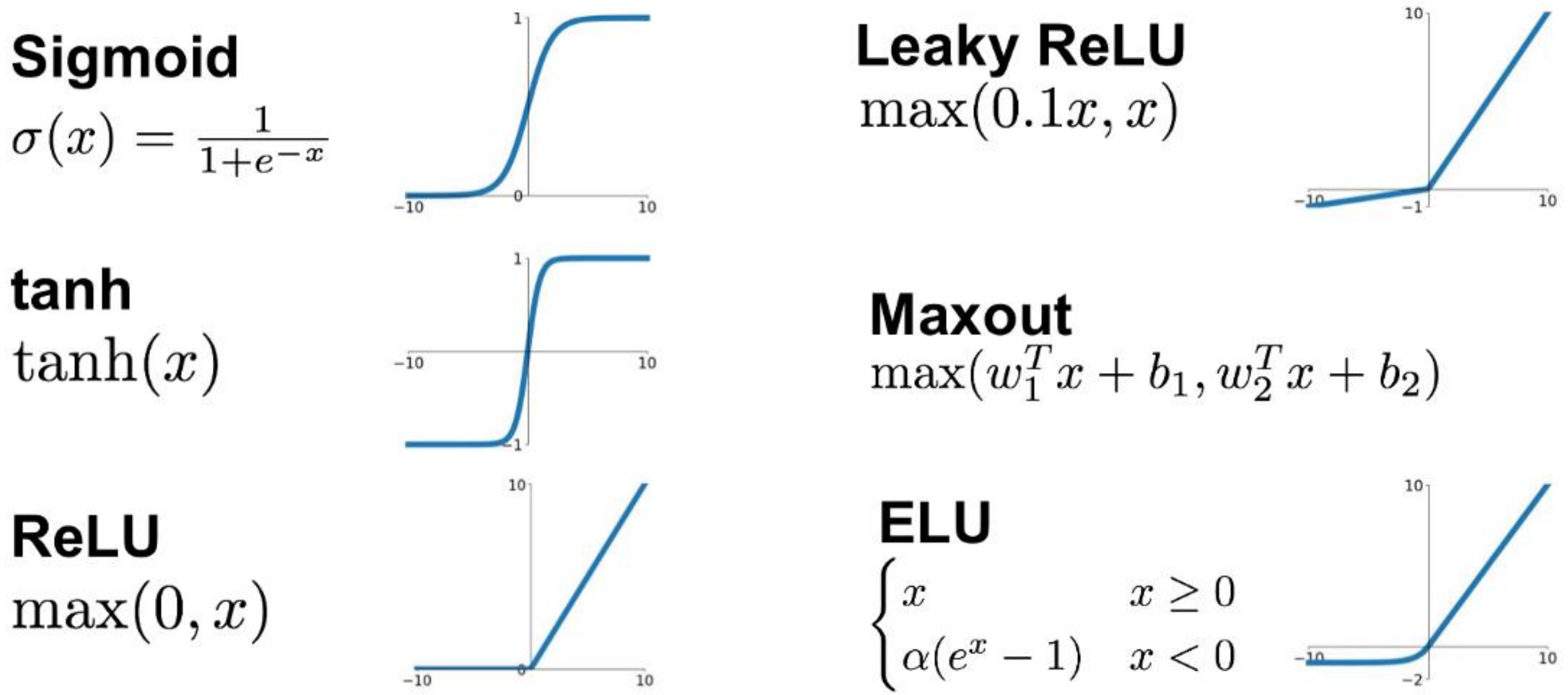
Gradient : $\frac{\partial J(w)}{\partial w}$

XOR Review

- **Backpropagation**: Using chain rule, calculate gradient step-by-step.



Activation Function



Optimizer

- **AdaDelta**

This is an another upgraded version of Adagrad.

$$G = \gamma G + (1-\gamma)(\nabla_\theta J(\theta_t))^2$$

$$\Delta_\theta = \frac{\sqrt{s+\epsilon}}{\sqrt{G+\epsilon}} \cdot \nabla_\theta J(\theta_t)$$

$$\theta = \theta - \Delta_\theta$$

$$s = \gamma s + (1-\gamma)\Delta_\theta^2$$

s: step size (instead of learning rate)

- **Adam**

This is mixture of RMSProp and momentum. This is one of the **most popular** gradient descent optimization algorithms.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta)$$

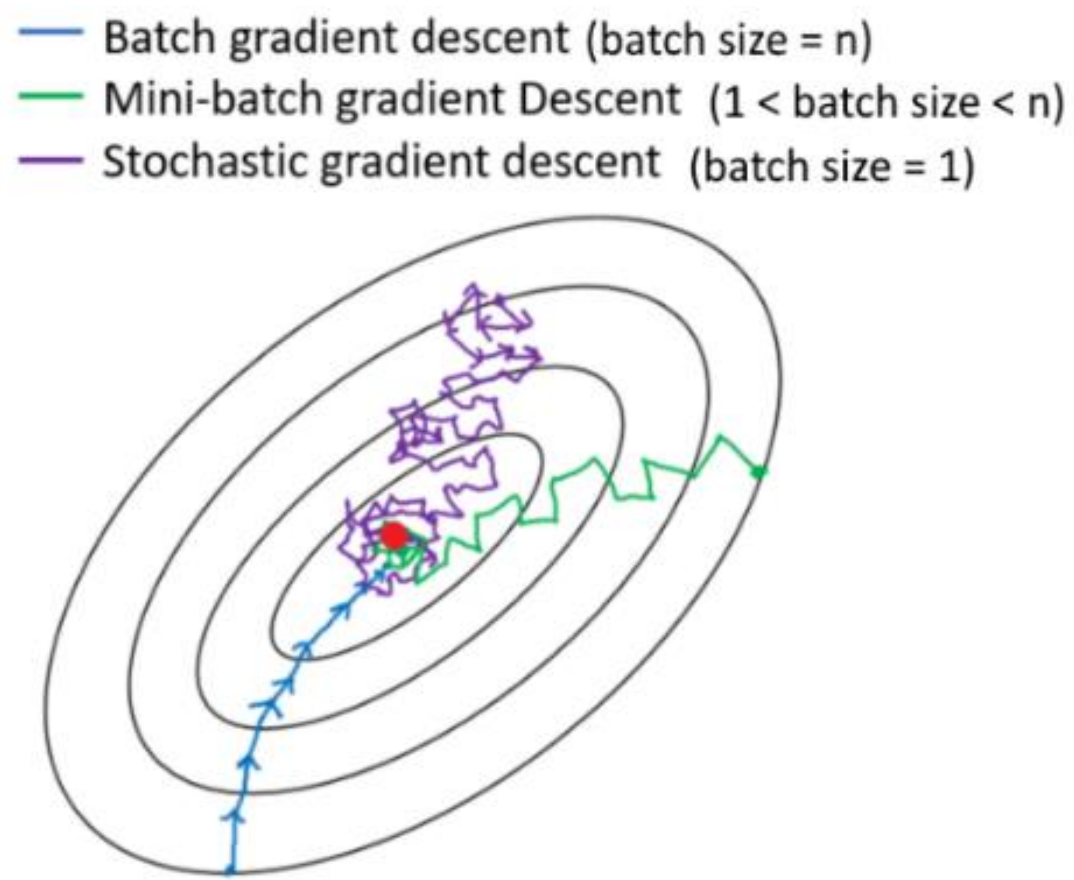
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta))^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

Mini-batch



- **Batch gradient descent:**

compute gradient with all the training data for each step.
-> It needs too much computation cost.

- **Stochastic gradient descent:**

compute gradient with one training data for each step.
-> gradient descents with a lot of noise.

- **Mini-batch gradient descent:**

-> compute gradient with n batch size of training data.
※ Choosing appropriate batch size is important when we train the model.

Weight initialization

(1) Xavier Normal Initialization[1]

$$W \sim N(0, Var(W))$$

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

(2) He Normal Initialization[2]

$$W \sim N(0, Var(W))$$

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

n_{in} = the number of nodes of the previous layer

n_{out} = the number of nodes of the next layer

- (1) is also called as **'Glorot-Bengio Initialization'**.
- (1) is efficient when we use **sigmoid** or **tanh** activation function.
- (2) is efficient when we use **ReLU** activation function.

[1] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.

[2] He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

Early stopping

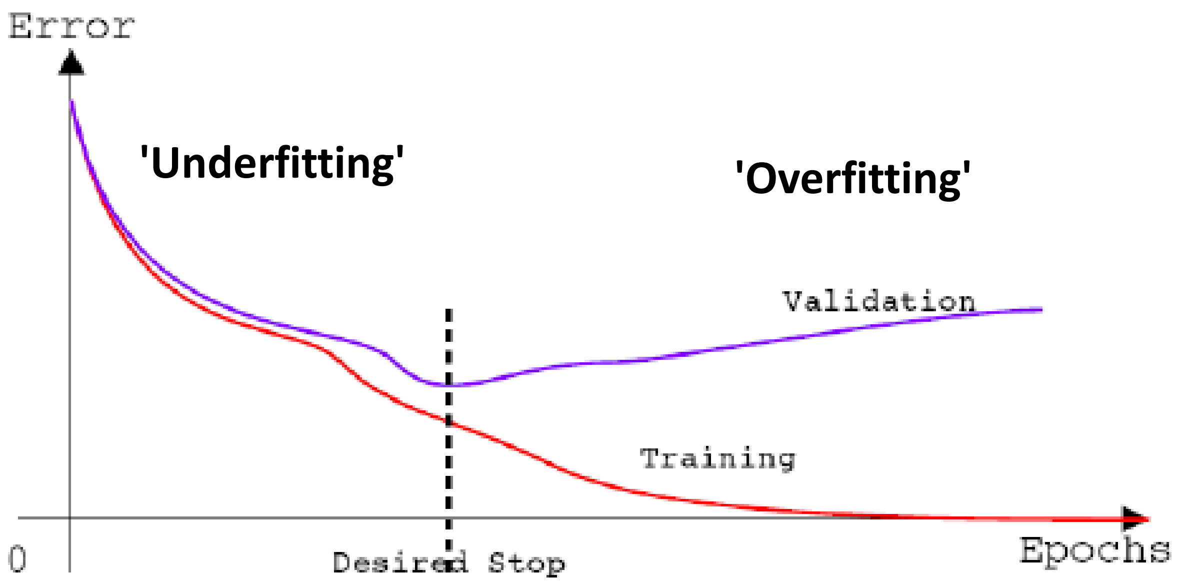
Train set	Dev set	Test set

Generally, deep learning models are trained with train/dev(validation)/test set.

We stop training when the validation error increase again while training error keeps decrease.

This is called **"Early Stopping".**

We use early stopping to prevent overfitting.



Parameter norm penalties

- To prevent overfitting, we add weight decay or weight restriction when calculating error term.

Weight decay: $$E_t = \frac{1}{N_t}\sum_{n\in D_t} E_n + \frac{\lambda}{2}\underbrace{||w||^2}_{\text{L2-norm}}$$

$$w^{t+1} = w^t - \varepsilon\left(\frac{1}{N_t}\sum \nabla E_n + \lambda w^t\right)$$

Weight restriction: $||w||^2 < c$

Parameter norm penalties

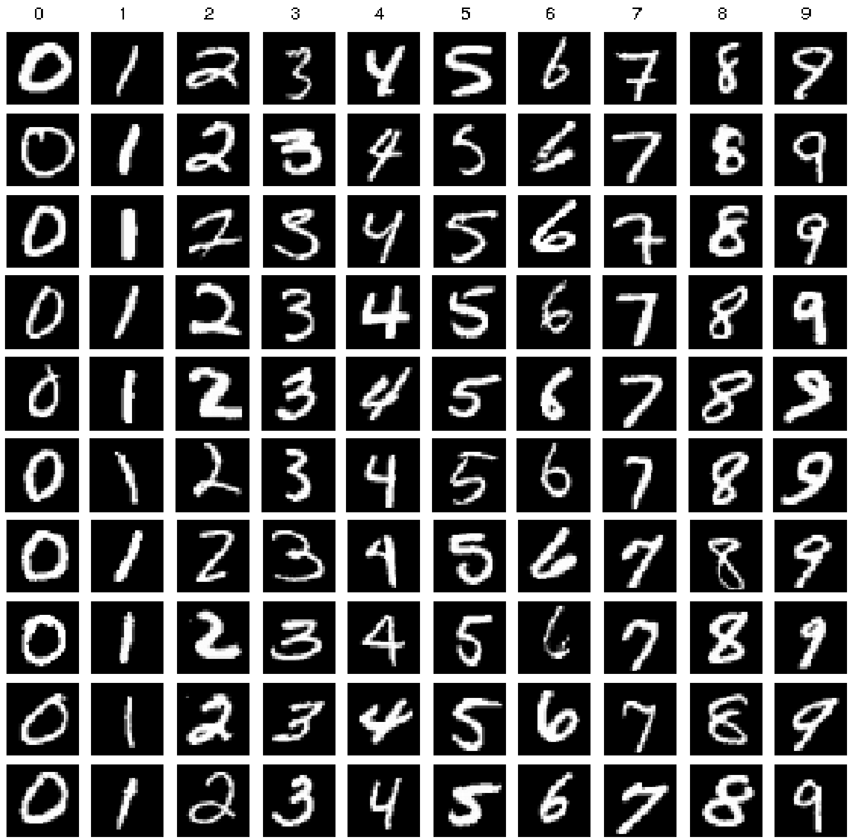
- To prevent overfitting, we add weight decay or weight restriction when calculating error term.

Weight decay: $$E_t = \frac{1}{N_t}\sum_{n \in D_t} E_n + \frac{\lambda}{2}\underbrace{||w||^2}_{\text{L2-norm}}$$

$$w^{t+1} = w^t - \varepsilon\left(\frac{1}{N_t}\sum \nabla E_n + \lambda w^t\right)$$

Weight restriction: $||w||^2 < c$

Assignment #2 : MNIST



Assignment #2 : MNIST

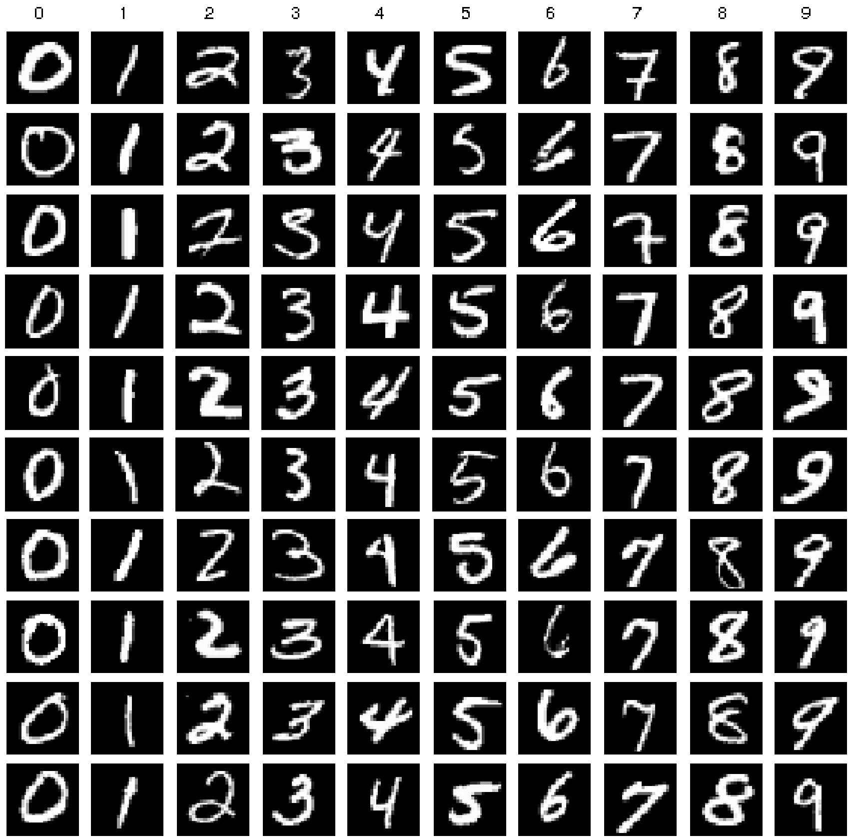
- **Introduction**

  - MNIST : **M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology database

  - It was created by "re-mixing" the samples from NIST's original dataset[1]s.

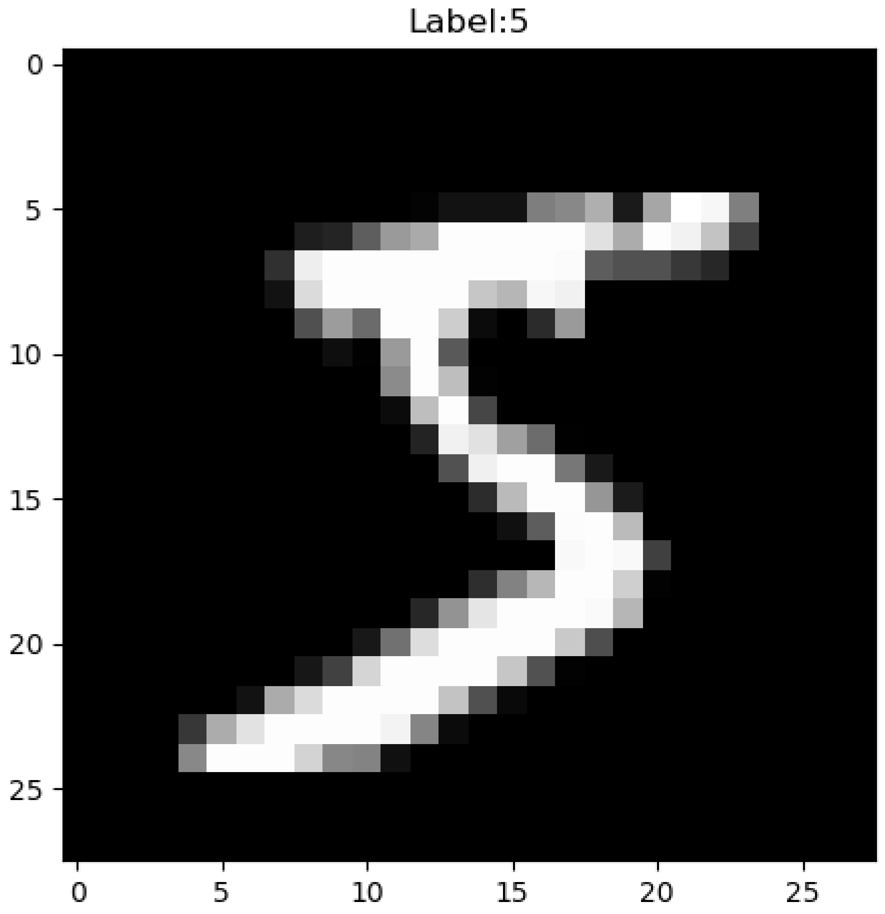
  - The database is also widely used for training and testing in the field of machine learning.

  - The database contains 60,000 training images, 10,000 validation images, and 10,000 testing images with 10 classes.



[1] https://www.nist.gov/system/files/documents/srd/nistsd19.pdf

Assignment #2 : MNIST



- Shape of each data : [28, 28]

- Range : 0.0 to 1.0

- You can see the image of each data.
  (available in the assignment code)

Code review

[MNIST_train.py]

[Objective]

Your model should classifiy of the images into 10 classes (0~9).

[Code structure]

- **MNIST_train.py**
- **MNIST_model.py**
- **MNIST_evaluation.py**

```python
if __name__ == '__main__':
    print('[MNIST_training]')
    # GPU 사용이 가능하면 사용하고, 불가능하면 CPU 활용
    print("GPU Available:", torch.cuda.is_available())
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # configuration
    cfg = Config()

    # 데이터 로드
    # MNIST datset: 28 * 28 사이즈의 이미지들을 가진 dataset
    train_data, val_data = data_load()

    # data 개수 확인
    print('The number of training data: ', len(train_data))

    # shape 및 실제 데이터 확인
    image, label = train_data[0]
    imgshow(image, label)

    # 학습 모델 생성
    model = MNIST_model().to(device)

    # 배치 생성
    train_batch_loader, val_batch_loader = generate_batch(train_data, val_data)

    ##############################################################################################################
    #                       TODO : 모델 학습을 위한 optimizer 정의                                                #
    ##############################################################################################################
    pass
    ##############################################################################################################
    #                                                   END OF YOUR CODE                                         #
    ##############################################################################################################
```

```python
def data_load():

    # MNIST dataset 다운로드
    train_data = dsets.MNIST(root='./dataset/', train=True, transform=transforms.ToTensor(), download=True)
    val_data = dsets.MNIST(root="./dataset/", train=False, transform=transforms.ToTensor(), download=True)

    return train_data, val_data
```

```python
def generate_batch(train_data, val_data):
    train_batch_loader = DataLoader(train_data, cfg.batch_size, shuffle=True)
    val_batch_loader = DataLoader(val_data, cfg.batch_size, shuffle=True)
    return train_batch_loader, val_batch_loader
```

Assignment #2 : MNIST

[MNIST_train.py]

```python
print("Start training")
for epoch in range(cfg.epoch):
    train_loss = 0
    train_batch_cnt = 0
    model.train()
    for img, label in train_batch_loader:
        # img.shape: [200,1,28,28]
        # label.shape: [200]
        img = img.to(device)
        label = label.to(device)

        # input data shape: [200,28*28]

        ################################################################################################
        #               TODO : foward path를 진행하고 손실을 loss에 저장 후 train_loss에 더함, 모델 학습 진행          #
        ################################################################################################
        pass
        #################################################################################################
        #                                              END OF YOUR CODE                                   #
        #################################################################################################

        train_batch_cnt += 1
    ave_loss = train_loss / train_batch_cnt
    training_time = (time.time() - start_time) / 60
    print('========================================')
    print("epoch:", epoch + 1)
    print("training dataset average loss: %.3f" % ave_loss)
    print("training_time: %.2f minutes" % training_time)

    # validation (for early stopping)
    correct_cnt = 0
    model.eval()
    for img, label in val_batch_loader:
        img = img.to(device)
        label = label.to(device)
        pred = model.forward(img.view(-1, 28 * 28))
        _, top_pred = torch.topk(pred, k=1, dim=-1)
        top_pred = top_pred.squeeze(dim=1)
        correct_cnt += int(torch.sum(top_pred == label))

    val_acc = correct_cnt / len(val_data) * 100
    print("validation dataset accuracy: %.2f" % val_acc)
    val_acc_list.append(val_acc)
    if val_acc > highest_val_acc:
        save_path = './saved_model/setting_1/epoch_' + str(epoch + 1) + '.pth'
        # 위와 같이 저장 위치를 바꾸어 가며 각 setting의 epoch마다의 state를 저장할 것.
        torch.save({'epoch': epoch + 1,
                    'model_state_dict': model.state_dict()},
                   save_path)
        highest_val_acc = val_acc
```

Assignment #2 : MNIST

[MNIST_model.py]

```python
class MNIST_model(nn.Module):
    def __init__(self):
        super().__init__()
        ##############################################################################################################
        #                          TODO : 4-layer feedforward 모델 생성 (evaluation report의 세팅을 사용할 것)          #
        ##############################################################################################################
        pass
        ###############################################################################################################
        #                                              END OF YOUR CODE                                               #
        ###############################################################################################################

    def forward(self, x):
        ##############################################################################################################
        #                          TODO : forward path 수행, 결과를 x에 저장                                           #
        ##############################################################################################################
        pass
        ###############################################################################################################
        #                                              END OF YOUR CODE                                               #
        ###############################################################################################################
        return x


class Config():
    def __init__(self):
        self.batch_size = 200
        self.lr_adam = 0.0001
        self.lr_adadelta = 0.1
        self.epoch = 100
        self.weight_decay = 1e-03
```

Assignment #2 : MNIST

[MNIST_evaluation.py]

```python
if __name__ == "__main__":
    print('[MNIST_evaluation]')
    cfg = Config()

    # 모델 생성
    model = MNIST_model()
    model.eval()

    # 데이터 로드
    test_data = data_load()

    # data 개수 확인
    print('The number of test data: ', len(test_data))

    # 배치 생성
    test_batch_loader = generate_batch(test_data)

    # test 시작
    acc_list = []


    # 저장된 state 불러오기
    save_path = "./saved_model/setting_1/epoch_1.pth"
    # TODO : 세팅값 마다 save_path를 바꾸어 로드
    checkpoint = torch.load(save_path)
    model.load_state_dict(checkpoint['model_state_dict'])
    epoch = checkpoint['epoch']
    correct_cnt = 0
    for img, label in test_batch_loader:
        pred = model.forward(img.view(-1, 28 * 28))
        _, top_pred = torch.topk(pred, k=1, dim=-1)
        top_pred = top_pred.squeeze(dim=1)

        correct_cnt += int(torch.sum(top_pred == label))

    accuracy = correct_cnt / len(test_data) * 100
    print("accuracy of the 87 epoch trained model:%.2f%%" % accuracy)
    acc_list.append(accuracy)
```

Assignment #2 : MNIST

[Evaluation report]

B	C	D	E	F	G	H	I	J	K	L	M	N
MNIST Evaluation Report												
	Batch_size	**Activation function**	**# of layers**	**Layer size**	**Epoch**	**Weight initialization**	**Optimizer**	**Learning rate**	**Weight decay**	**100 epoch training time**	**Early stopping epoch**	**Accuracy**
Setting #1	200	ReLU	4	200	100	He	Adadelta	0.1	X			
Setting #2	200	ReLU	4	200	100	He	Adam	0.0001	X			
Setting #3	200	ReLU	4	200	100	He	Adadelta	0.1	O(lambda=0.01)			
Setting #4	200	ReLU	4	200	100	He	Adam	0.0001	O(lambda=0.01)			
Validation dataset accuracy plot												

Setting #1 **Setting #2** **Setting #3** **Setting #4**

[결과 정리]

Assignment #2 : MNIST

Fill in these cells.

[Evaluation report]

B	C	D	E	F	G	H	I	J	K	L	M	N
MNIST Evaluation Report												
	Batch_size	**Activation function**	**# of layers**	**Layer size**	**Epoch**	**Weight initialization**	**Optimizer**	**Learning rate**	**Weight decay**	**100 epoch training time**	**Early stopping epoch**	**Accuracy**
Setting #1	200	ReLU	4	200	100	He	Adadelta	0.1	X			
Setting #2	200	ReLU	4	200	100	He	Adam	0.0001	X			
Setting #3	200	ReLU	4	200	100	He	Adadelta	0.1	O(lambda=0.01)			
Setting #4	200	ReLU	4	200	100	He	Adam	0.0001	O(lambda=0.01)			
Validation dataset accuracy plot												

Setting #1 **Setting #2** **Setting #3** **Setting #4**

[결과 정리]

Assignment #2 : MNIST

Plot an accuracy plot of the validation dataset for each setting.

[Evaluation report]

MNIST Evaluation Report												
	Batch_size	**Activation function**	**# of layers**	**Layer size**	**Epoch**	**Weight initialization**	**Optimizer**	**Learning rate**	**Weight decay**	**100 epoch training time**	**Early stopping epoch**	**Accuracy**
Setting #1	200	ReLU	4	200	100	He	Adadelta	0.1	X			
Setting #2	200	ReLU	4	200	100	He	Adam	0.0001	X			
Setting #3	200	ReLU	4	200	100	He	Adadelta	0.1	O(lambda=0.01)			
Setting #4	200	ReLU	4	200	100	He	Adam	0.0001	O(lambda=0.01)			

Validation dataset accuracy plot

Setting #1	**Setting #2**	**Setting #3**	**Setting #4**

[결과 정리]

Assignment #2 : MNIST

[Evaluation report]

B	C	D	E	F	G	H	I	J	K	L	M	N
MNIST Evaluation Report												
	Batch_size	**Activation function**	**# of layers**	**Layer size**	**Epoch**	**Weight initialization**	**Optimizer**	**Learning rate**	**Weight decay**	**100 epoch training time**	**Early stopping epoch**	**Accuracy**
Setting #1	200	ReLU	4	200	100	He	Adadelta	0.1	X			
Setting #2	200	ReLU	4	200	100	He	Adam	0.0001	X			
Setting #3	200	ReLU	4	200	100	He	Adadelta	0.1	O(lambda=0.01)			
Setting #4	200	ReLU	4	200	100	He	Adam	0.0001	O(lambda=0.01)			
Validation dataset accuracy plot												

Setting #1 **Setting #2** **Setting #3** **Setting #4**

Summarize the report of each experimental setting.

[결과 정리]

Assignment #2 : MNIST

- **Objective**

Your model should classifiy of the images into 10 classes (0~9).

- **Requirements**

1. Implement 4-layer perceptron with Pytorch or Tensorflow.

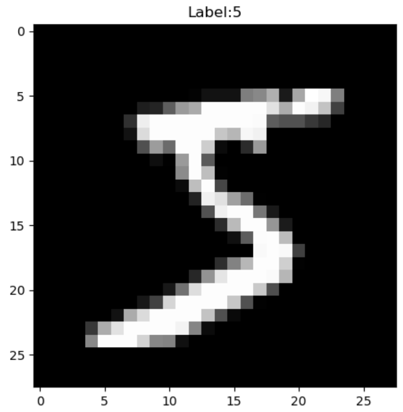
(Basic Pytorch code is provided)

2. You should experiment with 4 settings stated in the evaluation report, and report the result of each settings.

3. You should attach the plot of the validation dataset accuracy plot. (implemented in pytorch code)

4. You should report the experimental results.

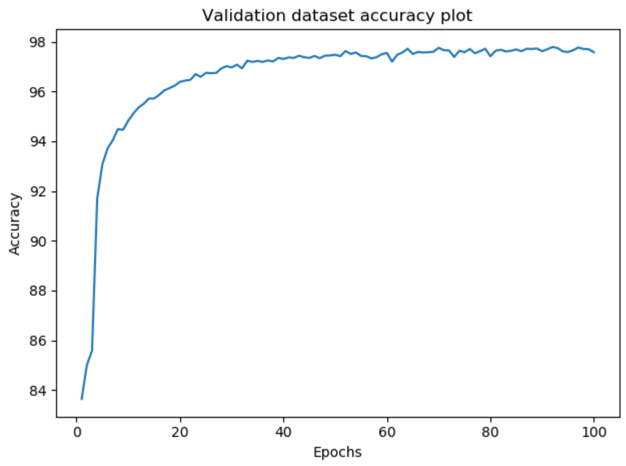
(all kinds of additional experiments are recommended)



model

"5!"

[Validation dataset accuracy plot]



Assignment #2 : MNIST

- **Evaluation Criteria**

Simplicity	How concisely did you write the code?
Performance	How well did the results of the code perform?
Brevity and Clarity	How concisely and clearly did you explain the results?

Assignment #2 : MNIST

- Due to : **~ 4.19(Sun)**

- Submission : Online submission on blackboard

- Your submission should contain
   1) The whole code of your implementation
   2) The evaluation report

- You must implement the components yourself!
- File name : StudentID_Name.zip

Q & A

조교 김도현 : dhkim1028@korea.ac.kr