
Multi-Layer Perceptron

6조 김은비 권동구 조성우 조형주

Contents

1. MLP 개요

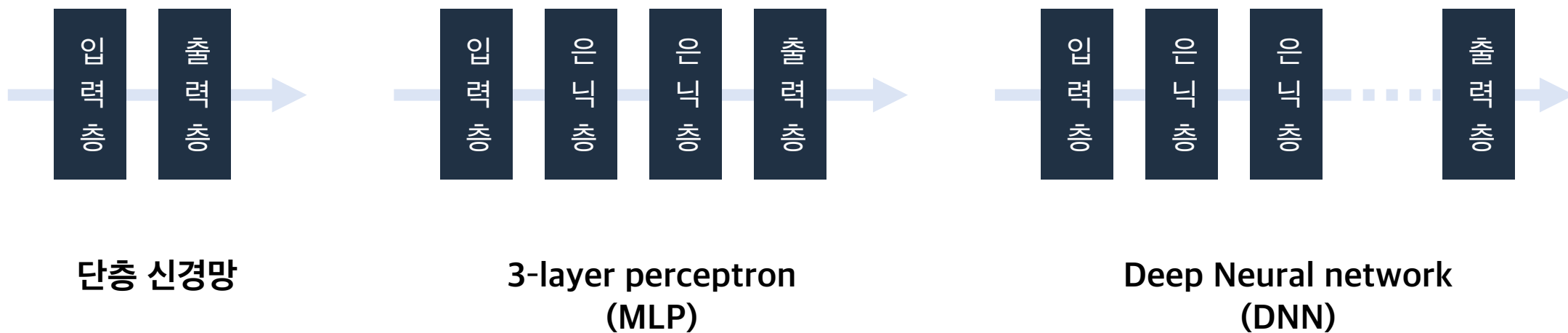
2. MLP 구조

3. MLP 학습

4. MLP 실습코드

1. MLP 개요

- 인공신경망 종류
- MLP 핵심 아이디어
- MLP 도입 이유



3 Layer 까지를 'Multi-layer Perceptron(MLP)'라고 지칭

① 은닉층의 도입

- 선형 분리 불가능한 문제를 선형분리 가능하도록 변환

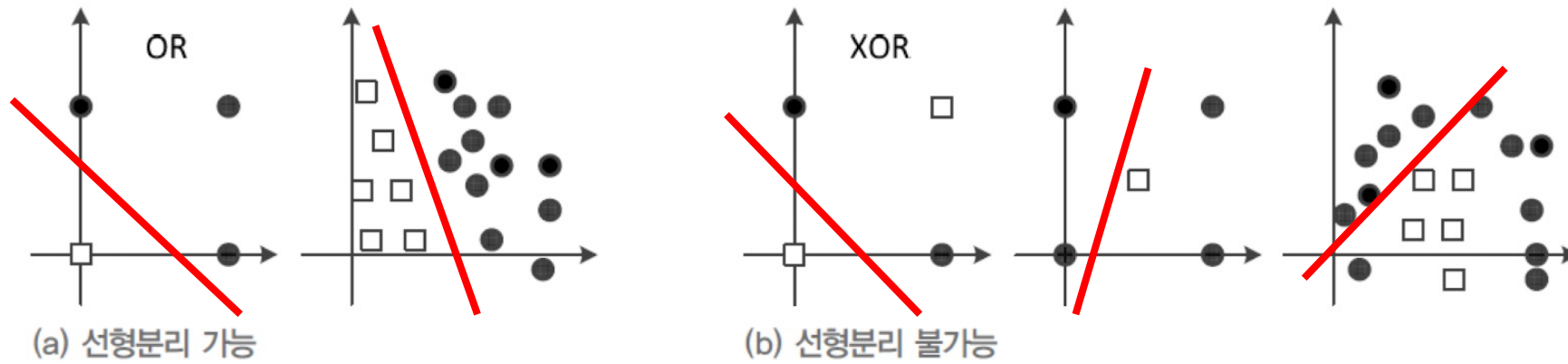
② Sigmoid 활성화함수의 도입

- 출력 값을 확률로 간주하여 융통성 있는 의사결정 가능

③ 오류 역전파(back-propagation) 알고리즘의 도입

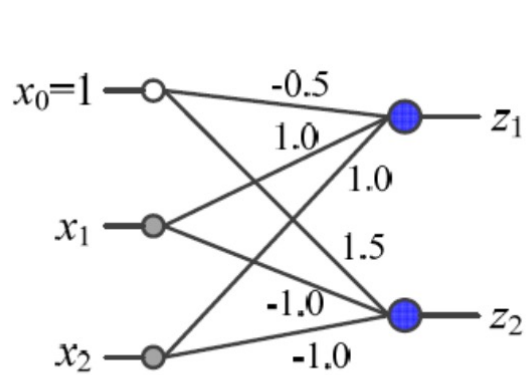
- 다층 신경망의 학습이 어려웠던 문제를 해결

Perceptron의 한계

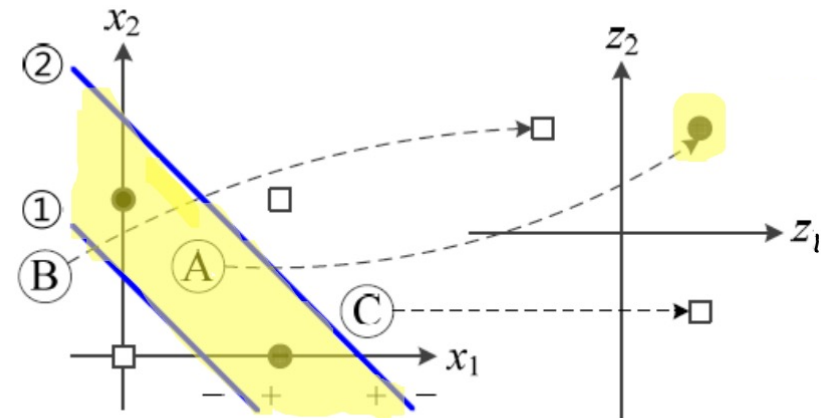


선형 분리가 불가능한 상황에서는 일정량의 오류가 발생

비선형 분류 문제 해결(XOR)



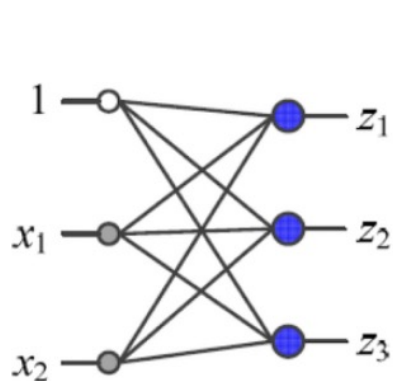
(a) 두 퍼셉트론을 병렬로 결합



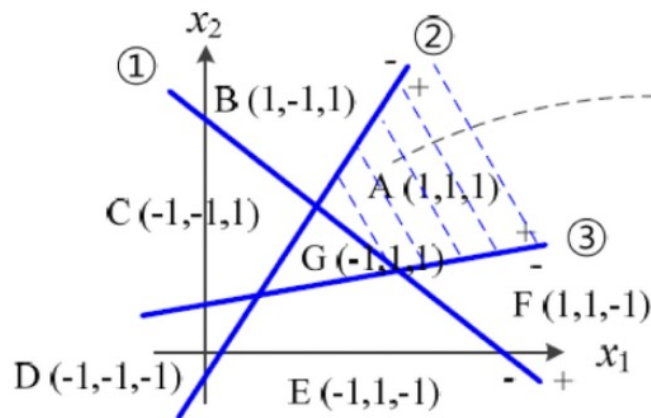
(b) 원래 특징 공간 x 를 새로운 특징 공간 z 로 변환

Perceptron 2개를 활용하여 새로운 특징공간으로 변형 후, 선형 분리

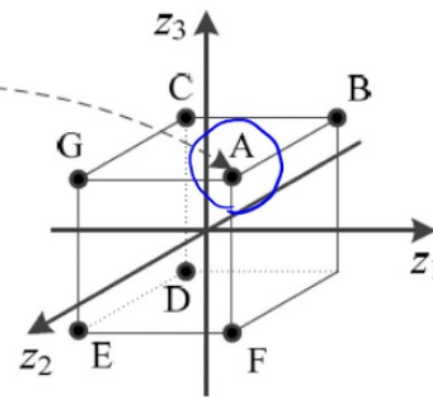
비선형 분류 문제 해결



(a) 퍼셉트론 3개를 결합



(b) 7개 부분공간으로 나눔

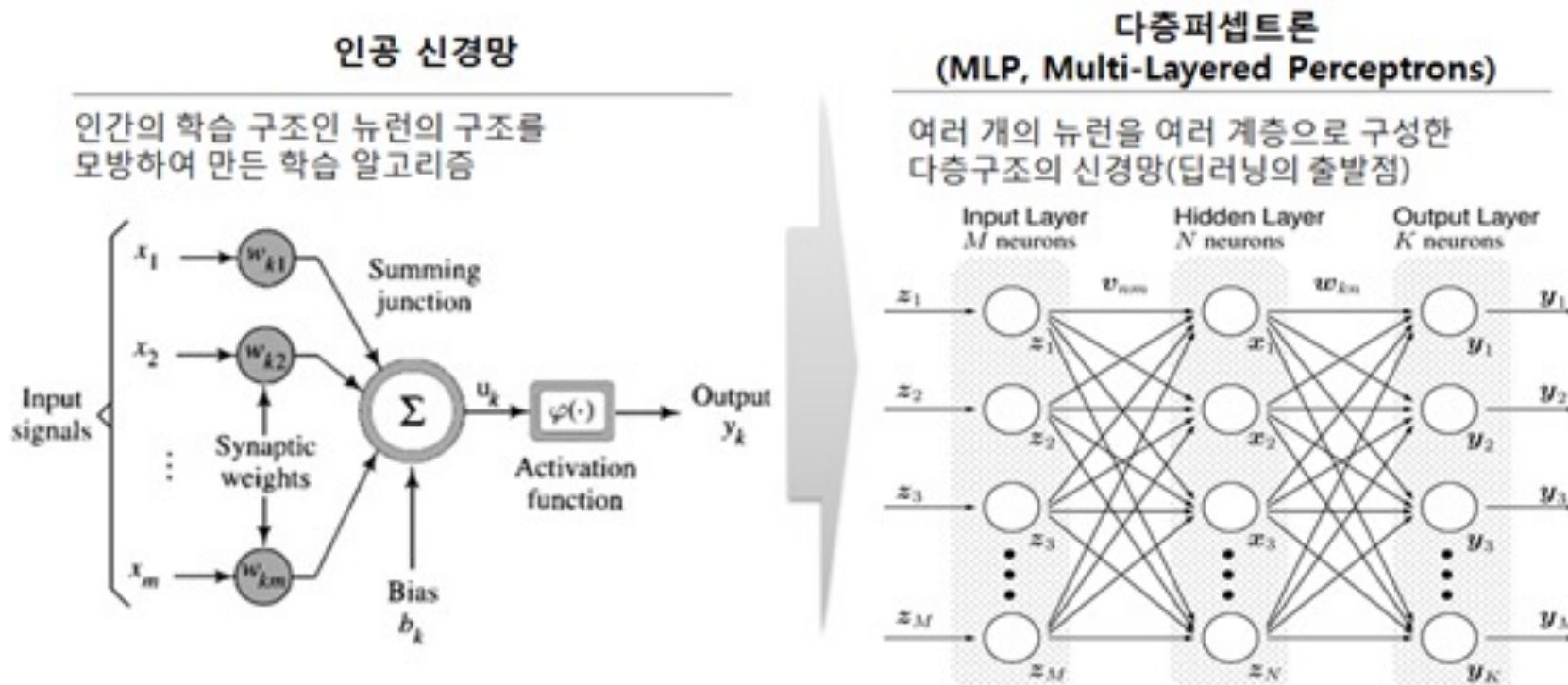


(c) 3차원 공간의 점으로 매핑

P개의 Perceptron을 활용하면, P차원 공간으로 매핑 가능 → 차원이 늘어날수록 분류가 쉬워짐!

2. MLP 구조

- 활성화함수



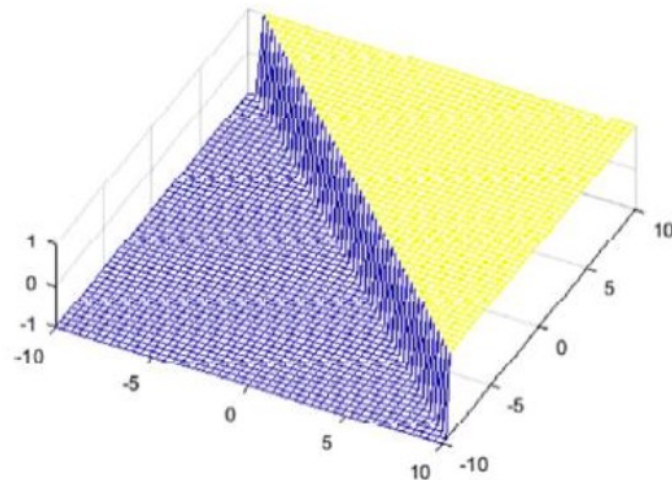
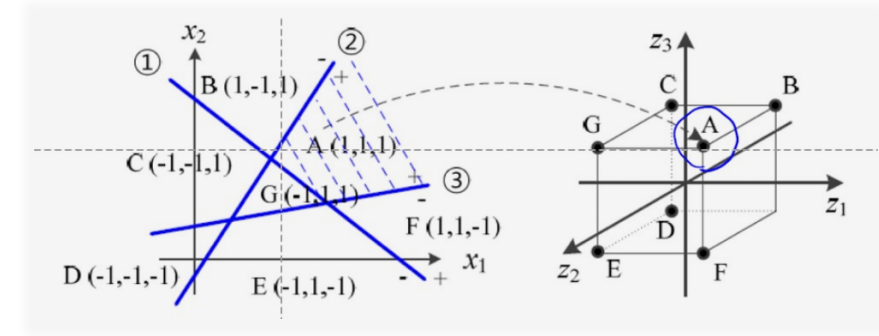
은닉층을 통해서 공간 변환을 수행하며, 이러한 과정을 통해 효율적인 특징을 뽑아내기도 함

▶ “Feature Learning”

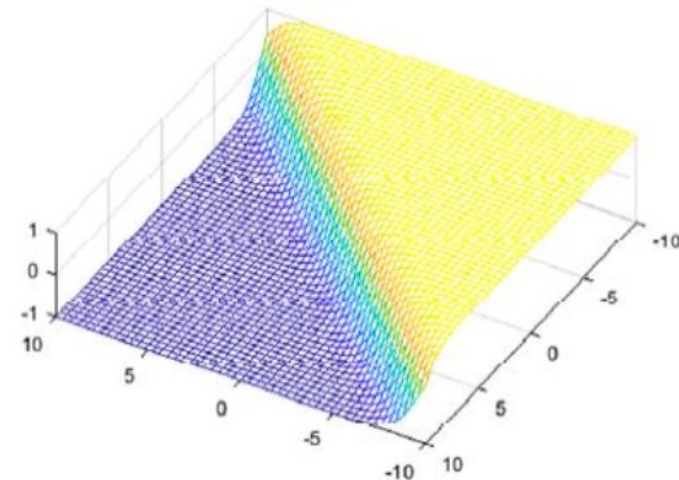
활성함수

2. MLP 구조

연성 활성화함수 사용



(a) 계단함수의 딱딱한 공간 분할

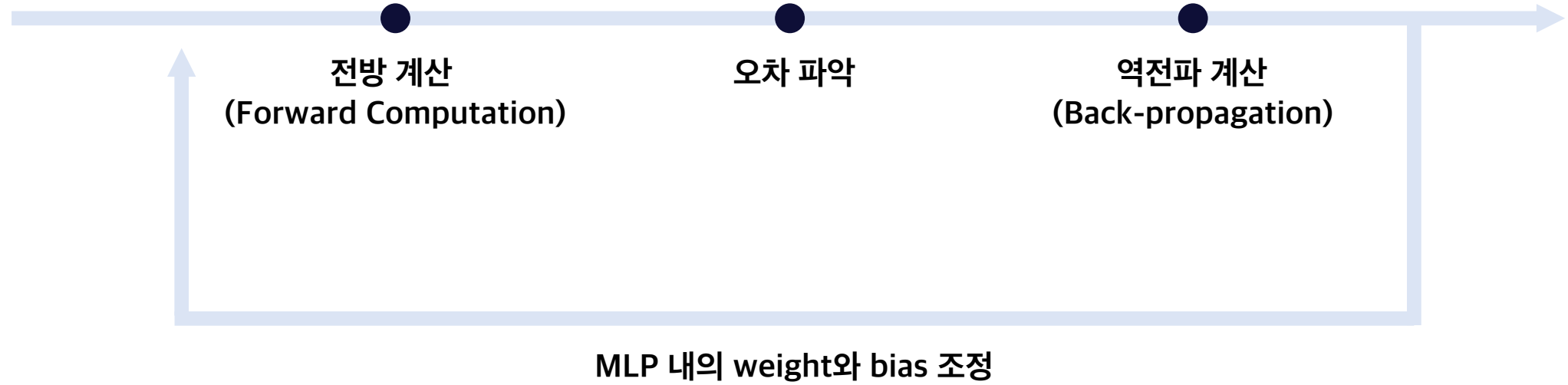


(b) 로지스틱 시그모이드의 부드러운 공간 분할

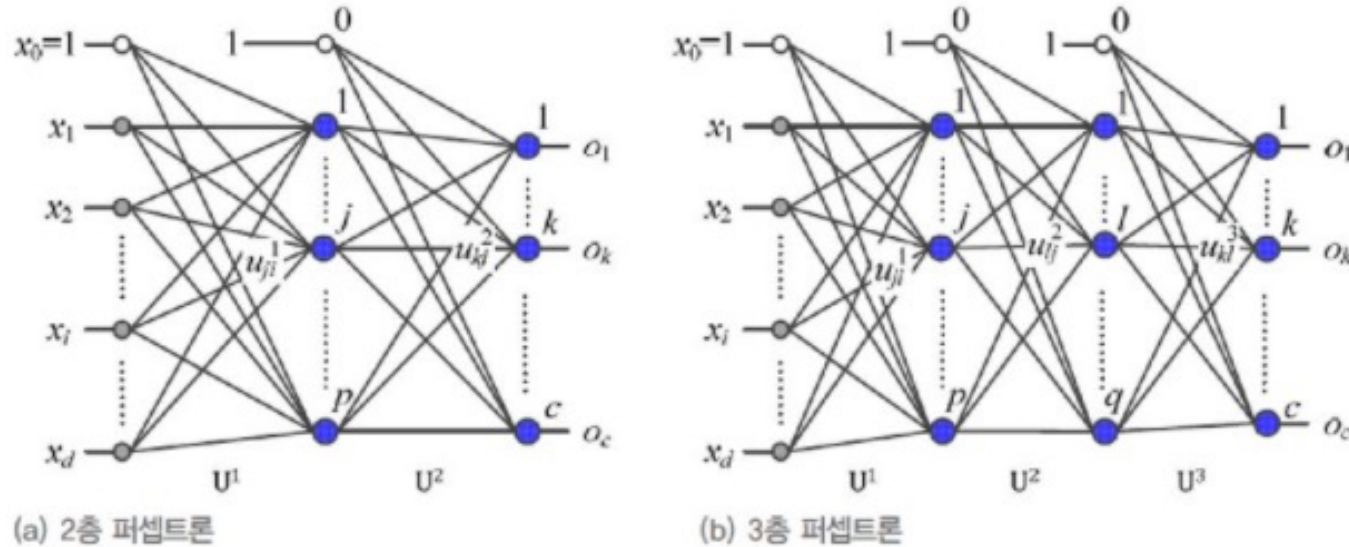
부드러운 의사결정을 위해서 ‘로지스틱 시그모이드’나 ‘하이퍼볼릭 탄젠트’ 같은 연성 활성화함수 사용

3. MLP 학습

- 학습과정
- Back-propagation 유도



전방 계산(Forward computation), 오류 파악



$$u_{ji}^l$$

▶ l-1 층의 i 번째 노드에서 l 층의 j 번째 노드로의 가중치

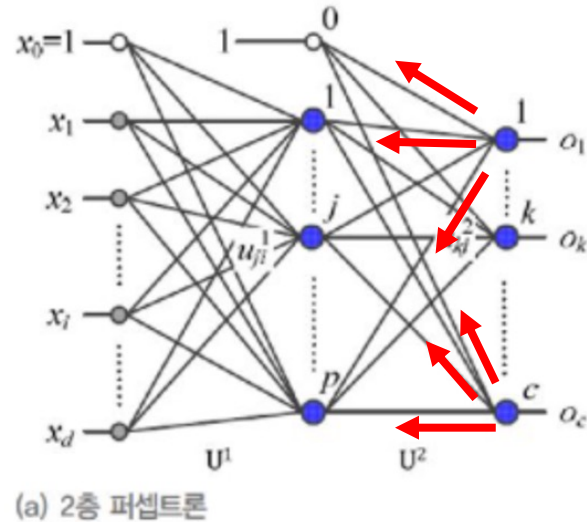
① 전방 계산(Forward computation)

- 입력 값에 대해서 가중치와 편향을 계산하여, 출력 값을 산출

② 오류 파악

- 출력 값에 대하여 정답과의 오류 계산

역전파(Back-propagation) 계산

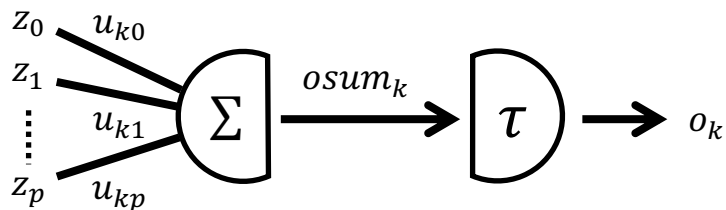
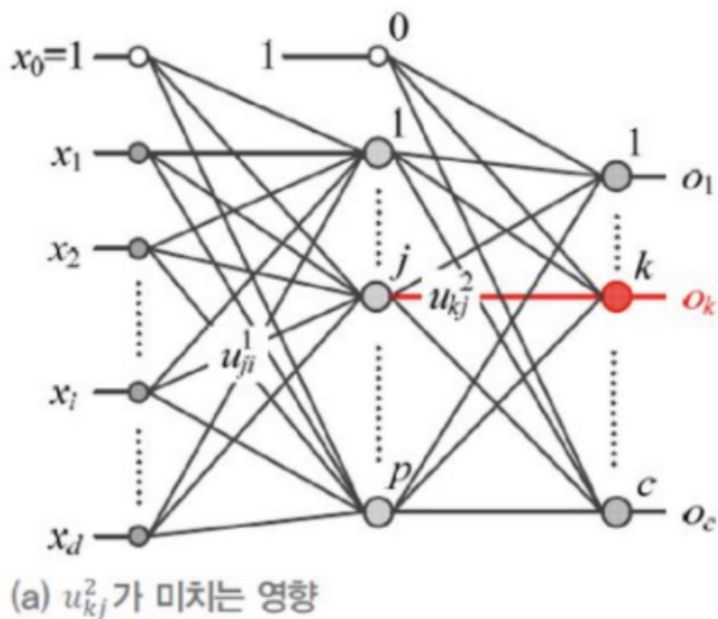


③ 역전파(Back-propagation)

- 다층 퍼셉트론을 학습하기 위한 경사하강법 기반의 알고리즘 방법
- 가중치와 편향 값이 오차에 기여하는 정도를 측정하여, 역방향으로 gradient를 측정

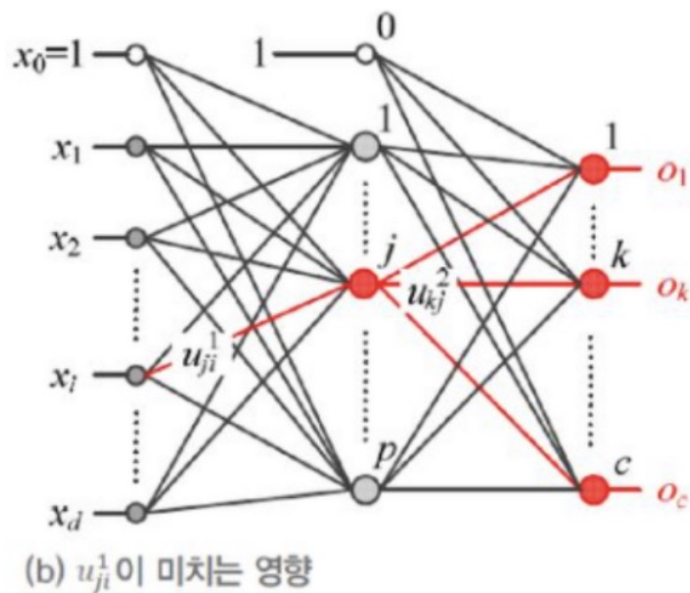
➡ 출력층의 오류를 왼쪽으로 전파하며 gradient를 계산

01. U^2 가중치에 대한 gradient 계산



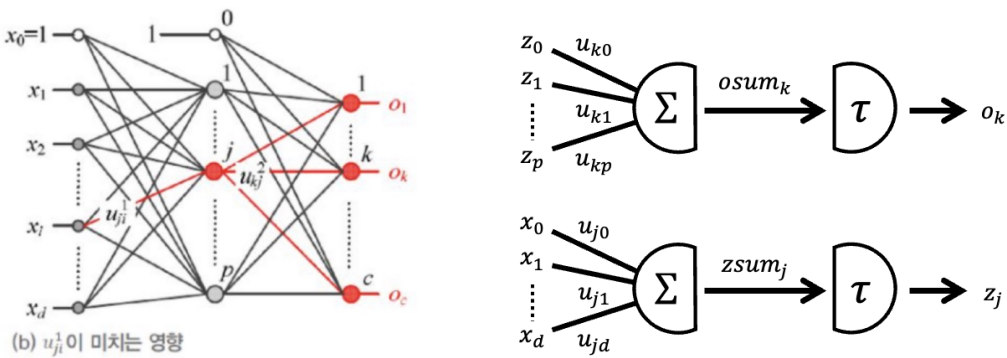
$$\begin{aligned}
 \frac{\partial J}{\partial u_{kj}^2} &= \frac{\partial (0.5 \|\mathbf{y} - \mathbf{o}(\mathbf{U}^1, \mathbf{U}^2)\|_2^2)}{\partial u_{kj}^2} \\
 &= \frac{\partial \left(0.5 \sum_{q=1}^c (y_q - o_q)^2 \right)}{\partial u_{kj}^2} \\
 &= \frac{\partial (0.5 (y_k - o_k)^2)}{\partial u_{kj}^2} \\
 &= -(y_k - o_k) \frac{\partial o_k}{\partial u_{kj}^2} \\
 &= -(y_k - o_k) \frac{\partial \tau(osum_k)}{\partial u_{kj}^2} \\
 &= -(y_k - o_k) \tau'(osum_k) \frac{\partial osum_k}{\partial u_{kj}^2} \\
 &= -(y_k - o_k) \tau'(osum_k) z_j
 \end{aligned}$$

02. U^1 가중치에 대한 gradient 계산

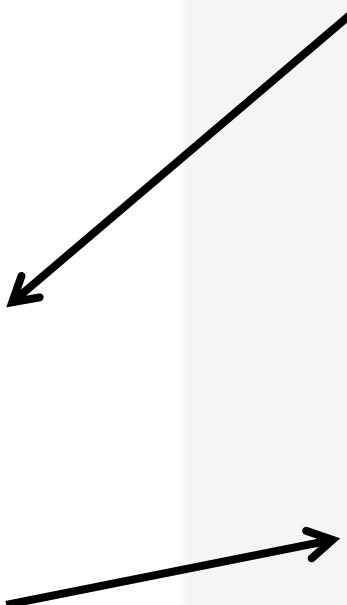


$$\begin{aligned}
 \frac{\partial J}{\partial u_{ji}^1} &= \frac{\partial (0.5 \|\mathbf{y} - \mathbf{o}(\mathbf{U}^1, \mathbf{U}^2)\|_2^2)}{\partial u_{ji}^1} \\
 &= \frac{\partial \left(0.5 \sum_{q=1}^c (y_q - o_q)^2 \right)}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \frac{\partial o_q}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \tau'(\text{osum}_q) \frac{\partial \text{osum}_q}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \tau'(\text{osum}_q) \frac{\partial \text{osum}_q}{\partial z_j} \frac{\partial z_j}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \tau'(\text{osum}_q) u_{qj}^2 \frac{\partial z_j}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \tau'(\text{osum}_q) u_{qj}^2 \tau'(zsum_j) x_i \\
 &= - \tau'(zsum_j) x_i \sum_{q=1}^c (y_q - o_q) \tau'(\text{osum}_q) u_{qj}^2
 \end{aligned}$$

02. U¹ 가중치에 대한 gradient 계산



$$\begin{aligned} &= \sum_{q=1}^c (o_q - y_q) \frac{\partial o_q}{\partial z_j} \frac{\partial z_j}{\partial u_{ji}^1} \\ &= \sum_{q=1}^c (o_q - y_q) \frac{\partial o_q}{\partial osum_q} \frac{\partial osum_q}{\partial z_j} \frac{\partial z_j}{\partial zsum_j} \frac{\partial zsum_j}{\partial u_{ji}^1} \end{aligned}$$

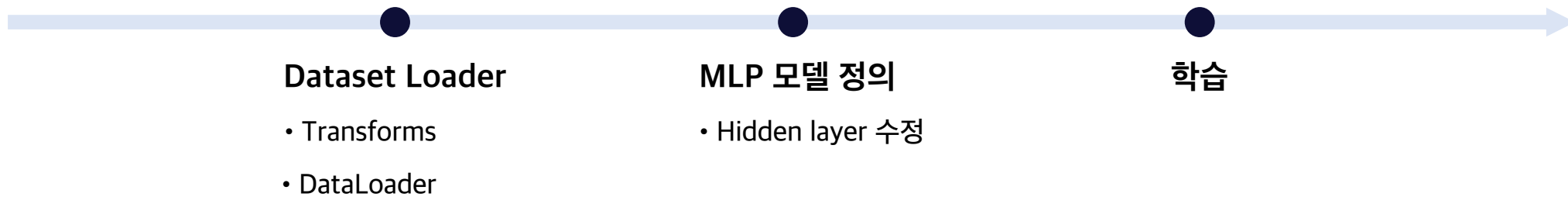


$$\begin{aligned} \frac{\partial J}{\partial u_{ji}^1} &= \frac{\partial (0.5 \|\mathbf{y} - \mathbf{o}(\mathbf{U}^1, \mathbf{U}^2)\|_2^2)}{\partial u_{ji}^1} \\ &= \frac{\partial (0.5 \sum_{q=1}^c (y_q - o_q)^2)}{\partial u_{ji}^1} \\ &= - \sum_{q=1}^c (y_q - o_q) \frac{\partial o_q}{\partial u_{ji}^1} \\ &= - \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) \frac{\partial osum_q}{\partial u_{ji}^1} \\ &= - \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) \frac{\partial osum_q}{\partial z_j} \frac{\partial z_j}{\partial u_{ji}^1} \\ &= - \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) u_{qj}^2 \frac{\partial z_j}{\partial u_{ji}^1} \\ &= - \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) u_{qj}^2 \tau'(zsum_j) x_i \\ &= - \tau'(zsum_j) x_i \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) u_{qj}^2 \end{aligned}$$


4. MLP 실습코드

- 실습코드 과정
- MLP 모델 정의
 - 학습

실습코드 과정



MLP 모델 정의

```
class MLP(torch.nn.Module):  
  
    def __init__(self, num_features, num_hidden_1, num_hidden_2, num_hidden_3, num_classes):  
        super(MLP, self).__init__()   
  
        self.num_classes = num_classes # 10  
        self.linear_1 = torch.nn.Linear(num_features, num_hidden_1)  
  
        self.linear_2 = torch.nn.Linear(num_hidden_1, num_hidden_2)  
  
        self.linear_3 = torch.nn.Linear(num_hidden_2, num_hidden_3)  
  
        self.linear_out = torch.nn.Linear(num_hidden_3, num_classes)  
  
        # Multi-class classification 문제를 풀기 위해 Softmax 함수를 사용  
        self.softmax = torch.nn.Softmax(dim=-1)  
  
    def forward(self, x):  
  
        ### activation 함수 변경 가능  
        ### 레이어간의 연결 추가, 변경  
        out = self.linear_1(x)  
        out = torch.sigmoid(out)  
  
        out = self.linear_2(out)  
        out = torch.tanh(out)  
  
        out = self.linear_3(out)  
        out = torch.tanh(out)  
  
        logits = self.linear_out(out)
```

① Hidden layer 추가

- 초기화 method에 hidden layer 추가

② forward 함수 수정

- 추가한 hidden layer 개수에 맞춰서, 레이어 간의 연결 및 활성화 함수 추가

학습

```
model = MLP(num_features=28*28,
            num_hidden_1=40,
            num_hidden_2=40,
            num_hidden_3=40,
            num_classes=10)

model = model.to(DEVICE)

optimizer = torch.optim.SGD(model.parameters(), lr=0.2) # optimizer는 가중치 업데이트 방법을 바꾸어 성능을 향상

NUM_EPOCHS = 5 # 변경 가능 (10 이상으로 설정하면 학습 시간이 너무 오래 걸릴 수 있습니다.)

def compute_accuracy_and_loss(model, data_loader, device):
    correct_pred, num_examples = 0, 0
    cross_entropy = 0.
    for i, (features, targets) in enumerate(data_loader):

        features = features.view(-1, 28*28).to(device)
        targets = targets.to(device)

        logits, probas = model(features)

        # Loss 계산 시에는 logits 이용
        cross_entropy += F.cross_entropy(logits, targets).item()

        # inference 시에는 probas 이용
        _, predicted_labels = torch.max(probas, 1)
        num_examples += targets.size(0)
        correct_pred += (predicted_labels == targets).sum()
    return correct_pred.cpu().float()/num_examples * 100, cross_entropy/num_examples
```

① Hidden layer 노드 지정

- 추가한 hidden layer에 맞춰서, 각각의 노드 개수 지정

② 학습률 및 에포크 지정

감사합니다.