02

딥러닝

다층 퍼셉트론과 역전파

방송대 컴퓨터과학과 이병래 교수

학습목차

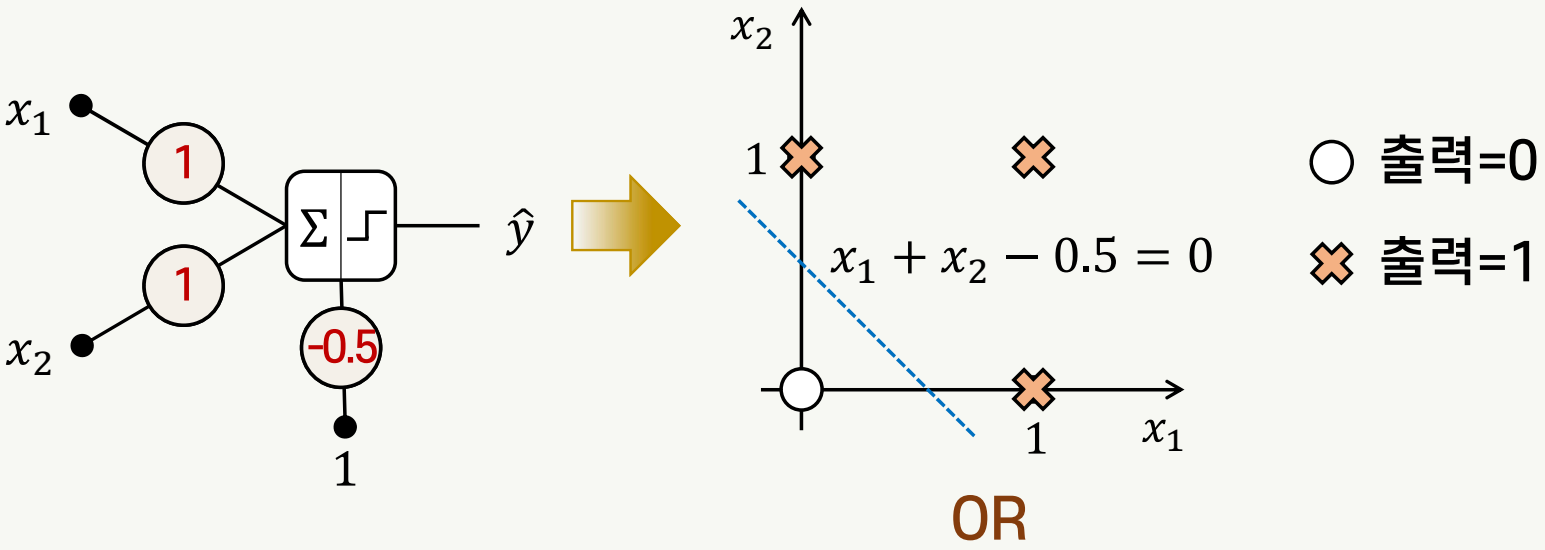
1. 다층 퍼셉트론의 개념
2. 역전파 학습
3. 실습: 역전파를 이용한 다층 퍼셉트론 학습
4. 다중 클래스 분류를 위한 다층 퍼셉트론 학습

01

다층 퍼셉트론의 개념

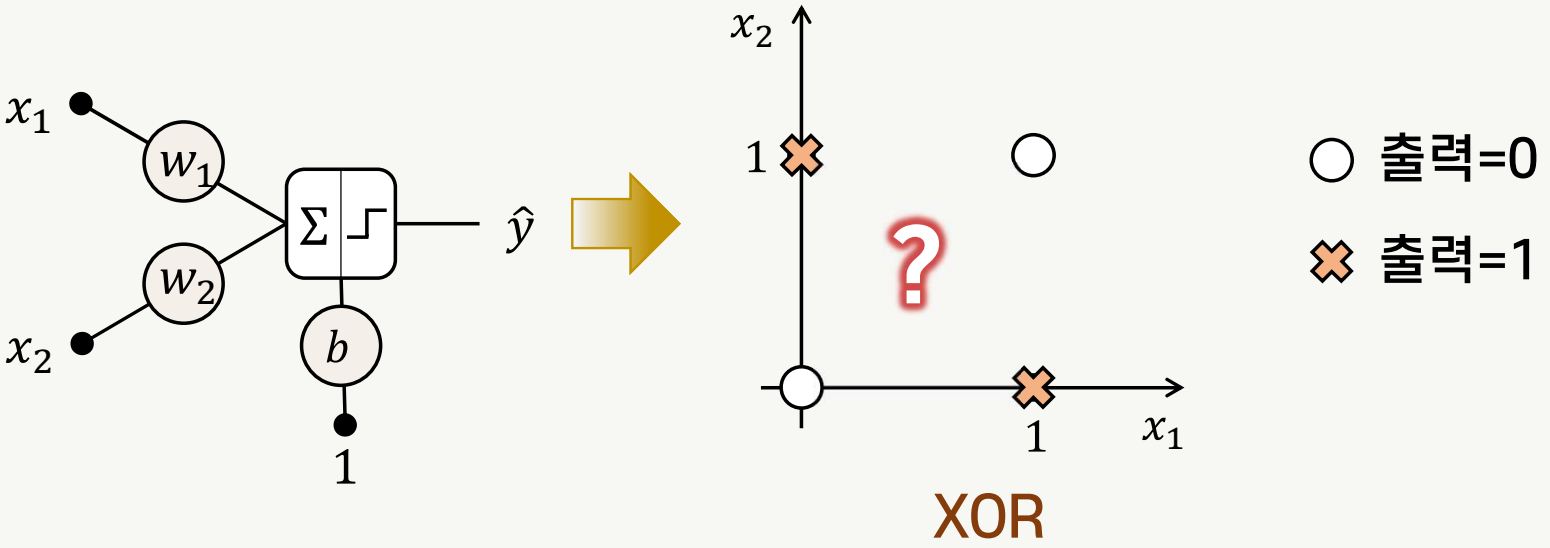
1. 단층 퍼셉트론의 한계

- 선형함수로 표현되는 결정경계로 국한되는 학습 능력



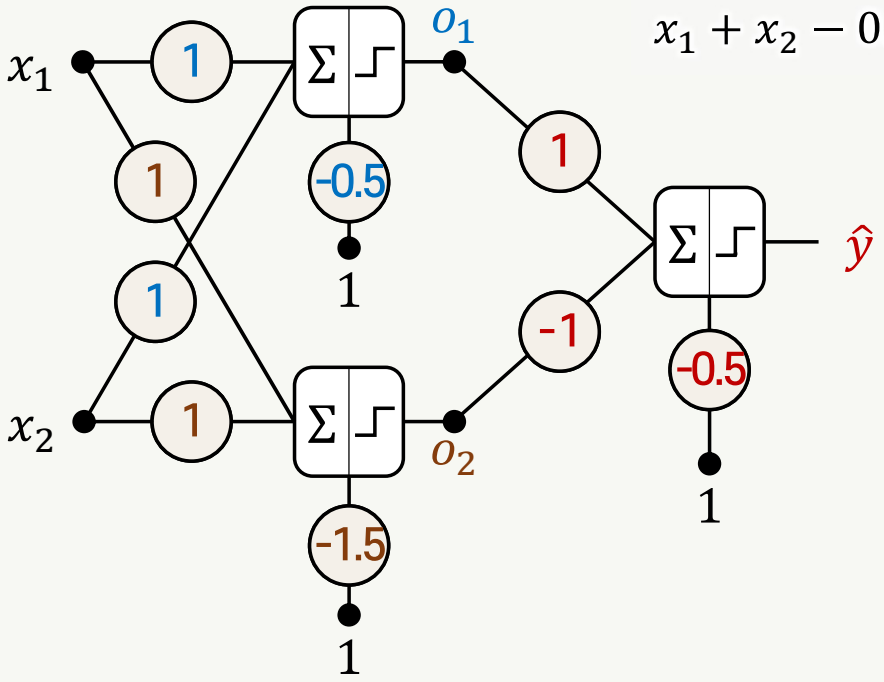
1. 단층 퍼셉트론의 한계

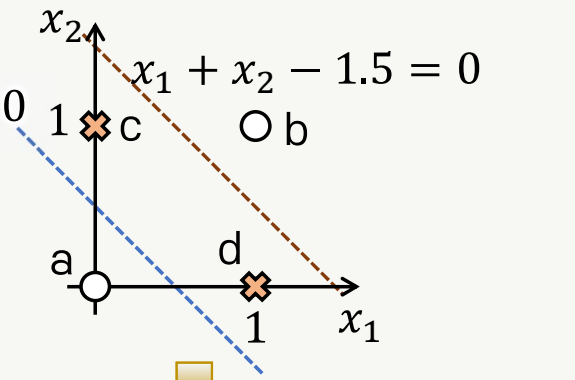
- 선형함수로 표현되는 결정경계로 국한되는 학습 능력

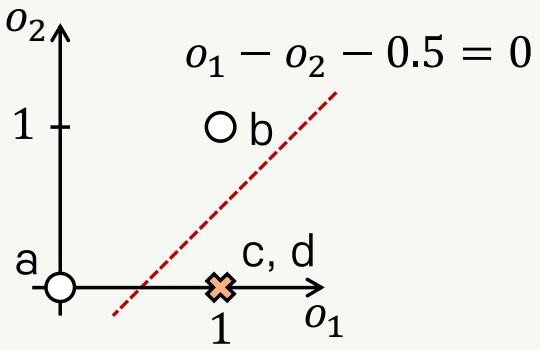


2. 다층 퍼셉트론을 이용한 문제 해결

- 추가 층을 이용하면 다양한 경계의 표현이 가능함







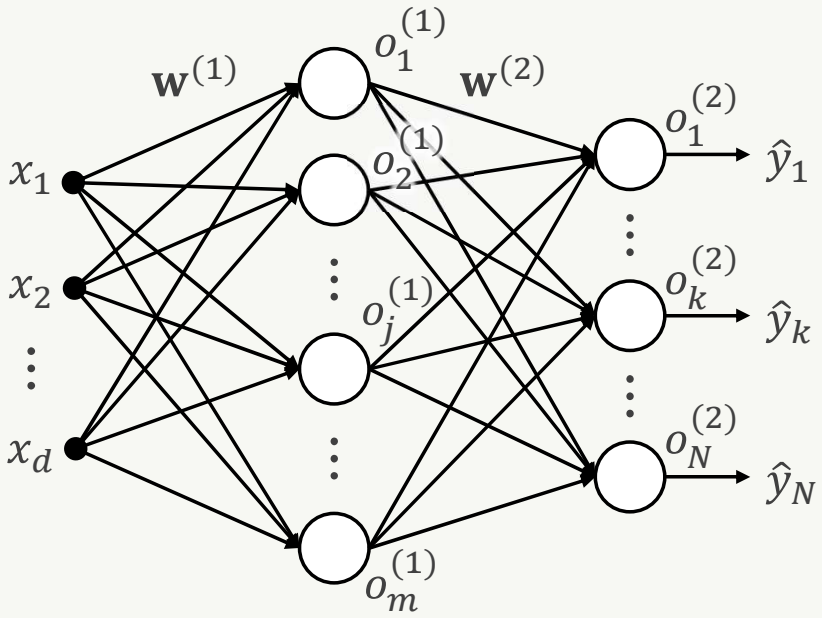
레이블이 제공되지 않는 내부 층의 학습 알고리즘이 필요함

02

역전파 학습

1. 다층 퍼셉트론(Multi-Layer Perceptron)

- 입력층과 출력층 사이에 1개 이상의 층이 있는 피드포워드 신경망
  - 은닉층(hidden layer) : 직접적인 레이블이 제공되지 않음



1. 다층 퍼셉트론(Multi-Layer Perceptron)

역전파(backpropagation)

- 1974년 Paul Werbos, 1986년 David Rumelhart 등이 발표한 다층 피드포워드 신경망의 학습 알고리즘
- 지도학습 방식
- Rumelhart의 논문에서는 뉴런의 활성함수로 시그모이드 함수 사용

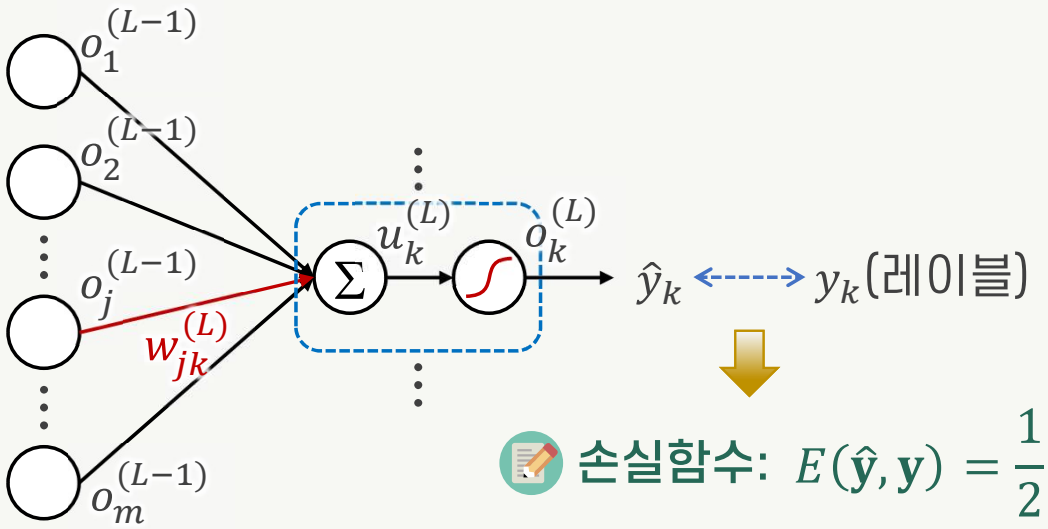
$$\varphi(u) = \frac{1}{1 + e^{-u}}$$

모든 u에 대해 미분을 구할 수 있음

$$\frac{d\varphi(u)}{du} = \varphi(u)\{1 - \varphi(u)\}$$

2. 역전파 알고리즘

출력층 연결 가중치의 학습 - k번째 출력 뉴런



$\hat{y}_k$ <--------> y_k(레이블)

손실함수: $E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$

은닉층$(L-1)$ 출력층(L)

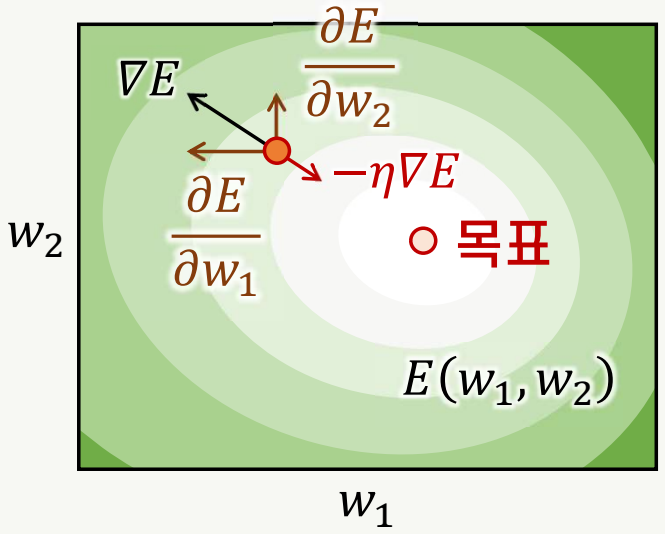
$$\hat{y}_k = \varphi\left(u_k^{(L)}\right),$$

$$u_k^{(L)} = o_1^{(L-1)}w_{1k}^{(L)} + o_2^{(L-1)}w_{2k}^{(L)} + \cdots + o_j^{(L-1)}w_{jk}^{(L)} + \cdots + o_m^{(L-1)}w_{mk}^{(L)} + b_k^{(L)}$$

2. 역전파 알고리즘

출력층 연결 가중치의 학습 - k번째 출력 뉴런

- 경사 하강법을 이용한 최적화



체인 룰(chain rule) 활용 : $y = f(u)$이고, $u = g(x)$라면

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

2. 역전파 알고리즘

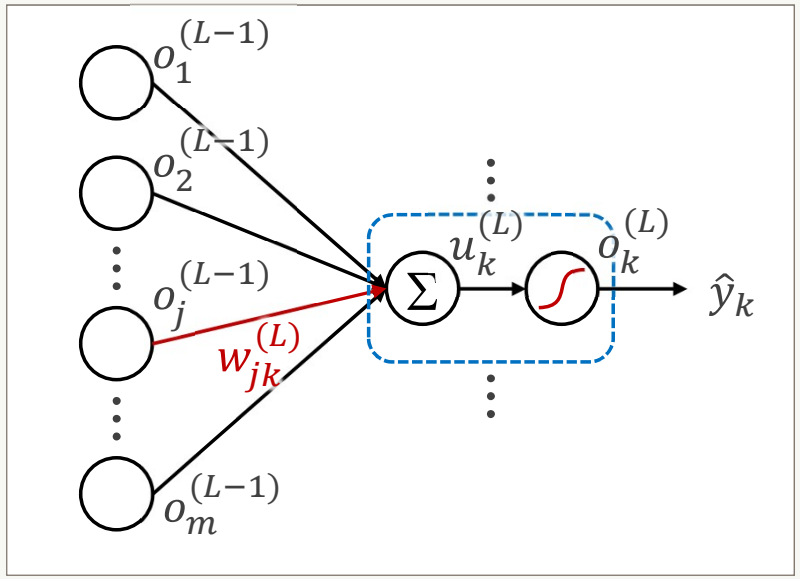
출력층 연결 가중치의 학습 - k번째 출력 뉴런

경사 하강법을 이용한 최적화

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}\sum_{p=1}^{N}(\hat{y}_p - y_p)^2$$

$$\frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial w_{jk}^{(L)}} = \frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial u_k^{(L)}} \cdot \frac{\partial u_k^{(L)}}{\partial w_{jk}^{(L)}}$$

$\delta_k^{(L)}$ $o_j^{(L-1)}$



$$\delta_k^{(L)} = \frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial u_k^{(L)}} = \hat{y}_k(1-\hat{y}_k)(\hat{y}_k - y_k)$$

2. 역전파 알고리즘

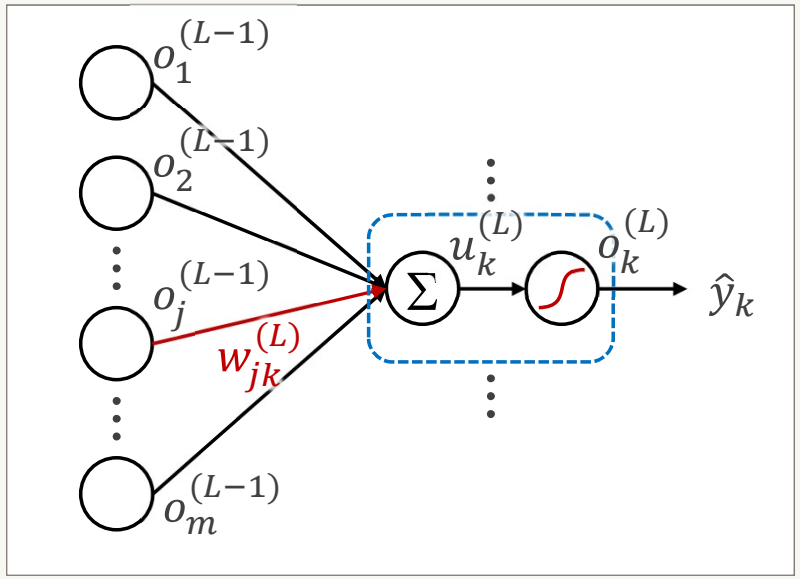
출력층 연결 가중치의 학습 - k번째 출력 뉴런

- 경사 하강법을 이용한 최적화

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}\sum_{p=1}^{N}(\hat{y}_p - y_p)^2$$

$$\frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial w_{jk}^{(L)}} = \delta_k^{(L)} o_j^{(L-1)},$$

$$\delta_k^{(L)} = \hat{y}_k(1-\hat{y}_k)(\hat{y}_k - y_k)$$



2. 역전파 알고리즘

출력층 연결 가중치의 학습 - k번째 출력 뉴런

경사 하강법을 이용한 최적화

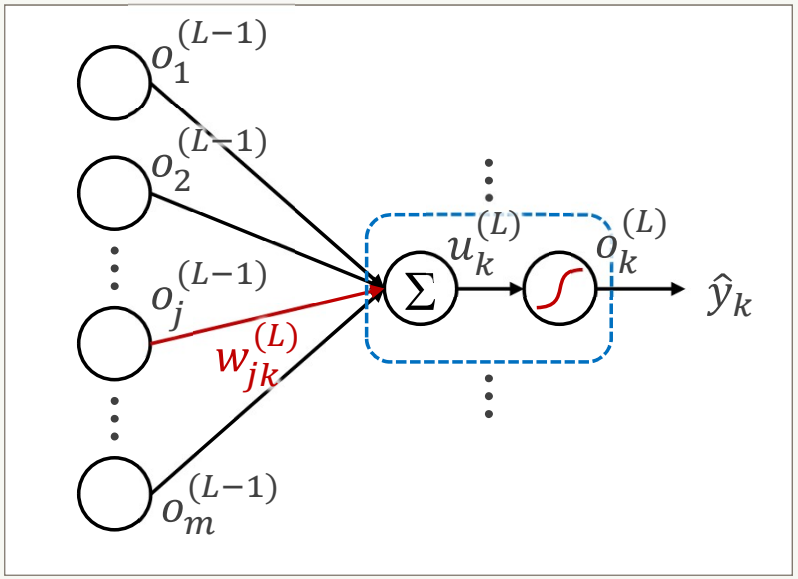
$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}\sum_{p=1}^{N}(\hat{y}_p - y_p)^2$$

- 가중치 업데이트

$$\Delta w_{jk}^{(L)} = \eta \delta_k^{(L)} o_j^{(L-1)}$$

$$\Delta b_k^{(L)} = \eta \delta_k^{(L)}$$

$$\delta_k^{(L)} = \hat{y}_k(1-\hat{y}_k)(\hat{y}_k - y_k)$$

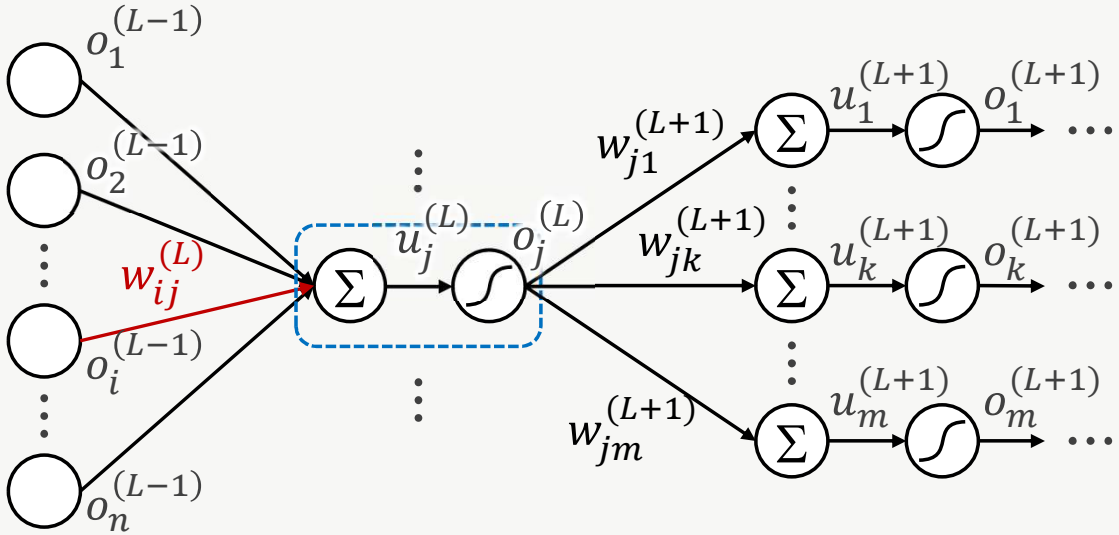


$$w_{jk}^{(L)}(t+1) = w_{jk}^{(L)}(t) - \Delta w_{jk}^{(L)}(t)$$

$$b_k^{(L)}(t+1) = b_k^{(L)}(t) - \Delta b_k^{(L)}(t)$$

2. 역전파 알고리즘

은닉층 연결 가중치의 학습 - j번째 은닉층 뉴런



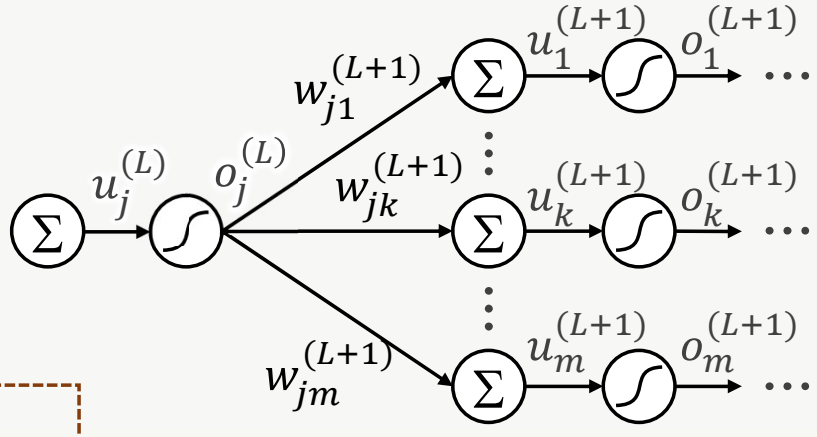
은닉층$(L-1)$ 은닉층(L) 은닉층 또는 출력층$(L+1)$

$$\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial w_{ij}^{(L)}} = \underbrace{\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial u_j^{(L)}}}_{\delta_j^{(L)}} \cdot \underbrace{\frac{\partial u_j^{(L)}}{\partial w_{ij}^{(L)}}}_{o_i^{(L-1)}}$$

2. 역전파 알고리즘

은닉층 연결 가중치의 학습 - j번째 은닉층 뉴런

$$\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial w_{ij}^{(L)}} = \underbrace{\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial u_j^{(L)}}}_{\delta_j^{(L)}} \cdot \underbrace{\frac{\partial u_j^{(L)}}{\partial w_{ij}^{(L)}}}_{o_i^{(L-1)}}$$



$$\delta_j^{(L)} = \frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial o_j^{(L)}} \cdot \frac{\partial o_j^{(L)}}{\partial u_j^{(L)}}$$

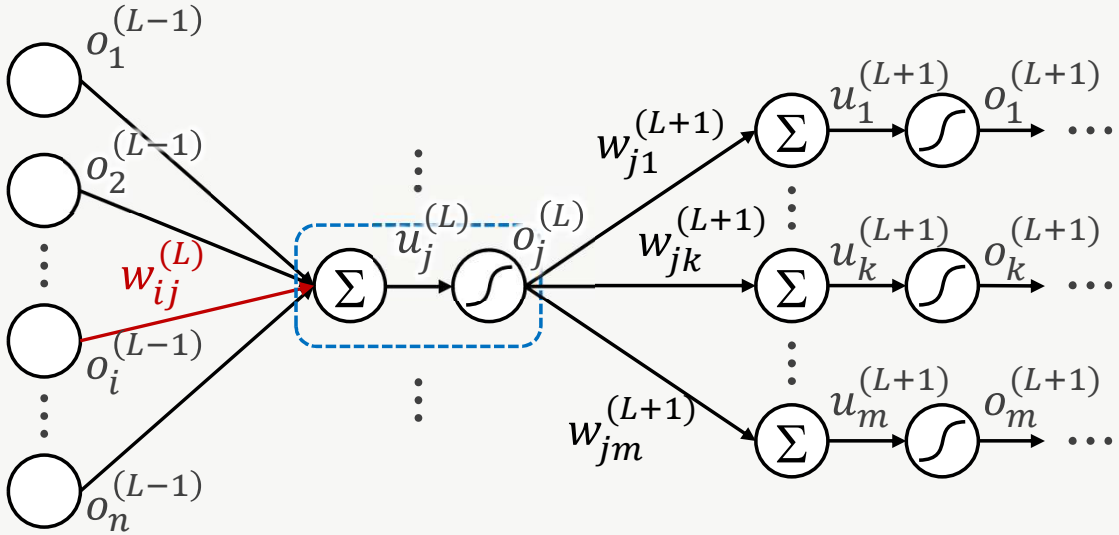
$$\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial o_j^{(L)}} = \sum_{k=1}^{m} \frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial u_k^{(L+1)}} \cdot \frac{\partial u_k^{(L+1)}}{\partial o_j^{(L)}}$$

$$= \sum_{k=1}^{m} \delta_k^{(L+1)} \cdot w_{jk}^{(L+1)}$$

$$\frac{\partial o_j^{(L)}}{\partial u_j^{(L)}} = o_j^{(L)}\left(1 - o_j^{(L)}\right)$$

2. 역전파 알고리즘

은닉층 연결 가중치의 학습 - j번째 은닉층 뉴런



은닉층$(L-1)$ 은닉층(L) 은닉층 또는 출력층$(L+1)$

$$\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial w_{ij}^{(L)}} = \delta_j^{(L)} o_i^{(L-1)}, \quad \delta_j^{(L)} = o_j^{(L)}\left(1-o_j^{(L)}\right)\left(\sum_{k=1}^{m}\delta_k^{(L+1)}\cdot w_{jk}^{(L+1)}\right)$$

2. 역전파 알고리즘

은닉층 연결 가중치의 학습 - j번째 은닉층 뉴런

$$\frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial w_{ij}^{(L)}} = \delta_j^{(L)} o_i^{(L-1)}, \quad \delta_j^{(L)} = o_j^{(L)} \left(1 - o_j^{(L)}\right) \left(\sum_{k=1}^{m} \delta_k^{(L+1)} \cdot w_{jk}^{(L+1)} \right)$$

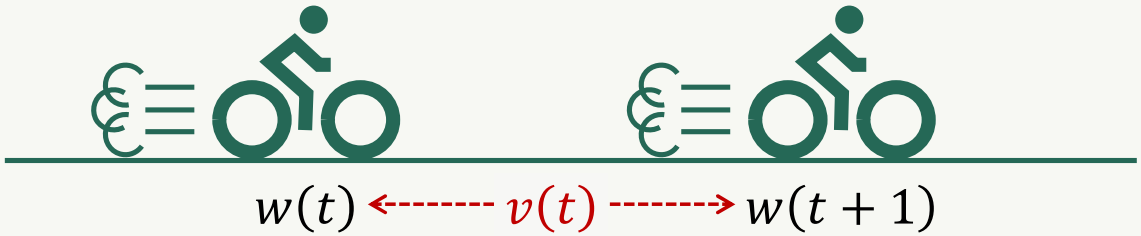
$$\Delta w_{ij}^{(L)} = \eta \delta_j^{(L)} o_i^{(L-1)}$$
$$\Delta b_j^{(L)} = \eta \delta_j^{(L)}$$

$$w_{ij}^{(L)}(t+1) = w_{ij}^{(L)}(t) - \Delta w_{ij}^{(L)}(t)$$
$$b_j^{(L)}(t+1) = b_j^{(L)}(t) - \Delta b_j^{(L)}(t)$$

3. 모멘텀을 이용한 학습

가속 기법을 이용한 경사 하강법의 개선

- w의 변화량을 '속도(velocity)'라는 개념으로 봄



$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mathbf{v}(t)$$

- 이전 시점의 속도를 모멘텀 m의 비율로 반영함

$$\mathbf{v}(t) = m\mathbf{v}(t-1) - \Delta\mathbf{w}(t), \quad \Delta\mathbf{w}(t) = \eta\frac{\partial E}{\partial \mathbf{w}}$$

03

실습: 역전파를 이용한 다층 퍼셉트론 학습

1. 붓꽃 식별을 위한 역전파 모델의 구현

시그모이드 활성함수

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

2-1 [3] 활성함수 - 시그모이드

```
1  def sigmoid(x):
2      ''' x : numpy array '''
3      return 1 / (1 + np.exp(-x))
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

손실함수

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}\sum_{p=1}^{N}(\hat{y}_p - y_p)^2$$

$$\mathbf{e} = \hat{\mathbf{y}} - \mathbf{y}$$

$$loss = \mathbf{e} \cdot \mathbf{e}$$

2-1 [4] 손실함수 - mse, cross entropy

```
1  def loss_mse(y, y_hat):
2      loss = 0.0
3      for i in range(len(y)):
4          err = y_hat[i] - y[i]
5          loss += np.dot(err, err)
6      return loss / len(y)
7  ......
```

학습 중 모델 손실 변화의 확인을 위한 것으로, MLP 학습에 직접 사용되지는 않음

1. 붓꽃 식별을 위한 역전파 모델의 구현

Dense 클래스

- 완전연결층을 구성하기 위한 클래스
- 인스턴스 변수
  - `self.nIn, self.nOut` : 완전연결층의 입력 및 출력의 수
  - `self.w, self.b` : 가중치와 바이어스
  - `self.activation` : 활성함수
  - `self.dE_du` : $\partial E/\partial u$를 계산하는 함수
  - `self.do_du` : $\partial o/\partial u$를 계산하는 함수
  - `self.velocity_w, self.velocity_b` : 모멘텀을 적용한 가중치 변화량
  - `self.in_vec, self.out_vec` : 역전파를 위한 층의 입력과 출력 보관

1. 붓꽃 식별을 위한 역전파 모델의 구현

Dense 클래스

메소드 목록

- `__init__(self, nIn, nOut, activation='sigmoid', loss='mse')` : 완전연결층의 초기화
- `output(self, X)` : 입력 X에 대한 출력 계산
- `gd(self, dw, db, momentum=0)` : 경사 하강법에 따라 w 및 b 갱신
- `dE_du_sigmoid_mse(self, y)` : 손실함수가 MSE일 때 $\partial E/\partial u$ 계산
- `dE_du_sigmoid_ce(self, y)` : 손실함수가 CE일 때 $\partial E/\partial u$ 계산
- `do_du_sigmoid(self)` : 시그모이드 함수에 대한 $\partial o/\partial u$ 계산

1. 붓꽃 식별을 위한 역전파 모델의 구현

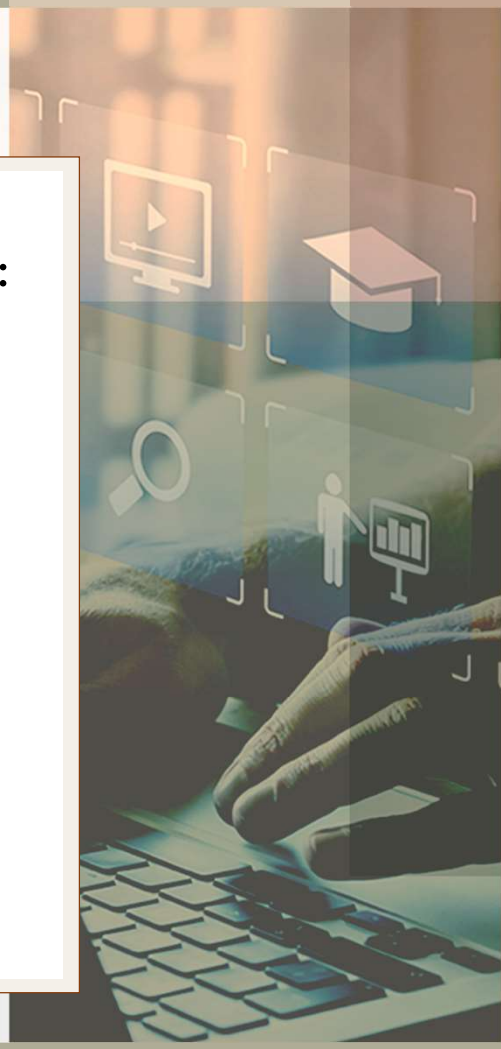
2-1 [5] Dense 클래스 - 완전연결층

```
1  class Dense():
2      def __init__(self, nIn, nOut, activation='sigmoid', loss='mse'):
3          self.nIn = nIn          # 입력의 수
4          self.nOut = nOut        # 출력의 수
5          # 가중치(w)와 바이어스(b)를 He normal 방식으로 초기화
6          rnd = np.random.default_rng()
7          self.w = rnd.normal(scale = np.sqrt(2.0 / self.nIn),
8                              size = (self.nOut, self.nIn)).astype(np.float32)
9          self.b = rnd.normal(scale = np.sqrt(2.0 / self.nIn),
10                             size = self.nOut).astype(np.float32)
11         # 활성함수 설정
12         if activation == 'sigmoid':
13             ......
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [5] Dense 클래스 - 완전연결층

```
1   class Dense():
2       def __init__(self, nIn, nOut, activation='sigmoid', loss='mse'):
3           self.nIn = nIn          # 입력의 수
4           self.nOut = nOut        # 출력의 수
...         ......
11          # 활성함수 설정
12          if activation == 'sigmoid':
13              self.activation = sigmoid
14              if loss == 'ce':    self.dE_du = self.dE_du_sigmoid_ce
15              else:               self.dE_du = self.dE_du_sigmoid_mse
16              self.do_du = self.do_du_sigmoid
17          # 모멘텀을 적용하기 위한 속도의 초깃값 설정
18          self.velocity_w, self.velocity_b = 0.0, 0.0
```



1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [5] Dense 클래스 - 완전연결층

```
1   class Dense():
...     ......
20      # 입력 x에 대한 출력 계산
21      def output(self, X):
22          self.in_vec = X          # BP 학습을 위해 입력 보관
23          # 입력의 가중 합 계산
24          u = np.array([np.dot(self.w[i], X) + self.b[i]
25                        for i in range(self.nOut)], dtype=np.float32)
26          # 활성함수를 적용한 출력 계산
27          self.out_vec = self.activation(u)  # BP 학습을 위해 출력 보관
28          return self.out_vec
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [5] Dense 클래스 - 완전연결층

```
 1   class Dense():
...      ......
30       # 경사 하강법에 따라 w 및 b 갱신
31       def gd(self, dw, db, momentum=0):
32           self.velocity_w = self.velocity_w * momentum - dw
33           self.velocity_b = self.velocity_b * momentum - db
34           self.w += self.velocity_w
35           self.b += self.velocity_b
36
37       def dE_du_sigmoid_mse(self, y):
38           return (self.out_vec - y) * self.do_du_sigmoid()
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [5] Dense 클래스 - 완전연결층

```
 1  class Dense():
...      .......
37      def dE_du_sigmoid_mse(self, y):
38          return (self.out_vec - y) * self.do_du_sigmoid()
39
40      def dE_du_sigmoid_ce(self, y):
41          return self.out_vec - y
42
43      def do_du_sigmoid(self):
44          return self.out_vec * (1 - self.out_vec)
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

BP_Model 클래스

- 역전파 학습을 하는 피드포워드 모델 클래스
- 인스턴스 변수
  - `self.nLayers` : 모델의 층 수
  - `self.layers` : 모델을 구성하는 완전연결층의 배열
  - `self.ohe` : 레이블을 one-hot encoding한 벡터의 배열
  - `self.loss` : 손실함수

1. 붓꽃 식별을 위한 역전파 모델의 구현

BP_Model 클래스

메소드 목록

- `__init__(self, nUnitLst, loss='mse',`
  `activation_h='sigmoid', activation_o='sigmoid')` :
  역전파 모델의 초기화
- `predict(self, X)` : 입력 표본의 배열 X에 대한 출력을 추론함
- `fit(self, X, y, N, epochs, eta=0.01, momentum=0)` : N개의 입력 표본 배열 X와 레이블 배열 y를 이용하여 모델을 훈련함

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
1  class BP_Model():
2      def __init__(self, nUnitLst, loss='mse',
3                   activation_h='sigmoid', activation_o='sigmoid'):
4          layers = []
5          self.nLayers = len(nUnitLst) - 1
6          # 은닉층 구성
7          for i in range(self.nLayers - 1):
8              layers.append(Dense(nUnitLst[i], nUnitLst[i+1],
9                                  activation=activation_h, loss=loss))
10         # 출력층 구성
11         layers.append(Dense(nUnitLst[self.nLayers-1],
12           nUnitLst[self.nLayers], activation=activation_o, loss=loss))
13             ......
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
class BP_Model():
    def __init__(self, nUnitLst, loss='mse',
                 activation_h='sigmoid', activation_o='sigmoid'):
        .......
        # 출력층 구성
        layers.append(Dense(nUnitLst[self.nLayers-1],
           nUnitLst[self.nLayers], activation=activation_o, loss=loss))
        self.layers = np.array(layers, dtype=object)
        self.ohe = np.identity(nUnitLst[-1])
        if loss == 'ce':
            self.loss = loss_ce
        else:
            self.loss = loss_mse
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
 1  class BP_Model():
...     ......
20      def predict(self, x):
21          res = []
22          for j in range(len(x)):
23              xx = x[j]
24              for i in range(self.nLayers):
25                  xx = self.layers[i].output(xx)
26              res.append(xx)
27          return np.array(res)
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
 1  class BP_Model():
...      ......
29       def fit(self, X, y, N, epochs, eta=0.01, momentum=0):
30           # 학습표본의 인덱스를 무작위 순서로 섞음
31           idx = list(range(N))
32           np.random.shuffle(idx)
33           X = np.array([X[idx[i]] for i in range(N)])
34           if self.layers[self.nLayers-1].nOut == 1:
35               y = np.array([[y[idx[i]]] for i in range(N)])
36           else:
37               y = np.array([self.ohe[y[idx[i]]] for i in range(N)])
38
39           f = 'Epochs = {:4d}    Loss = {:8.5f}'
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
 1  class BP_Model():
...     ......
29      def fit(self, X, y, N, epochs, eta=0.01, momentum=0):
...         ......
39          f = 'Epochs = {:4d}    Loss = {:8.5f}'
40          # w와 b의 변화량을 저장할 수 있게 준비함
41          dw, db = [], []
42          for i in range(self.nLayers):
43              dw.append(np.zeros((self.layers[i].nOut,
44                                  self.layers[i].nIn), dtype=np.float32))
45              db.append(np.zeros(self.layers[i].nOut, dtype=np.float32))
46          for n in range(epochs):
47              ......
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
 1  class BP_Model():
...     ......
29      def fit(self, X, y, N, epochs, eta=0.01, momentum=0):
...         ......
46          for n in range(epochs):
47              for m in range(N):
48                  # output layer
49                  iCurrLayer = self.nLayers - 1
50                  currLayer = self.layers[iCurrLayer]
51                  self.predict([X[m]])
52                  delta = currLayer.dE_du(y[m])
53                  du_dw = currLayer.in_vec
54                  ......
```

$$\frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial w_{jk}^{(L)}} = \frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial u_k^{(L)}} \cdot \frac{\partial u_k^{(L)}}{\partial w_{jk}^{(L)}}$$

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
 1  class BP_Model():
...     ......
29      def fit(self, X, y, N, epochs, eta=0.01, momentum=0):
...         ......
46          for n in range(epochs):
47              for m in range(N):
48                  # output layer
...                 ......
54                  for j in range(currLayer.nOut):
55                      dw[iCurrLayer][j] = eta * delta[j] * du_dw
56                      db[iCurrLayer][j] = eta * delta[j]
57                  nextDelta = delta
58                  nextLayer = currLayer
```

$$\frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial w_{jk}^{(L)}} = \frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial u_k^{(L)}} \cdot \frac{\partial u_k^{(L)}}{\partial w_{jk}^{(L)}}$$

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
29      def fit(self, X, y, N, epochs, eta=0.01, momentum=0):
...         ......
46          for n in range(epochs):
47              for m in range(N):
...                 ......
60                  # hidden layers
61                  for iCurrLayer in range(self.nLayers-2, -1, -1):
62                      currLayer = self.layers[iCurrLayer]
63                      dE_do = []
64                      for nO in range(currLayer.nOut):
65                          sDeltaW = nextDelta * nextLayer.w[:, nO]
66                          dE_do.append(sDeltaW.sum())
67                      ......
```

$$\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial o_j^{(L)}} = \sum_{k=1}^{m} \frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial u_k^{(L+1)}} \cdot \frac{\partial u_k^{(L+1)}}{\partial o_j^{(L)}}$$

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
29      def fit(self, X, y, N, epochs, eta=0.01, momentum=0):
...         ......
46          for n in range(epochs):
47              for m in range(N):
...                 ......
60                  # hidden layers
61                  for iCurrLayer in range(self.nLayers-2, -1, -1):
...                     ......
67                      delta = dE_do  * currLayer.do_du()
68                      du_dw = currLayer.in_vec
69                      for j in range(currLayer.nOut):
70                          dw[iCurrLayer][j] = eta * delta[j] * du_dw
71                          db[iCurrLayer][j] = eta * delta[j]
```

$$\delta_j^{(L)} = \frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial o_j^{(L)}} \cdot \frac{\partial o_j^{(L)}}{\partial u_j^{(L)}}$$

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
29      def fit(self, X, y, N, epochs, eta=0.01, momentum=0):
...         ......
46          for n in range(epochs):
47              for m in range(N):
...                 ......
60                  # hidden layers
61                  for iCurrLayer in range(self.nLayers-2, -1, -1):
...                     ......
69                      for j in range(currLayer.nOut):
70                          dw[iCurrLayer][j] = eta * delta[j] * du_dw
71                          db[iCurrLayer][j] = eta * delta[j]
72                      nextDelta = delta
73                      nextLayer = currLayer
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [6] BP_Model : BP 학습을 하는 피드포워드 모델 클래스

```
29      def fit(self, X, y, N, epochs, eta=0.01, momentum=0):
...         ......
46          for n in range(epochs):
47              for m in range(N):
...                 ......
75                  for i in range(self.nLayers):
76                      self.layers[i].gd(dw[i], db[i], momentum)
77
78              # 학습 과정 출력
79              if n < 10 or (n+1) % 100 == 0:
80                  y_hat = self.predict(X)
81                  print(f.format(n+1, self.loss(y, y_hat)))
```

1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [8] 훈련 데이터 준비하기

```
1  nSamples = 150
2  nDim = 2
3  target = 'versicolor'          # 식별하고자 하는 붓꽃 종류 지정
4  X_tr, y_tr, labels = prepare_data(target)
```

2-1 [9] BP_Model 객체 생성 및 학습

```
1  bp_iris = BP_Model([nDim, 4, 1], loss='mse',
2                     activation_h='sigmoid', activation_o='sigmoid')
3  bp_iris.fit(X_tr, y_tr, nSamples, epochs=1000, eta=0.1, momentum=0.9)
```

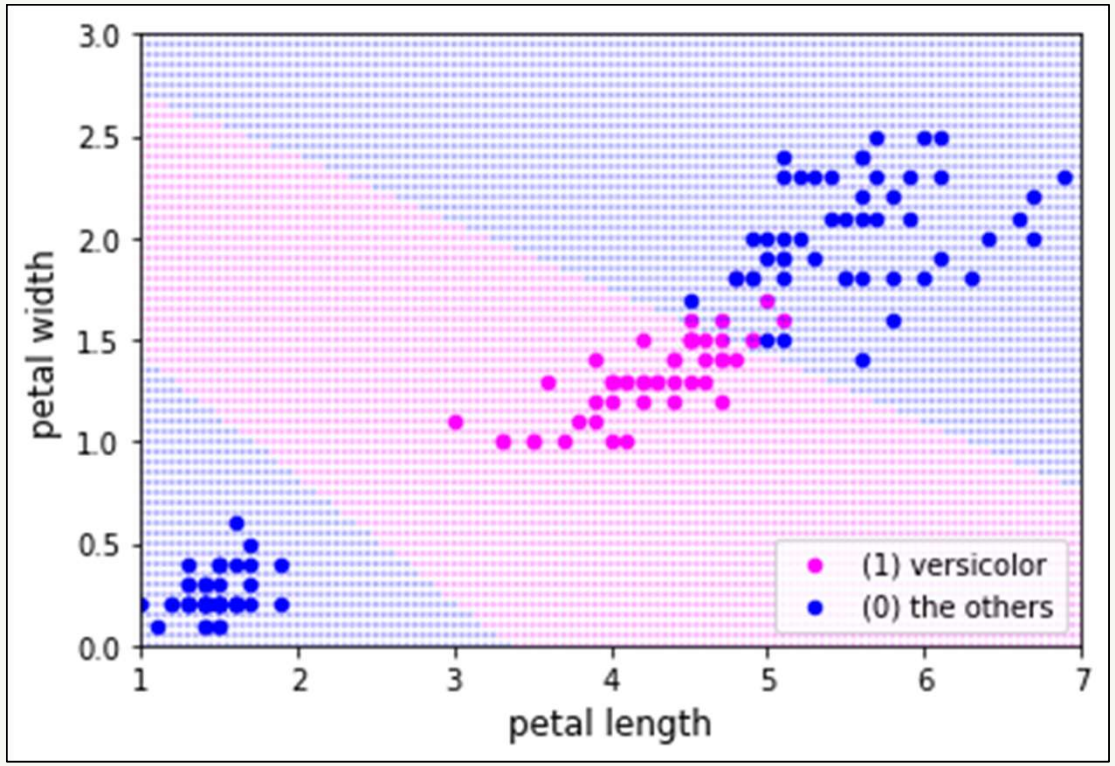
1. 붓꽃 식별을 위한 역전파 모델의 구현

2-1 [10] 특징 공간 결정 영역 시각화

```
visualize(bp_iris, X_tr, y_tr,
          multi_class=False,
          class_id=labels,
          labels=[1, 0],
          colors=['magenta', 'blue'],
          xlabel='petal length',
          ylabel='petal width')
```

2. 학습된 모델의 추론 결과

- ‘versicolor’를 식별하도록 학습된 퍼셉트론의 결정경계



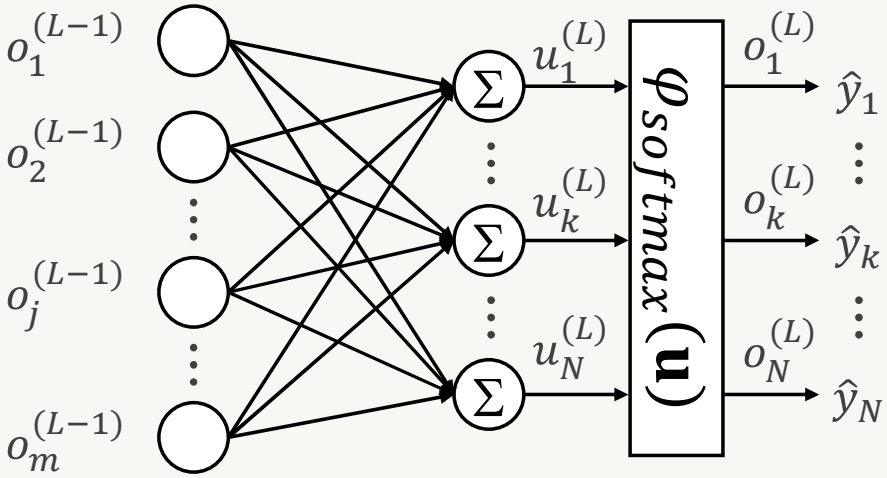
04

다중 클래스 분류를 위한 다층 퍼셉트론 학습

1. 다중 클래스 분류를 위한 활성함수 및 손실함수

소프트맥스(softmax)

- 다중 클래스 분류 문제에서 출력층의 활성함수로 적합한 함수
- 출력층이 클래스 집합의 확률분포를 나타내는 값을 출력함

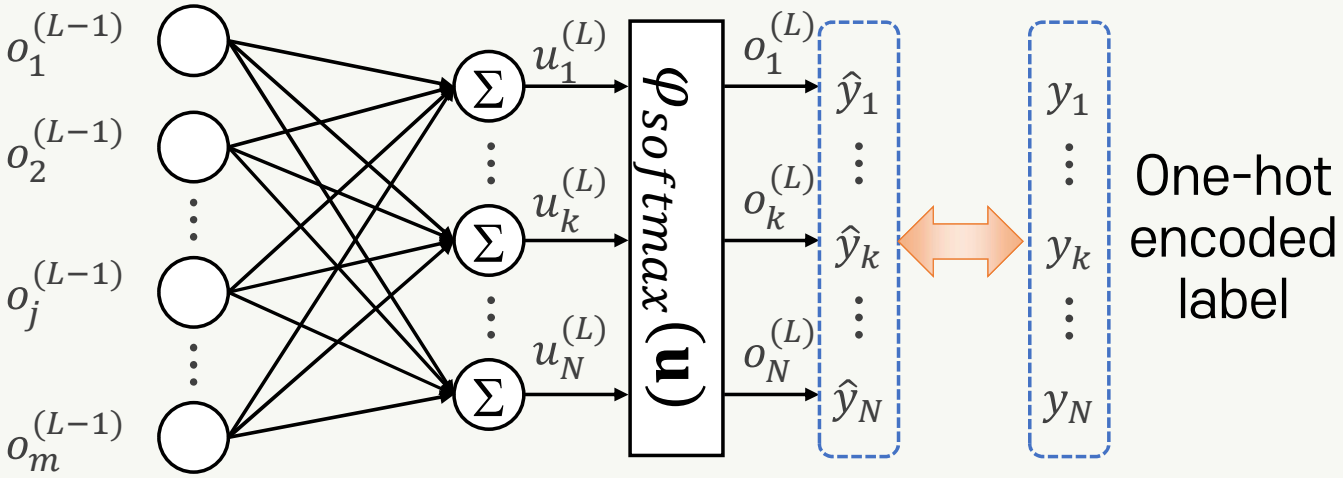


$$\varphi_{softmax}(\mathbf{u})_k = \frac{e^{u_k}}{\sum_{i=1}^{N} e^{u_i}}, \qquad k = 1,2,\cdots,N$$

1. 다중 클래스 분류를 위한 활성함수 및 손실함수

교차 엔트로피(cross entropy)

- 추정된 확률분포와 레이블에 해당되는 확률분포의 차이를 측정하는데 적합한 손실함수

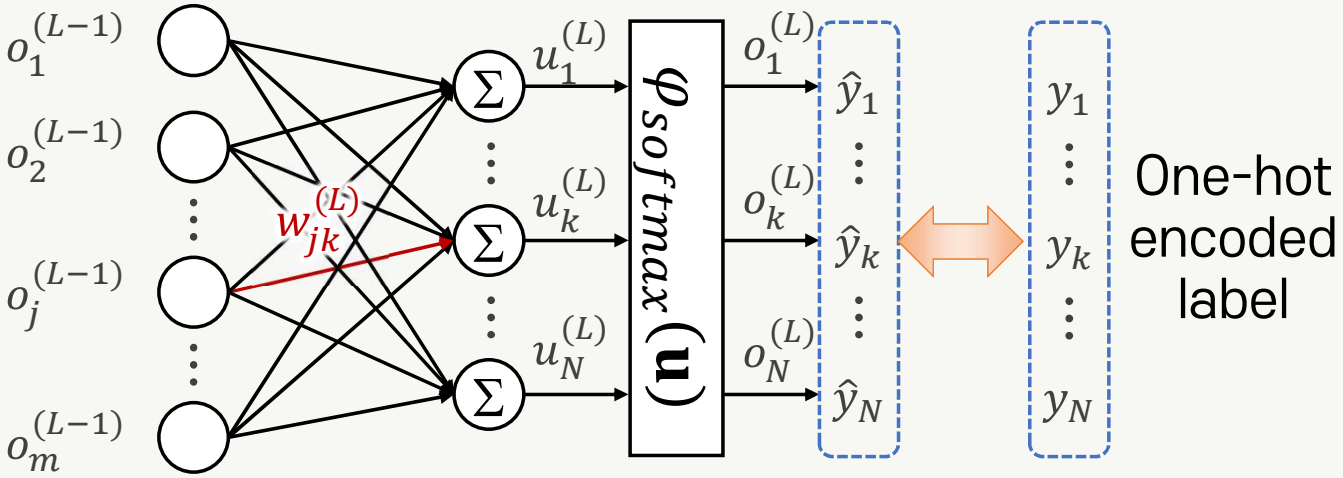


$$E(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{N} -y_i \ln \hat{y}_i, \quad \hat{y}_i = \varphi_{softmax}(\mathbf{u})_i$$

1. 다중 클래스 분류를 위한 활성함수 및 손실함수

BP 학습을 위한 출력층 노드의 δ 계산

- 소프트맥스 활성함수와 교차 엔트로피 손실함수를 사용하는 경우



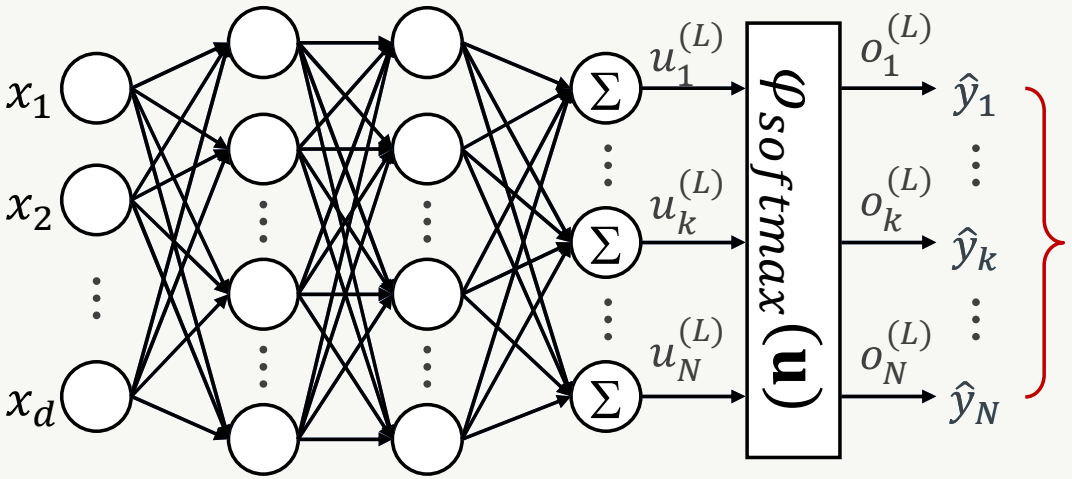
$$\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial w_{jk}^{(L)}} = \frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial u_k^{(L)}} \cdot \frac{\partial u_k^{(L)}}{\partial w_{jk}^{(L)}}$$

$\frac{\partial u_k^{(L)}}{\partial w_{jk}^{(L)}} \rightarrow o_j^{(L-1)}$

$\frac{\partial E(\hat{\mathbf{y}},\mathbf{y})}{\partial u_k^{(L)}} \rightarrow \delta_k^{(L)} = \hat{y}_k - y_k$

2. 학습된 MLP를 이용한 추론

- 소프트맥스 출력 중 가장 큰 확률에 해당되는 클래스 선택



$$class = \underset{i}{\text{argmax}}\, \hat{y}_i$$

3. 붓꽃 식별을 위한 역전파 모델의 구현 - 3개의 클래스

2-2 [2] 데이터 준비 함수 정의하기

```
1  def prepare_data():
2      iris = load_iris()                # iris data set 읽기
3      X = iris.data[:, 2:]              # 4개의 특징 중 꽃잎의 길이와 폭 선택
4      y = iris.target                   # 각 표본의 레이블
5      lbl_str = iris.target_names       # 'setosa', 'versicolor', 'virginica'
6      return X, y, lbl_str
```

3. 붓꽃 식별을 위한 역전파 모델의 구현 - 3개의 클래스

소프트맥스 활성함수 추가

$$\varphi_{softmax}(\mathbf{u})_k = \frac{e^{u_k}}{\sum_{i=1}^{N} e^{u_i}}, \qquad k = 1,2,\cdots,N$$

2-2 [3] 활성함수 - 시그모이드와 소프트맥스

```
1  def sigmoid(x):
2      ''' x : numpy array '''
3      return 1 / (1 + np.exp(-x))
4
5  def softmax(x):
6      ''' x : numpy array '''
7      o = np.exp(x)
8      return o / o.sum()
```

3. 붓꽃 식별을 위한 역전파 모델의 구현 - 3개의 클래스

2-2 [5] Dense 클래스 - 완전연결층

```
 1  class Dense():
 2      def __init__(self, nIn, nOut, activation='sigmoid', loss='mse'):
...         ......
11          # 활성함수 설정
12          if activation == 'softmax':
13              self.activation = softmax
14              self.dE_du = self.dE_du_softmax
15          else:   # sigmoid
16              self.activation = sigmoid
17              if loss == 'ce':    self.dE_du = self.dE_du_sigmoid_ce
18              else:               self.dE_du = self.dE_du_sigmoid_mse
19              self.do_du = self.do_du_sigmoid
...         ......
49      def dE_du_softmax(self, y):
50          return self.out_vec - y
```

3. 붓꽃 식별을 위한 역전파 모델의 구현 - 3개의 클래스

2-2 [8] 훈련 데이터 준비하기

```
1  nSamples = 150
2  nDim = 2
3  nClasses = 3
4  X, y, labels = prepare_data()
```

2-2 [9] BP_Model 객체 생성 및 학습

```
1  bp_iris_multi = BP_Model([nDim, 4, nClasses], loss='ce',
2                           activation_h='sigmoid', activation_o= 'softmax')
3  bp_iris_multi.fit(X, y, nSamples, epochs=1000,
4                    eta=0.01, momentum=0.9)
```

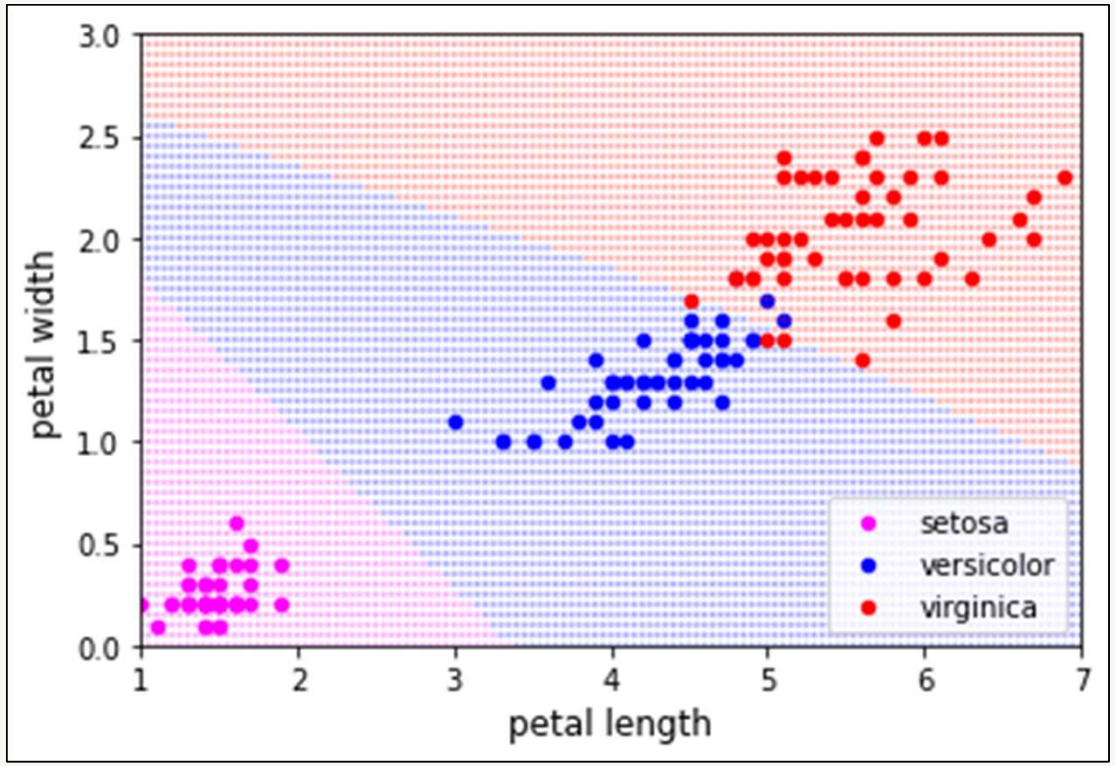
3. 붓꽃 식별을 위한 역전파 모델의 구현 - 3개의 클래스

2-2 [10] 특징 공간 결정 영역 시각화

```
visualize(bp_iris_multi, X, y,
          multi_class=True,
          class_id=labels,
          labels=[0, 1, 2],
          colors=['magenta', 'blue', 'red'],
          xlabel='petal length',
          ylabel='petal width')
```

3. 붓꽃 식별을 위한 역전파 모델의 구현 - 3개의 클래스

- 역전파 알고리즘으로 학습된 세 붓꽃 클래스의 결정경계



정리하기

- 피드포워드 신경망을 2개 이상의 층으로 구성하면 다양한 형태의 경계를 표현할 수 있다.
- 역전파 알고리즘은 지도학습 방식으로 학습하며, 미분 가능한 활성함수를 사용한다.
- 경사 하강법을 위해 학습 대상 파라미터(가중치, 바이어스 등)에 대한 손실함수의 편미분을 구한다.
- 출력층에서 입력층 방향으로 역전파 알고리즘을 적용하면 은닉층 연결 가중치에 대한 손실함수 편미분을 체인 룰을 이용하여 구할 수 있다.

정리하기

- 모멘텀의 비율로 이전 시점의 속도(w의 변화량)를 w의 업데이트에 반영하는 방식으로 경사 하강법을 개선할 수 있다.
- 3개 이상의 다중 클래스 분류를 위한 신경망 모델에서 출력층에 적합한 활성함수는 소프트맥스이다.
- 소프트맥스는 입력 벡터를 확률밀도를 나타내는 값으로 변환하여 출력한다.
- 추정된 확률분포와 레이블에 해당되는 확률분포의 차이를 측정하는 데는 교차 엔트로피가 적합하다.

다음시간안내

03

딥러닝 프레임워크