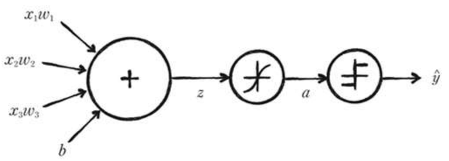
06 2개의 층을 연결합니다

- 다층 신경망

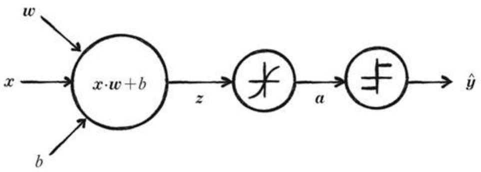
06-1 신경망 알고리즘의 벡터화

배치 경사 하강법




```
z = np.sum(x * self.w) + self.b
```

점곱(dot product), 스칼라곱(scalar product)




```
z = np.dot(x, self.w) + self.b
```

$$XW = [x_1\ x_2\ x_3]\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3$$

전체 샘플에 대해 행렬 곱셈 적용하기

$$
\boldsymbol{XW}=\begin{array}{l} \text{3개의 특성} \\ \left.\begin{array}{l}\text{m개의 샘플}\end{array}\right. \end{array}
\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} \\ & \vdots & \\ x_1^{(m)} & x_2^{(m)} & x_3^{(m)} \end{bmatrix}
\overset{\text{점 곱}}{\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}}
=\begin{bmatrix} x_1^{(1)}w_1+x_2^{(1)}w_2+x_3^{(1)}w_3 \\ x_1^{(2)}w_1+x_2^{(2)}w_2+x_3^{(2)}w_3 \\ \vdots \\ x_1^{(m)}w_1+x_2^{(m)}w_2+x_3^{(m)}w_3 \end{bmatrix}
$$

$$(m,\ n)\cdot(n,\ k)=(m,\ k)$$

```
np.dot(x, w)
```

cancer 데이터셋의 정방향 계산

배치 경사 하강법은 전체 데이터셋을 모두 계산한 다음 그레이디언트를 업데이트합니다

(364, 30)·(30, 1) = (364,1)

$$
\boldsymbol{XW}+\boldsymbol{b}=\underbrace{\begin{bmatrix} x_1^{(1)} & \cdots & x_{30}^{(1)} \\ \vdots & & \vdots \\ x_1^{(364)} & \cdots & x_{30}^{(364)} \end{bmatrix}}_{30}\Bigg\}364
\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{30} \end{bmatrix}\Bigg\}30
+\begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}\Bigg\}364
=\begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(364)} \end{bmatrix}\Bigg\}364
$$

```
def forpass(self, x):
    z = np.dot(x, self.w) + self.b
    return z
```

↑

```
def forpass(self, x):
    z = np.sum(x * self.w) + self.b
    return z
```

$$
\begin{bmatrix} x_1^{(1)} & \cdots & x_{30}^{(1)} \\ & \vdots & \\ x_1^{(364)} & \cdots & x_{30}^{(364)} \end{bmatrix}
\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{30} \end{bmatrix}+b=
\begin{bmatrix} x_1^{(1)}w_1+x_2^{(1)}w_2+\cdots+x_{30}^{(1)}w_{30} \\ \vdots \\ x_1^{(364)}w_1+x_2^{(364)}w_2+\cdots+x_{30}^{(364)}w_{30} \end{bmatrix}+b
$$

$$
=\begin{bmatrix} x_1^{(1)}w_1+x_2^{(1)}w_2+\cdots+x_{30}^{(1)}w_{30}+b \\ \vdots \\ x_1^{(364)}w_1+x_2^{(364)}w_2+\cdots+x_{30}^{(364)}w_{30}+b \end{bmatrix}
$$

cancer 데이터셋의 역방향 계산

(30, 364)·(364, 1) = (30,1)

$$
X^TE = \begin{array}{c c}
 & \overbrace{\hphantom{x_1^{(1)} \quad \cdots \quad x_1^{(364)}}}^{364} \\
30 & \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(364)} \\ x_2^{(1)} & & x_2^{(364)} \\ \vdots & & \vdots \\ x_{30}^{(364)} & & x_{30}^{(364)} \end{bmatrix}
\end{array}
\begin{array}{c c}
\begin{bmatrix} e^{(1)} \\ e^{(2)} \\ \vdots \\ e^{(364)} \end{bmatrix} & 364
\end{array}
= \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{30} \end{bmatrix}
$$

행렬의 전치
$(364, 30)^T = (30, 364)$

```
def backprop(self, x, err):
    w_grad = x * err
    b_grad = 1 * err
    return w_grad, b_grad
```

→

```
def backprop(self, x, err):
    m = len(x)
    w_grad = np.dot(x.T, err) / m        # 가중치에 대한 평균 그레이디언트를 계산합니다.
    b_grad = np.sum(err) / m             # 절편에 대한 평균 그레이디언트를 계산합니다.
    return w_grad, b_grad
```

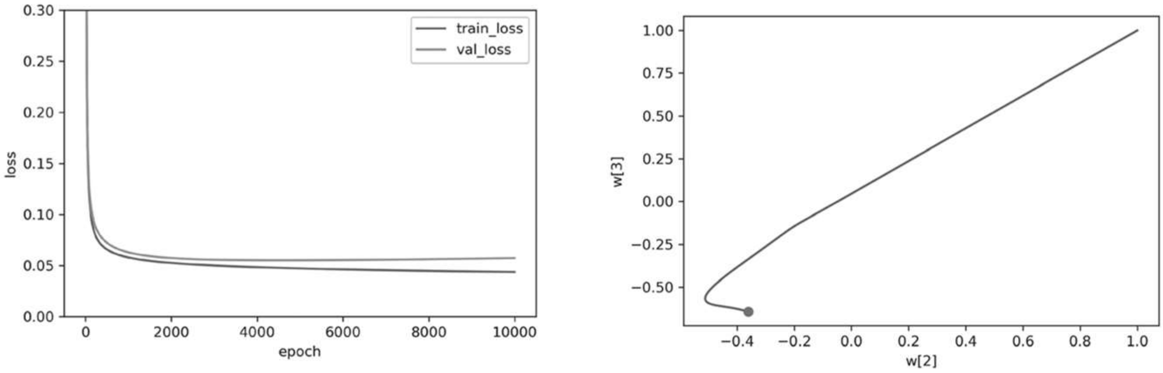
fit() 메서드 수정

```
for i in range(epochs):                         # epochs만큼 반복합니다.
    loss = 0
    indexes = np.random.permutation(np.arange(len(x)))   # 인덱스를 섞습니다.
    for i in indexes:                           # 모든 샘플에 대해 반복합니다.
        z = self.forpass(x[i])                  # 정방향 계산
        a = self.activation(z)                  # 활성화 함수 적용
        err = -(y[i] - a)                       # 오차 계산
```

↓

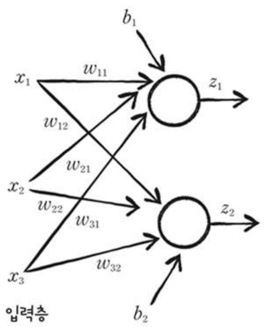
```
# epochs만큼 반복합니다.
for i in range(epochs):
    z = self.forpass(x)                # 정방향 계산을 수행합니다.
    a = self.activation(z)             # 활성화 함수를 적용합니다.
    err = -(y - a)                     # 오차를 계산합니다.
```

cancer 데이터셋에 배치 경사 하강법 적용하기



06-2 2개의 층을 가진 신경망을 구현합니다

하나의 층에 여러 개의 뉴런을 사용합니다

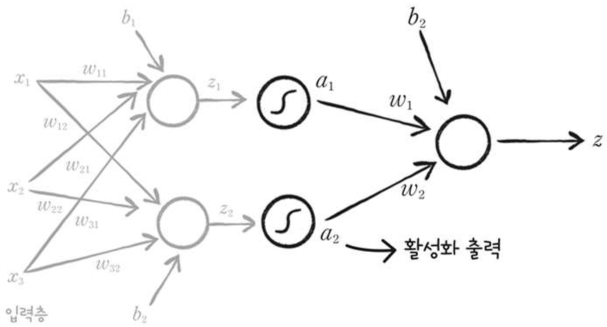


$$x_1w_{11}+x_2w_{21}+x_3w_{31}+b_1=z_1$$
$$x_1w_{12}+x_2w_{22}+x_3w_{32}+b_2=z_2$$

$$[x_1\ x_2\ x_3]\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}+[b_1\ b_2]=[z_1\ z_2]$$

전체 샘플일 경우: $\boldsymbol{XW}_1+\boldsymbol{b}_1=\boldsymbol{Z}_1$

출력을 하나로 모읍니다

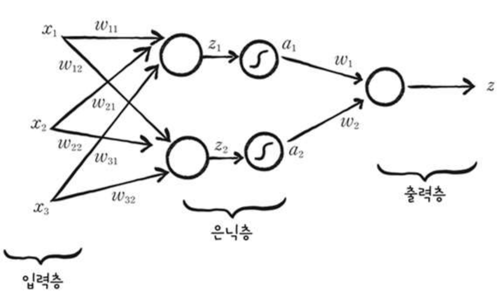


$$a_1w_1+a_2w_2+b_2=z$$

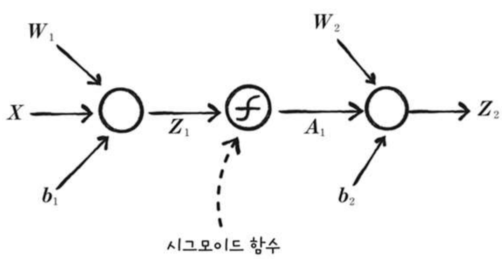
$$[a_1\ a_2]\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}+b_2=z$$

전체 샘플일 경우:$\boldsymbol{A}_1\boldsymbol{W}_2+\boldsymbol{b}_2=\boldsymbol{Z}_2$

은닉층이 추가된 신경망을 알아봅니다

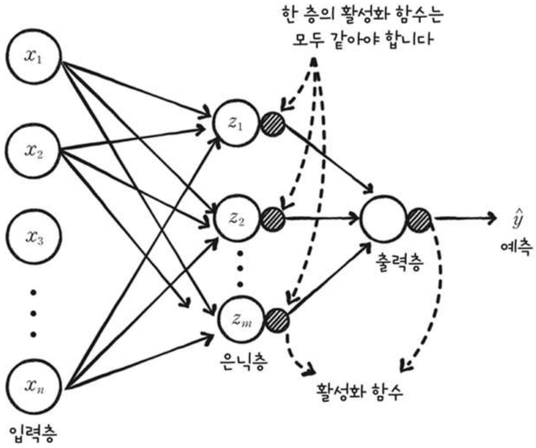


행렬로 표기

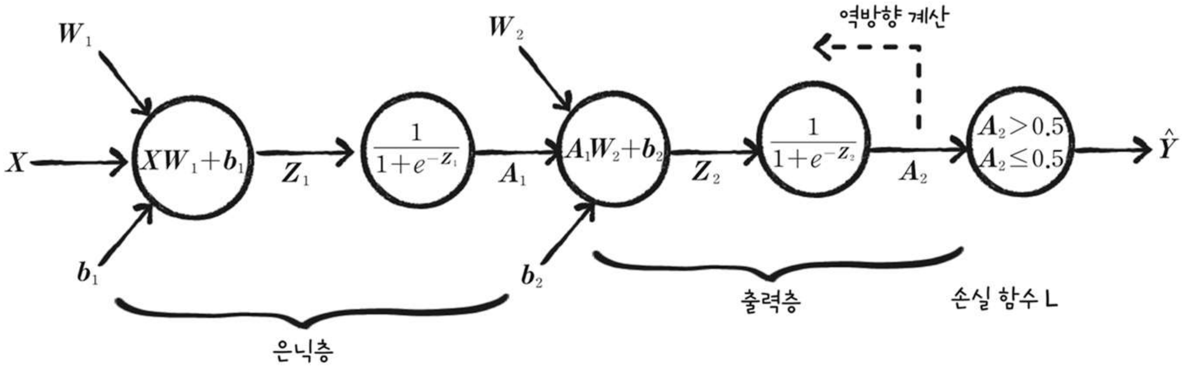


다층 신경망의 개념을 정리합니다

다층 퍼셉트론 == 완전 연결 신경망 == 밀집 신경망 == 피드 포워드 신경망



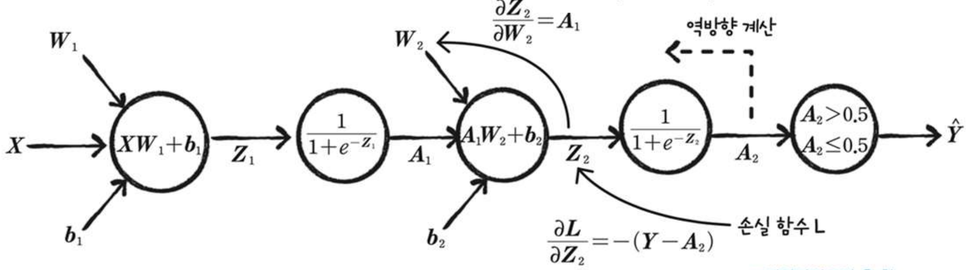
다층 신경망에 경사 하강법을 적용합니다



가중치에 대해 손실 함수를 미분합니다(출력층)

$$\frac{\partial L}{\partial W_2}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2}$$

$$=\underbrace{\begin{bmatrix}-1.37 & 0.96\\ & \vdots \\ 2.10 & -0.17\end{bmatrix}}_{m\text{개}}$$

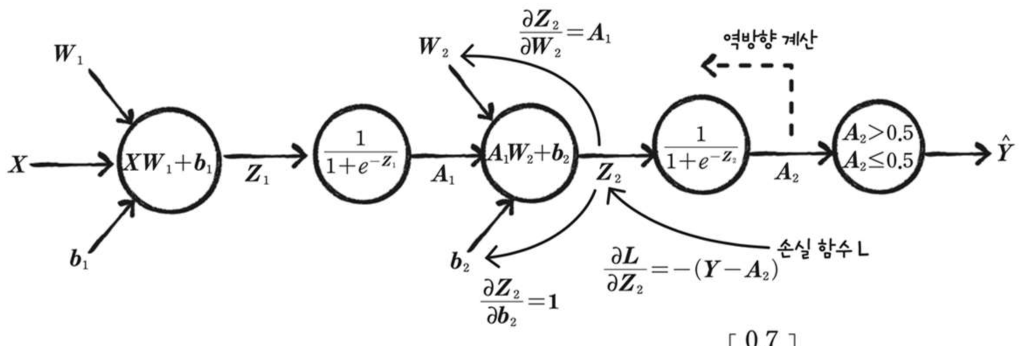


그레이디언트의 총 합

$$\frac{\partial L}{\partial W_2}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2}=A_1^T(-(Y-A_2))=\underbrace{\begin{bmatrix}-1.37 \cdots 2.10\\ 0.96 \quad -0.17\end{bmatrix}}_{m\text{개}}\begin{bmatrix}0.7\\0.3\\ \vdots\\0.6\end{bmatrix}=\begin{bmatrix}-0.12\\0.36\end{bmatrix}$$

타깃과 예측의 차이

절편에 대하여 손실 함수를 미분합니다(출력층)

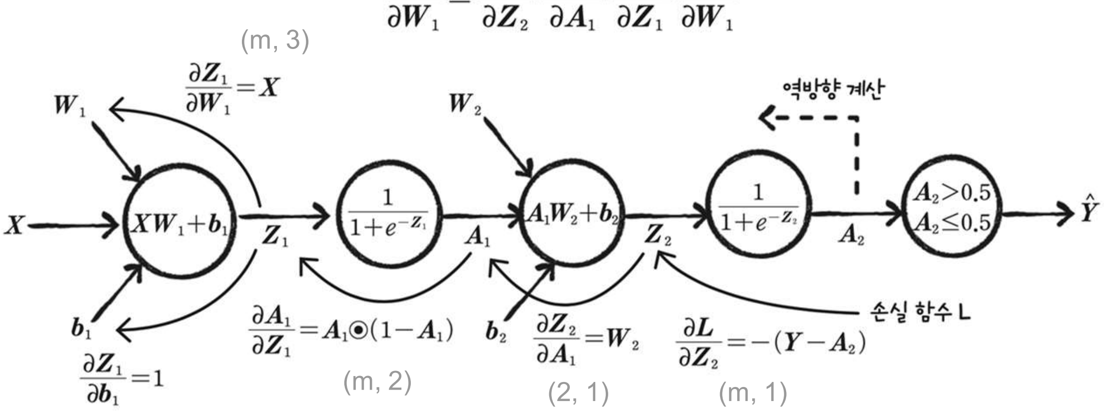


$$\frac{\partial L}{\partial b_2}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial b_2}=\mathbf{1}^T(-(Y-A_2))=[\underbrace{1 \cdots 1}_{m\text{개}}]\begin{bmatrix}0.7\\0.3\\ \vdots\\0.6\end{bmatrix}=0.18$$

타깃과 예측의 차이

가중치에 대하여 손실 함수를 미분합니다(은닉층)

$$\frac{\partial L}{\partial W_1}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial W_1}$$



$$\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}=-(Y-A_2)W_2^T\odot A_1\odot(1-A_1)$$

도함수를 곱합니다(은닉층) - 1

$$\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}=-(Y-A_2)W_2^T=\begin{bmatrix}0.7\\0.3\\\vdots\\0.6\end{bmatrix}[0.02\quad 0.16]=\left.\begin{bmatrix}0.014 & 0.112\\ \vdots & \vdots\\ 0.012 & 0.096\end{bmatrix}\right\}m\text{개}$$

(m, 1)·(1, 2) = (m, 2)

$$\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}=-(Y-A_2)W_2^T\odot A_1\odot(1-A_1)$$

$$=\begin{bmatrix}0.014 & 0.112\\ \vdots & \vdots\\ 0.012 & 0.096\end{bmatrix}\odot\begin{bmatrix}2.37 & 6.10\\ \vdots & \vdots\\ 1.81 & 4.82\end{bmatrix}\odot\left(1-\begin{bmatrix}2.37 & 6.10\\ \vdots & \vdots\\ 1.81 & 4.82\end{bmatrix}\right)$$

$$=\left.\begin{bmatrix}-0.045 & -3.48\\ \vdots & \vdots\\ -0.018 & -1.768\end{bmatrix}\right\}m\text{개}$$

(m, 1)·(1, 2)◎(m, 2) = (m, 2)

도함수를 곱합니다(은닉층) - 2

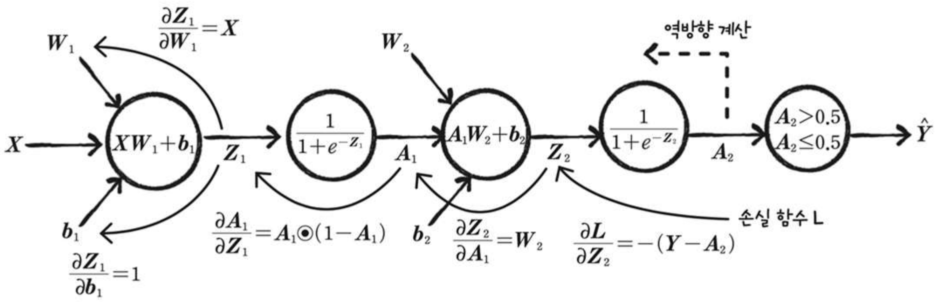
$$\frac{\partial L}{\partial W_1}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial W_1}=X^T(-(Y-A_2)W_2^T\odot A_1\odot(1-A_1))$$

$$=\overbrace{\begin{bmatrix}3&1&&4\\6&10&\cdots&8\\2&7&&1\end{bmatrix}}^{m\text{개}}\left.\begin{bmatrix}-0.045&-3.48\\&\vdots\\-0.018&-1.768\end{bmatrix}\right\}m\text{개}$$

$$=\begin{bmatrix}1.20&0.012\\-0.001&-0.3080\\0.27&0.119\end{bmatrix}$$

(3, m)·((m, 1)·(1, 2)◎(m, 2)) = (3, 2)

절편에 대하여 손실 함수를 미분하고 도함수를 곱합니다



$$\frac{\partial L}{\partial b_1}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial b_1}=\mathbf{1}^T(-(Y-A_2)W_2^T\odot A_1\odot(1-A_1))$$

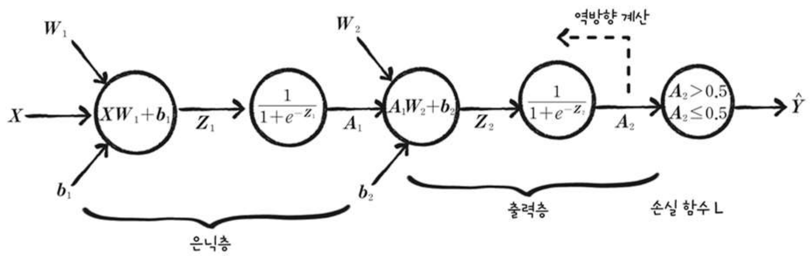
$$=\begin{bmatrix}1&1&\cdots&1\end{bmatrix}\left.\begin{bmatrix}-0.045&-3.48\\&\vdots\\-0.018&-1.768\end{bmatrix}\right\}m\text{개}$$

(1, m)·((m, 1)·(1, 2)◎(m, 2)) = (1, 2)

$$=\begin{bmatrix}0.121&-0.034\end{bmatrix}$$

2개의 층을 가진 신경망 구현하기

정방향 계산




```
def forpass(self, x):
    z1 = np.dot(x, self.w1) + self.b1          # 첫 번째 층의 선형식을 계산합니다
    self.a1 = self.activation(z1)              # 활성화 함수를 적용합니다
    z2 = np.dot(self.a1, self.w2) + self.b2    # 두 번째 층의 선형식을 계산합니다.
    return z2
```

2개의 층을 가진 신경망 구현하기

역방향 계산

```
def backprop(self, x, err):
    m = len(x)         # 샘플 개수
    # 출력층의 가중치와 절편에 대한 그레이디언트를 계산합니다.
    w2_grad = np.dot(self.a1.T, err) / m
    b2_grad = np.sum(err) / m
    # 시그모이드 함수까지 그레이디언트를 계산합니다.
    err_to_hidden = np.dot(err, self.w2.T) * self.a1 * (1 - self.a1)
    # 은닉층의 가중치와 절편에 대한 그레이디언트를 계산합니다.
    w1_grad = np.dot(x.T, err_to_hidden) / m
    b1_grad = np.sum(err_to_hidden, axis=0) / m
    return w1_grad, b1_grad, w2_grad, b2_grad
```

$$\frac{\partial L}{\partial W_2}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2}=A_1^T(\underbrace{-(Y-A_2)}_{\text{err}})$$

$$\frac{\partial L}{\partial b_2}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial b_2}=1^T(\underbrace{-(Y-A_2)}_{\text{err}})$$

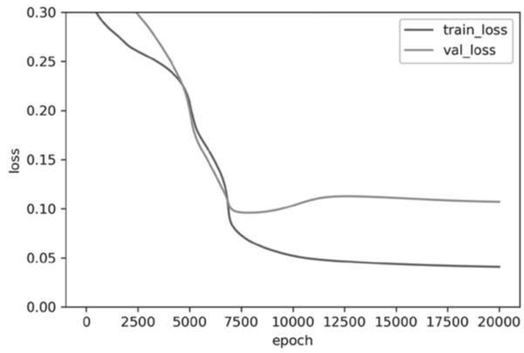
$$\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}=-(Y-A_2)W_2^T\odot A_1\odot(1-A_1)$$

$$\frac{\partial L}{\partial W_1}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial W_1}=X^T(\overbrace{-(Y-A_2)W_2^1\odot W_2^T\odot(1-A_1)}^{\text{err\_to\_hidden}})$$

$$\frac{\partial L}{\partial b_1}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial b_1}=1^T(\overbrace{-(Y-A_2)W_2^1\odot W_2^T\odot(1-A_1)}^{\text{err\_to\_hidden}})$$

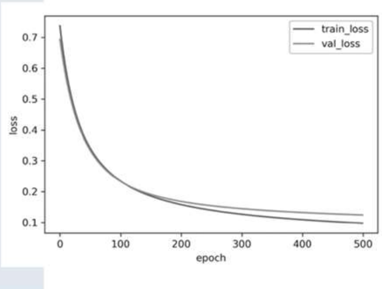
init_weights() 메서드 추가하고 훈련하기

```
def init_weights(self, n_features):
    self.w1 = np.ones((n_features, self.units))  # (특성 개수, 은닉층의 크기)
    self.b1 = np.zeros(self.units)               # 은닉층의 크기
    self.w2 = np.ones((self.units, 1))           # (은닉층의 크기, 1)
    self.b2 = 0
```



가중치 초기화 개선하기

```
class RandomInitNetwork(DualLayer):

    def init_weights(self, n_features):
        np.random.seed(42)
        self.w1 = np.random.normal(0, 1,
                                   (n_features, self.units))  # (특성 개수, 은닉층의 크기)
        self.b1 = np.zeros(self.units)                         # 은닉층의 크기
        self.w2 = np.random.normal(0, 1, (self.units, 1))      # (은닉층의 크기, 1)
        self.b2 = 0
```



06-3 미니 배치를 사용하여 모델을 훈련합니다

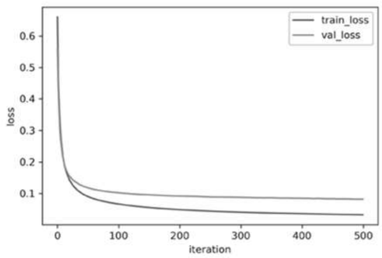
```
class MinibatchNetwork(RandomInitNetwork):

    def __init__(self, units=10, batch_size=32, learning_rate=0.1, l1=0, l2=0):
        super( ).__init__(units, learning_rate, l1, l2)
        self.batch_size = batch_size     # 배치 크기
```

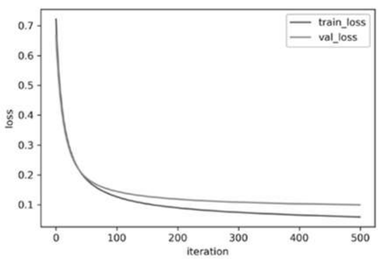
```
def fit(self, x, y, epochs=100, x_val=None, y_val=None):
    y = y.reshape(-1, 1)              # 타깃을 열 벡터로 바꿉니다.
    self.init_weights(x.shape[1])     # 은닉층과 출력층의 가중치를 초
    np.random.seed(42)
    # epochs만큼 반복합니다.
    for i in range(epochs):
        loss = 0
        # 제너레이터 함수에서 반환한 미니 배치를 순환합니다.
        for x_batch, y_batch in self.gen_batch(x, y):
            y_batch = y_batch.reshape(-1, 1)   # 타깃을 열 벡터로
            m = len(x_batch)                   # 샘플 개수를 저장
            a = self.training(x_batch, y_batch, m)
```

```
# 미니 배치 제너레이터 함수
def gen_batch(self, x, y):
    length = len(x)
    bins = length // self.batch_size        # 미니 배치 횟수
    if length % self.batch_size:
        bins += 1                           # 나누어 떨어지지 않을 때
    indexes = np.random.permutation(np.arange(len(x)))  # 인덱스를 섞습니다.
    x = x[indexes]
    y = y[indexes]
    for i in range(bins):
        start = self.batch_size * i
        end = self.batch_size * (i + 1)
        yield x[start:end], y[start:end]    # batch_size만큼 슬라이싱하여 반환합니다.
```

미니 배치 경사 하강법 훈련

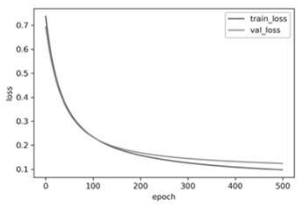
```
minibatch_net = MinibatchNetwork(l2=0.01, batch_size=32)
minibatch_net.fit(x_train_scaled, y_train, x_val=x_val_scaled, y_val=y_val,
epochs=500)
```






```
minibatch_net = MinibatchNetwork(l2=0.01, batch_size=128)
minibatch_net.fit(x_train_scaled, y_train, x_val=x_val_scaled, y_val=y_val, epochs=500)
```

배치 경사 하강법



사이킷런을 사용해 다층 신경망 훈련하기

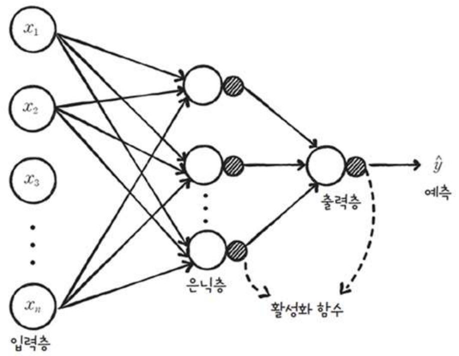
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, ), activation='logistic',
                    solver='sgd', alpha=0.01, batch_size=32,
                    learning_rate_init=0.1, max_iter=500)
```

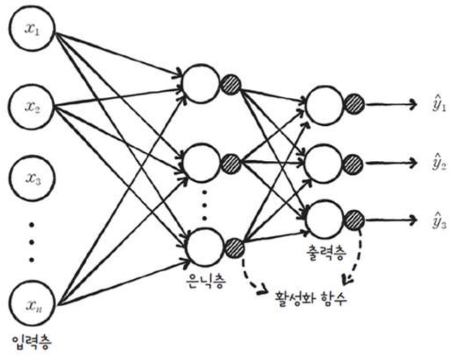
```
mlp.fit(x_train_scaled, y_train)
mlp.score(x_val_scaled, y_val)
0.989010989010989
```

07 여러 개를 분류합니다

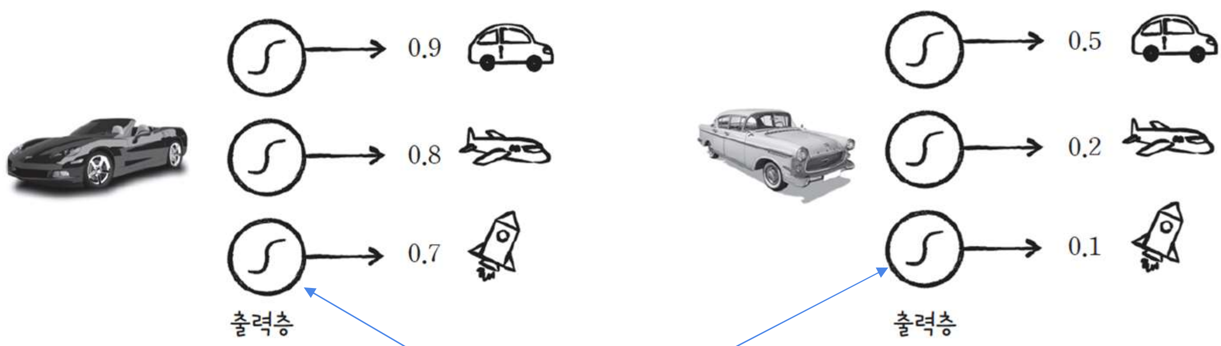
- 다중 분류(multiclass classification)

07-1 여러 개의 이미지를 분류하는 다층 신경망을 만듭니다





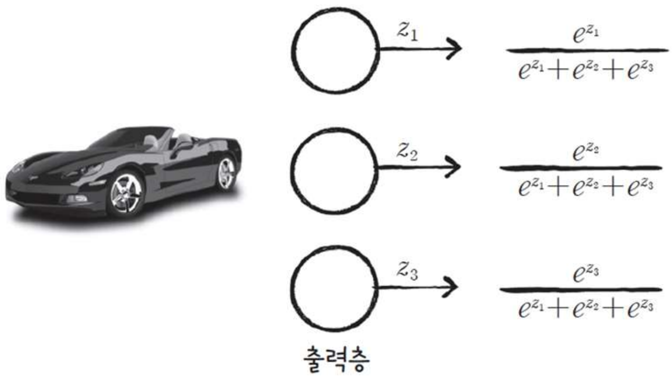
다중 분류의 문제점



시그모이드 함수

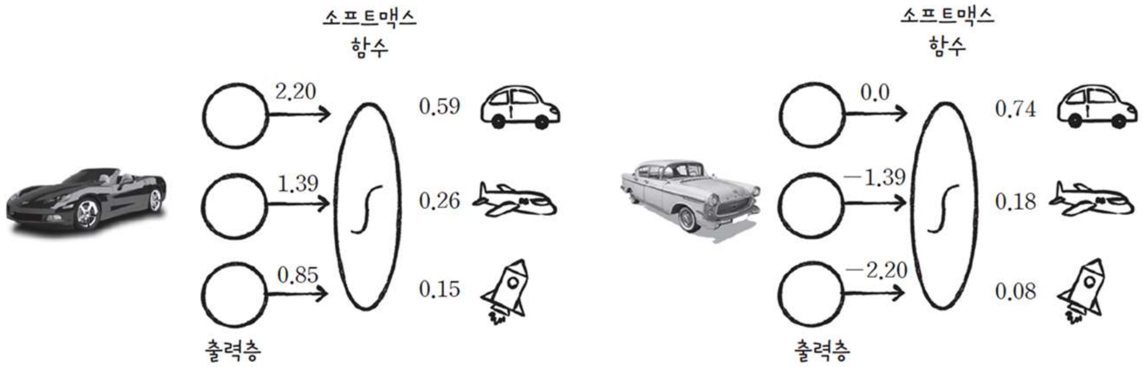
소프트맥스(softmax) 함수

$$\frac{e^{z_i}}{e^{z_1}+e^{z_2}+e^{z_3}}$$



출력 정규화

$$\hat{y}_1=\frac{e^{2.20}}{e^{2.20}+e^{1.39}+e^{0.85}}=0.59 \quad \hat{y}_2=\frac{e^{1.39}}{e^{2.20}+e^{1.39}+e^{0.85}}=0.26 \quad \hat{y}_3=\frac{e^{0.85}}{e^{2.20}+e^{1.39}+e^{0.85}}=0.15$$



$$\hat{y}_1=\frac{e^{0.0}}{e^{0.0}+e^{-1.39}+e^{-2.20}}=0.74 \quad \hat{y}_2=\frac{e^{-1.39}}{e^{0.0}+e^{-1.39}+e^{-2.20}}=0.18 \quad \hat{y}_3=\frac{e^{-2.20}}{e^{0.0}+e^{-1.39}+e^{-2.20}}=0.08$$

다중 분류를 위한 손실 함수

크로스 엔트로피 손실 함수

$$L=-\sum_{c=1}^{c} y_c log(a_c)=-(y_1 log(a_1)+y_2 log(a_2)+\cdots+y_c log(a_c))=-1\times log(a_{y=1})$$

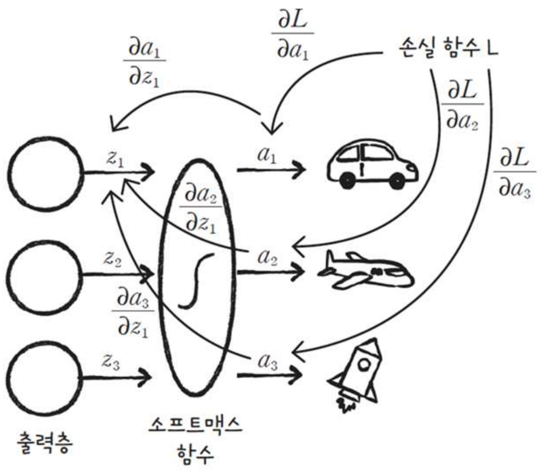
로지스틱 손실 함수

$$L=-(y log(a)+(1-y) log(1-a)) \qquad y=\begin{cases} -log\ a & \text{(양성 클래스인 경우)} \\ -log(1-a) & \text{(음성 클래스인 경우)} \end{cases}$$

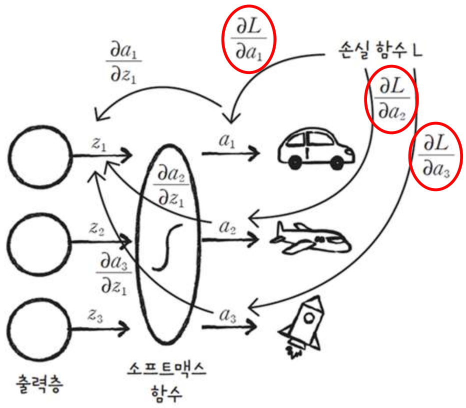
크로스 엔트로피 손실 함수의 미분

z_1에 대한 미분 (다변수 함수의 연쇄 법칙)

$$\frac{\partial L}{\partial z_1}=\frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial z_1}+\frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_1}+\frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_1}$$



∂L/∂a

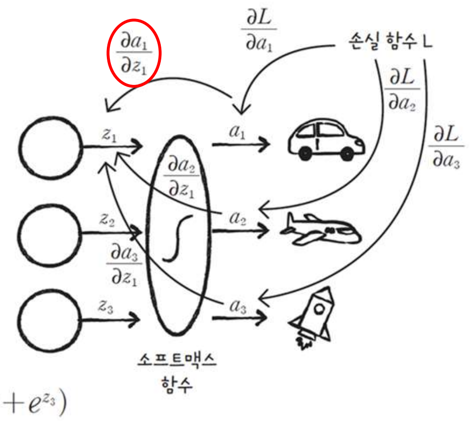


$$L=-(y_1 log\ \ (a_1)+y_2 log(a_2)+y_3 log(a_3))$$

$$\frac{\partial L}{\partial a_1}=-\frac{\partial}{\partial a_1}(y_1 loga_1+y_2 loga_2+y_3 loga_3)=-\frac{y_1}{a_1}$$

$$\frac{\partial L}{\partial a_2}=-\frac{y_2}{a_2} \qquad \frac{\partial L}{\partial a_3}=-\frac{y_3}{a_3}$$

$\partial a_1/\partial z_1$



$$a_1=\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

$$\frac{\partial a_1}{\partial z_1}=\frac{\partial}{\partial z_1}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right)=\frac{(e^{z_1}+e^{z_2}+e^{z_3})\frac{\partial}{\partial z_1}e^{z_1}-e^{z_1}\frac{\partial}{\partial z_1}(e^{z_1}+e^{z_2}+e^{z_3})}{(e^{z_1}+e^{z_2}+e^{z_3})^2}$$

$$=\frac{e^{z_1}(e^{z_1}+e^{z_2}+e^{z_3})-e^{z_1}e^{z_1}}{(e^{z_1}+e^{z_2}+e^{z_3})^2}$$

$$=\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}-\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right)^2=a_1-a_1{}^2=a_1(1-a_1)$$

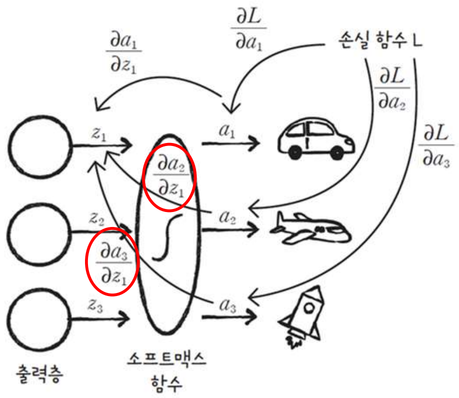
$\partial a_2/\partial z_1$, $\partial a_3/\partial z_1$

$$\frac{\partial a_2}{\partial z_1}=\frac{\partial}{\partial z_1}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right)=\frac{(e^{z_1}+e^{z_2}+e^{z_3})\frac{\partial}{\partial z_1}e^{z_2}-e^{z_2}\frac{\partial}{\partial z_1}(e^{z_1}+e^{z_2}+e^{z_3})}{(e^{z_1}+e^{z_2}+e^{z_3})^2}$$

$$=\frac{O-e^{z_2}e^{z_1}}{(e^{z_1}+e^{z_2}+e^{z_3})^2}=-a_2a_1$$

$$\frac{\partial a_3}{\partial z_1}=\frac{\partial}{\partial z_1}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right)=\frac{(e^{z_1}+e^{z_2}+e^{z_3})\frac{\partial}{\partial z_1}e^{z_3}-e^{z_3}\frac{\partial}{\partial z_1}(e^{z_1}+e^{z_2}+e^{z_3})}{(e^{z_1}+e^{z_2}+e^{z_3})^2}$$

$$=\frac{O-e^{z_3}e^{z_1}}{(e^{z_1}+e^{z_2}+e^{z_3})^2}=-a_3a_1$$



$\partial L/\partial z_1$

$$\frac{\partial L}{\partial z_1}=\frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial z_1}+\frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_1}+\frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_1}$$

$$\frac{\partial L}{\partial z_1}=\left(-\frac{y_1}{a_1}\right)\frac{\partial a_1}{\partial z_1}+\left(-\frac{y_2}{a_2}\right)\frac{\partial a_2}{\partial z_1}+\left(-\frac{y_3}{a_3}\right)\frac{\partial a_3}{\partial z_1}$$

$$=\left(-\frac{y_1}{a_1}\right)a_1(1-a_1)+\left(-\frac{y_2}{a_2}\right)(-a_2a_1)+\left(-\frac{y_3}{a_3}\right)(-a_3a_1)$$

$$=-y_1(1-a_1)+y_2a_1+y_3a_1=-y_1+(y_1+y_2+y_3)a_1=-(y_1-a_1)$$

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{z}}=-(\boldsymbol{y}-\boldsymbol{a})$$

다중 분류 신경망을 구현합니다

소프트맥스 함수 구현

```
def sigmoid(self, z):
    a = 1 / (1 + np.exp(-z))     # 시그모이드 계산
    return a

def softmax(self, z):
    # 소프트맥스 함수
    exp_z = np.exp(z)
    return exp_z / np.sum(exp_z, axis=1).reshape(-1, 1)
```

$$\begin{bmatrix} \vdots \\ 0.9,\ 0.8,\ 0.7 \\ 0.5,\ 0.2,\ 0.1 \\ \vdots \end{bmatrix} \xrightarrow[\text{np.exp(z)}]{} \begin{bmatrix} \vdots \\ e^{0.9},\ e^{0.8},\ e^{0.7} \\ e^{0.5},\ e^{0.2},\ e^{0.1} \\ \vdots \end{bmatrix}$$

$$\begin{bmatrix} & \vdots & \\ 2.45 & 2.22 & 2.01 \\ 1.64 & 1.22 & 1.10 \\ & \vdots & \end{bmatrix} \xrightarrow[\text{np.sum(exp\_z, axis=1)}]{} [\cdots,\ 6.69,\ 3.97,\ \cdots] \xrightarrow[\text{reshape}(-1,\ 1)]{} \begin{bmatrix} \vdots \\ 6.69 \\ 3.97 \\ \vdots \end{bmatrix}$$

init_weights 메서드 수정

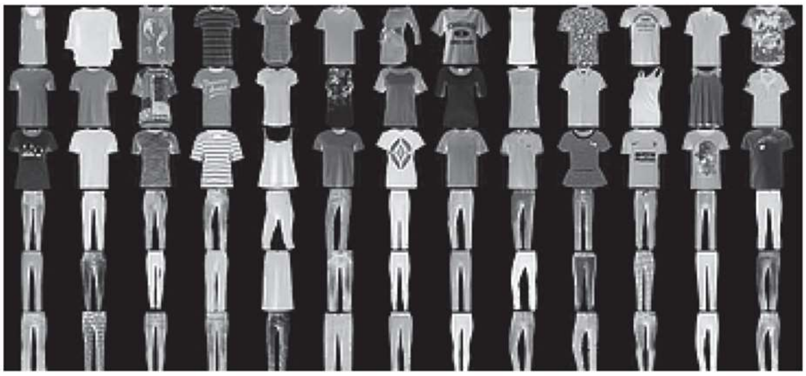
```
def init_weights(self, n_features, n_classes):
    ...
    self.w2 = np.random.normal(0, 1, (self.units, n_classes)) # (은닉층의 크기, 클래스 개수)
    self.b2 = np.zeros(n_classes)
```

update_val_loss() 메서드 수정

```
def update_val_loss(self, x_val, y_val):
    ...
    a = self.softmax(z)                 # 활성화 함수를 적용합니다.
    ...
    # 크로스 엔트로피 손실과 규제 손실을 더하여 리스트에 추가합니다.
    val_loss = np.sum(-y_val*np.log(a))
    ...
```

의류 이미지를 분류합니다

패션 MNIST 데이터셋



텐서플로 2.0 설치

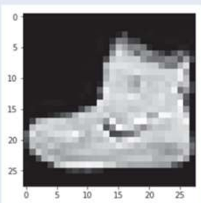
현재 코랩에 설치된 텐서플로는 2.3.0입니다. 책에서처럼 수동으로 최신 버전을 설치할 필요가 없습니다.

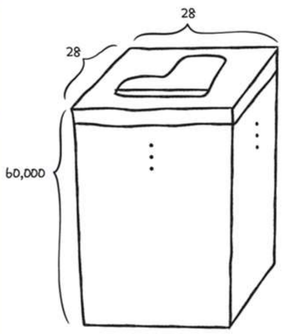
데이터 준비

```
(x_train_all, y_train_all), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data( )
```

```
print(x_train_all.shape, y_train_all.shape)
(60000, 28, 28) (60000,)
```

```
import matplotlib.pyplot as plt
plt.imshow(x_train_all[0], cmap='gray')
plt.show( )
```





타깃 확인

```
print(y_train_all[:10])
[9 0 0 3 0 2 7 2 5 5]
```

```
class_names = ['티셔츠/윗도리', '바지', '스웨터', '드레스', '코트',
               '샌들', '셔츠', '스니커즈', '가방', '앵클부츠']
```

```
print(class_names[y_train_all[0]])
앵클부츠
```

```
np.bincount(y_train_all)
array([6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000])
```

훈련 세트와 검증 세트 준비

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(x_train_all, y_train_all,
stratify=y_train_all, test_size=0.2, random_state=42)
```

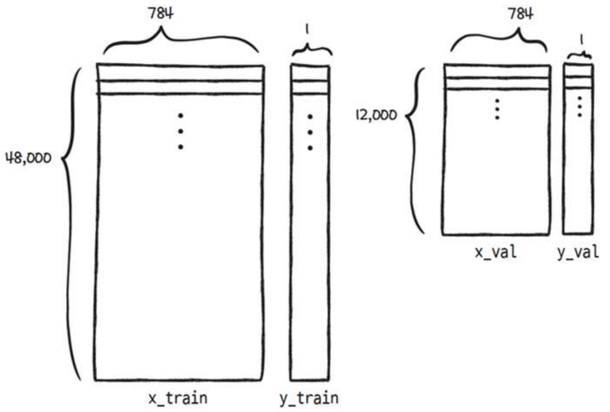
```
np.bincount(y_train)
array([4800, 4800, 4800, 4800, 4800, 4800, 4800, 4800, 4800, 4800])
np.bincount(y_val)
array([1200, 1200, 1200, 1200, 1200, 1200, 1200, 1200, 1200, 1200])
```

```
x_train = x_train / 255
x_val = x_val / 255
```

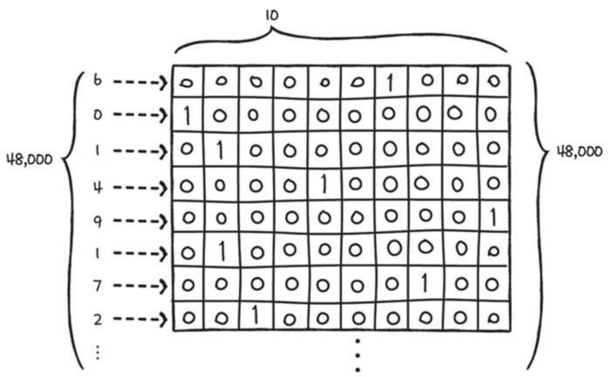
훈련 세트와 검증 세트 차원 변경

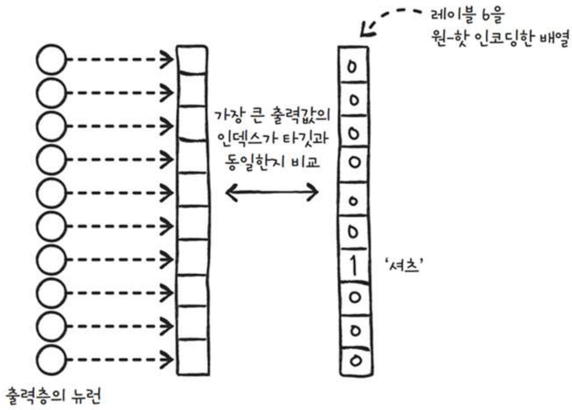
```
x_train = x_train.reshape(-1, 784)
x_val = x_val.reshape(-1, 784)
```

```
print(x_train.shape, x_val.shape)
(48000, 784) (12000, 784)
```



타깃을 원-핫 인코딩으로 바꾸기





to_categorical 함수로 원-핫 인코딩하기

```
tf.keras.utils.to_categorical([0, 1, 3])
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 0., 1.]], dtype=float32)
```

```
y_train_encoded = tf.keras.utils.to_categorical(y_train)
y_val_encoded = tf.keras.utils.to_categorical(y_val)
```

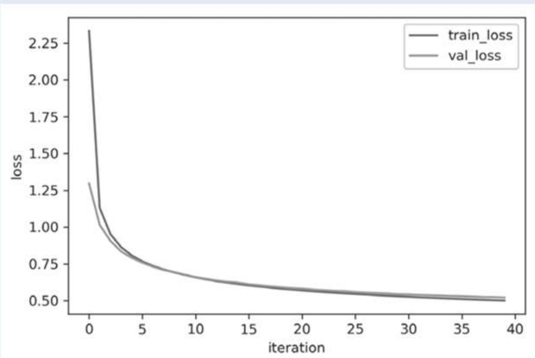
```
print(y_train_encoded.shape, y_val_encoded.shape)
(48000, 10) (12000, 10)
```

```
print(y_train[0], y_train_encoded[0])
6 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

MutliClassNetwork으로 다중 분류 신경망 훈련하기

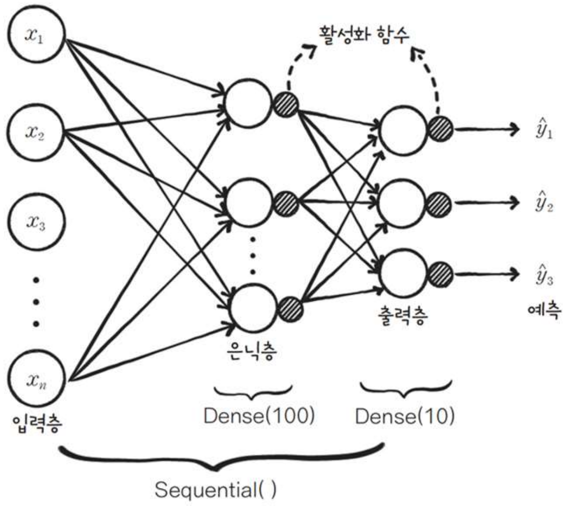
```
fc = MultiClassNetwork(units=100, batch_size=256)
fc.fit(x_train, y_train_encoded,
       x_val=x_val, y_val=y_val_encoded, epochs=40)
........................................
```

```
plt.plot(fc.losses)
plt.plot(fc.val_losses)
plt.ylabel('loss')
plt.xlabel('iteration')
plt.legend(['train_loss', 'val_loss'])
plt.show( )
```




```
fc.score(x_val, y_val_encoded)
0.8150833333333334
```

07-2 텐서플로와 케라스를 사용하여 신경망을 만듭니다



Sequential 모델 훈련하기

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential( )
```

```
model.add(Dense(100, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

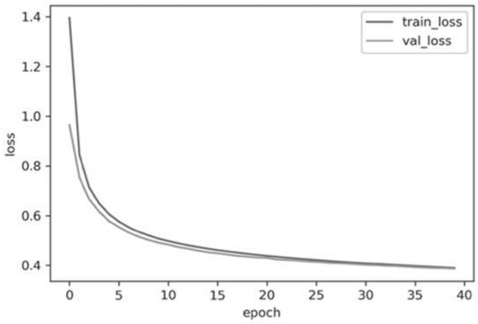
```
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(x_train, y_train_encoded, epochs=40,
                    validation_data=(x_val, y_val_encoded))
Train on 48000 samples, validate on 12000 samples
Epoch 1/20
48000/48000 [==============================] - 2s 45us/sample - loss: 1.3944 - accu-
racy: 0.6396 - val_loss: 0.9643 - val_accuracy: 0.7212
```

손실과 정확도 그리기

```
print(history.history.keys( ))
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```