02_신경망 학습

신경망
학습

- 신경망 학습이란?

> **신경망 학습의 의미**

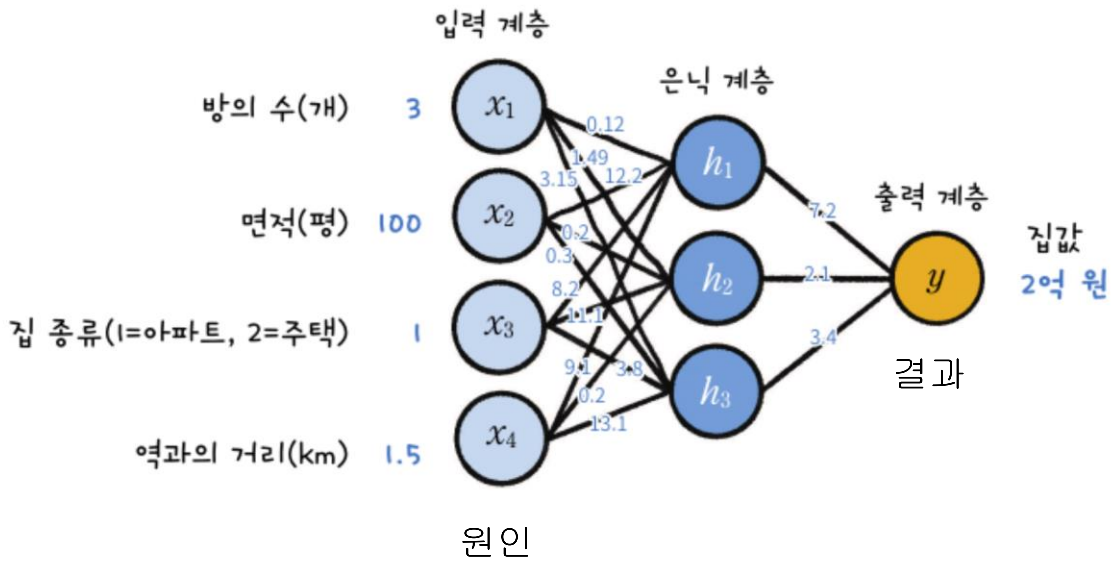
- 신경망을 학습한다는 것은 입력 데이터와 예측해야 할 타깃 데이터가 제공될 뿐, 추론을 위한 규칙은 제공되지 않는다.
- 즉, 신경망에 입력 데이터가 들어왔을 때 어떤 출력 데이터를 만들어야 할지를 정하는 규칙은 함수적 매핑 관계로 표현된다.
- 가중 합산과 활성 함수가 연결되어 뉴런을 구성하고, 뉴런이 모여 계층을 구성하며, 계층이 쌓여 신경망의 계층 구조가 정의된다.

• 신경망 학습이란?

> 집값 예측 문제일 때

- 입력 데이터(방의 수, 면적, 집 종류, 역과의 거리 등)이 입력되고 타깃 데이터(집값)이 있다.
- 다음과 같은 규칙이 만들어지면 집값을 예측할 수 있는 추론 능력이 생겼다고 볼 수 있다.

규칙을 딥러닝이 스스로 만듦



• 신경망 학습 : 회귀

> 데이터 준비

```
from sklearn.datasets import load_diabetes

# 당뇨병 환자 데이터
diabetes = load_diabetes()
x = diabetes.data[:, 2]
y = diabetes.target
print(diabetes.data.shape, diabetes.target.shape)
```

Out : (442, 10) (442,)

• 신경망 학습 : 회귀

> 가중치 업데이트

```
# 가중치 초기화
w = 1.0
b = 1.0

y_hat = x[0] * w + b
print('예측 데이터 :', y_hat)
print('실제 데이터 :', y[0])
```

Out : 예측 데이터 : 1.0616962065186886

실제 데이터 : 151.0

• 신경망 학습 : 회귀

> 가중치 업데이트

```
# 가중치 값을 조절해 예측값 바꾸기
w_inc = w + 0.1
y_hat_inc = x[0] * w_inc + b
print('변경된 예측값 :', y_hat_inc)

# 예측값 증가 정도 확인
w_rate = (y_hat_inc - y_hat) / (w_inc - w)
print('증가 정도 :', w_rate)
```

Out : 변경된 예측값 : 1.0678658271705574

증가 정도 : 0.061696206518688734

• 신경망 학습 : 회귀

> 가중치 업데이트

```
# 변화율로 가중치 업데이트
w_new = w + w_rate
print(w_new)
```

Out : 1.0616962065186888

> 오차와 변화율을 곱하여 가중치 업데이트

```
# 변화율로 가중치 업데이트
err = y[0] - y_hat
w_new = w + w_rate * err
print(w_new)
```

Out : 10.250624555904514

• 신경망 학습 : 회귀

> 가중치 업데이트

```
# 변화율로 절편 업데이트
b_inc = b + 0.1
y_hat_inc = x[0] * w + b_inc

b_rate = (y_hat_inc - y_hat) / (b_inc - b)
print('절편 변화율 :', b_rate)

err = y[0] - y_hat
b_new = b + b_rate * err
print(b_new)
```

Out : 1

150.9383037934813

• 신경망 학습 : 회귀

> 가중치 업데이트

```
# 반복하여 w 구하기
for x_i, y_i in zip(x, y):
    y_hat = x_i * w + b
    err = y_i - y_hat
    w_rate = x_i
    b_rate = 1
    w = w + w_rate * err
    b = b + b_rate * err
print(w, b)
```

Out : 587.8654539985689 99.40935564531424

• 신경망 학습 : 회귀

> 가중치 업데이트

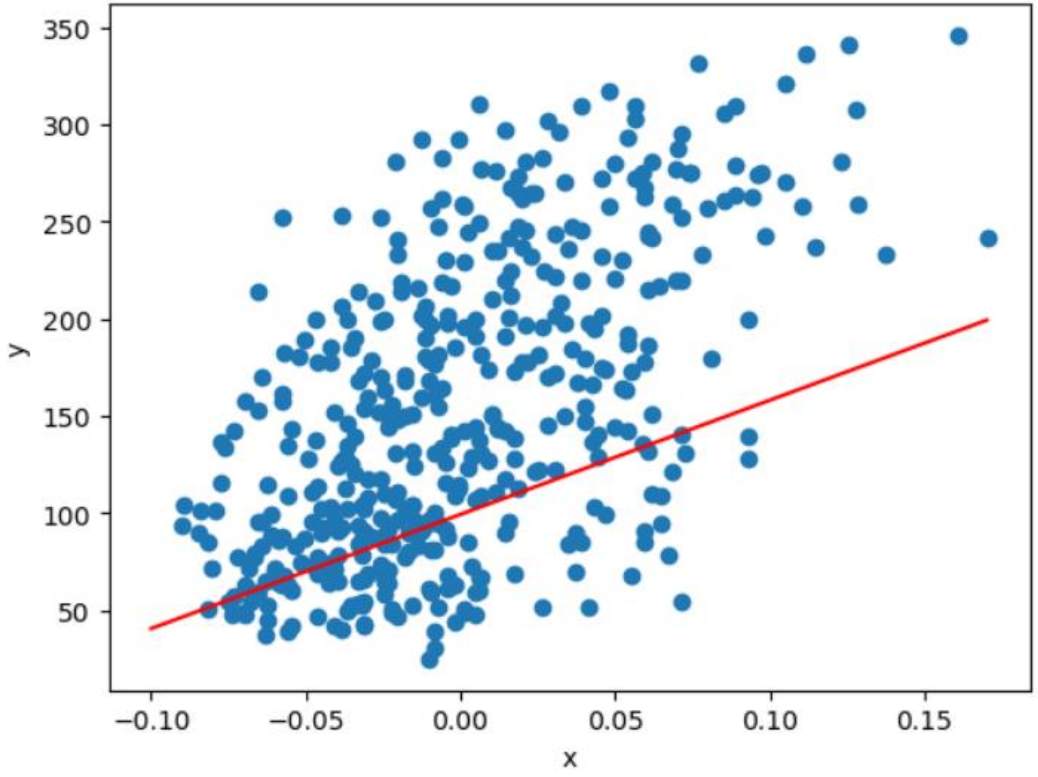
```
import matplotlib.pyplot as plt
# 예측선 그리기
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.17, 0.17 * w + b)
plt.plot([pt1[0], pt2[0]],[pt1[1],pt2[1]], color='red')
plt.xlabel('x'); plt.ylabel('y')
plt.show()
```

신경망 학습

- 신경망 학습 : 회귀

> 가중치 업데이트

Out :



신경망 학습

• 신경망 학습 : 회귀

> 가중치 업데이트

```
# 에포크를 반복하기  에포크: 모든 데이터를 받아 학습하는 횟수
w = 1.0
b = 1.0
for i in range(100):
    for x_i, y_i in zip(x, y):
        y_hat = x_i * w + b
        err = y_i - y_hat
        w_rate = x_i
        b_rate = 1
        w = w + w_rate * err    x(y-y^)
        b = b + b_rate * err    1(y-y^)
```

신경망 학습

- 신경망 학습 : 회귀

> 가중치 업데이트

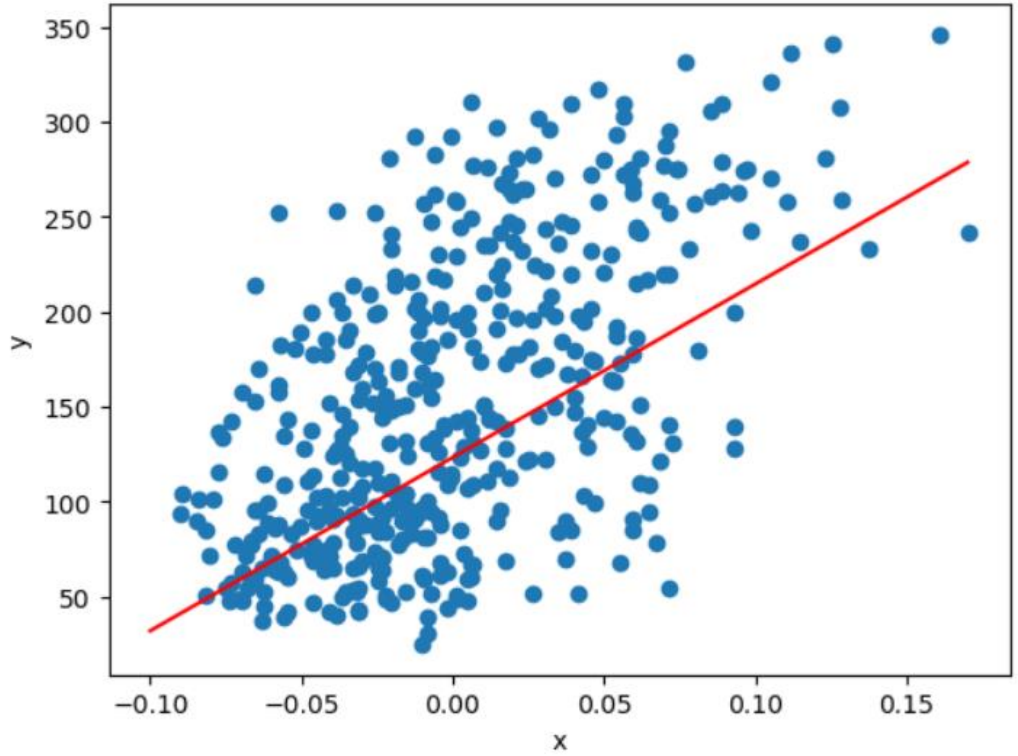
```
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.17, 0.17 * w + b)
plt.plot([pt1[0], pt2[0]],[pt1[1],pt2[1]], color='red')
plt.xlabel('x'); plt.ylabel('y')
plt.show()
```

신경망 학습

• 신경망 학습 : 회귀

> 가중치 업데이트

Out :



신경망
학습

• 신경망 학습 : 회귀

> 손실함수를 사용한 가중치 업데이트 : 손실함수 loss

- 제곱 오차 $SE=(y-\hat{y})^2$ 최솟값 0 MSE : 회귀
- 가중치에 대한 편미분 $\frac{\partial SE}{\partial w}=-2(y-\hat{y})x$ w에 대한 미분
- 가중치 업데이트 $w=w-\frac{\partial SE}{\partial w}=w+(y-\hat{y})x$ 에러
- 절편에 대한 제곱 오차 업데이트 $b=b-\frac{\partial SE}{\partial b}=b+(y-\hat{y})$

편미분

• 신경망 학습 : 회귀

> 선형 회귀 뉴런 생성

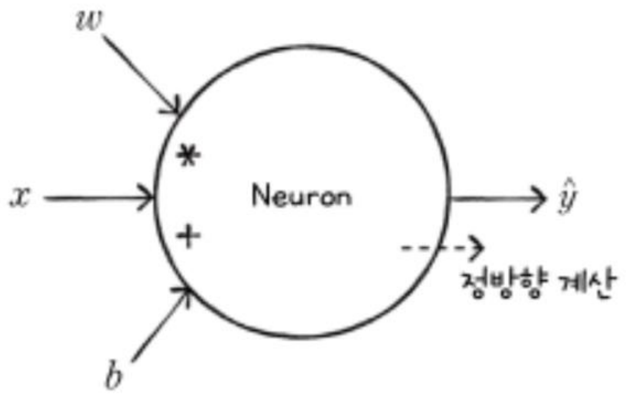
```
# 뉴런 구조를 클래스로 생성
class Neuron:
    # 기본 가중치 생성
    def __init__(self):
        self.w = 1.0
        self.b = 1.0
```

신경망 학습

• 신경망 학습 : 회귀

> 선형 회귀 뉴런 생성

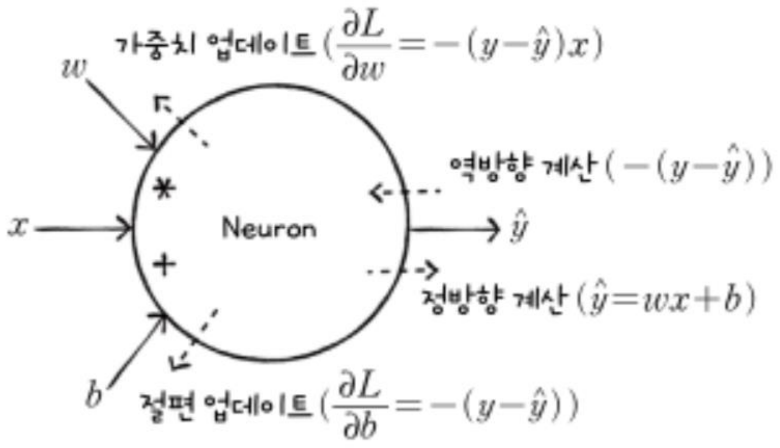
```
# 정방향 계산 함수
def forpass(self, x):
    y_hat = x * self.w + self.b
    return y_hat
```



• 신경망 학습 : 회귀

> 선형 회귀 뉴런 생성

```
# 역방향 오차 가중치 업데이트
def backprop(self, x, err):
    w_grad = x * err
    b_grad = 1 * err
    return w_grad, b_grad
```



• 신경망 학습 : 회귀

> 선형 회귀 뉴런 생성

```
# 훈련을 위한 fit() 메서드 구현
def fit(self, x, y, epochs = 100):
    for i in range(epochs):
        for x_i, y_i in zip(x, y):
            y_hat = self.forpass(x_i)
            err = -(y_i - y_hat)
            w_grad, b_grad = self.backprop(x_i, err)
            self.w -= w_grad
            self.b -= b_grad
```

• 신경망 학습 : 회귀

> 선형 회귀 뉴런 생성

```
# 클래스 호출
neuron = Neuron()

# 학습(기본 epoch 100회)
neuron.fit(x, y)
print('학습된 w :', neuron.w)
print('학습된 b :', neuron.b)
```

Out : 학습된 w : 913.5973364345905

학습된 b : 123.39414383177204

신경망
학습

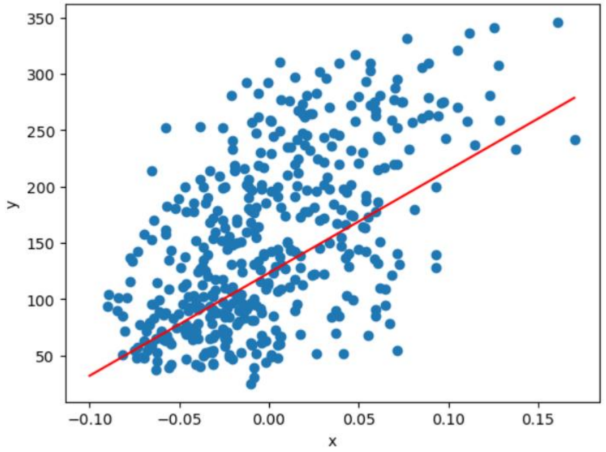
• 신경망 학습 : 회귀

회귀의 활성화함수는 y=x 임/ 결론 없음

> 학습된 결과 확인

```
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * neuron.w + neuron.b)
pt2 = (0.17, 0.17 * neuron.w + neuron.b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]], color='red')
plt.xlabel('x'); plt.ylabel('y')
plt.show()
```

Out :



신경망 학습

• 신경망 학습 : 분류

출력: 1개 b도 한개
입력 n개 w도 n개

> 로지스틱 손실 함수

엔트로이 계산식 비슷

$$L=-(ylog(a)+(1-y)log(1-a))$$

> 손실 함수 미분값 차이

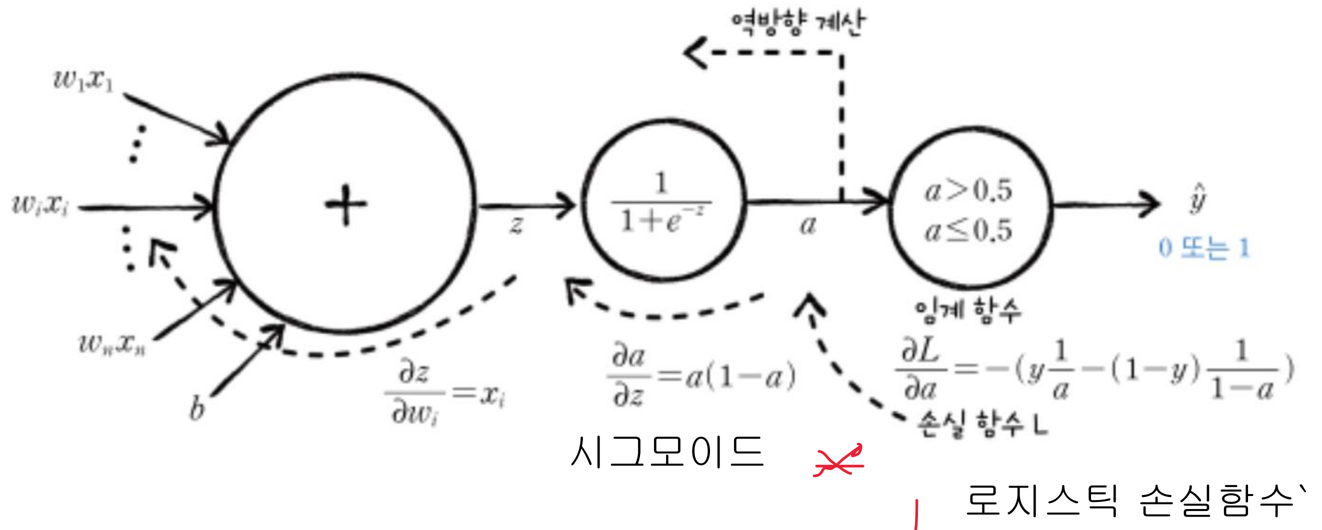
	제곱 오차의 미분	로지스틱 손실 함수의 미분
가중치에 대한 미분	$\frac{\partial SE}{\partial w}=-(y-\hat{y})x$	$\frac{\partial}{\partial w_i}L=-(y-a)x_i$
절편에 대한 미분	$\frac{\partial SE}{\partial b}=-(y-\hat{y})1$	$\frac{\partial}{\partial b}L=-(y-a)1$

: 분류든 회귀든 동일한 공식으로 계산은 같음

신경망 학습

• 신경망 학습 : 분류

> 로지스틱 손실 함수 역전파



> 가중치 업데이트

$$w_i=w_i-\frac{\partial L}{\partial w_i}=w_i+(y-a)x_i \qquad b=b-\frac{\partial L}{\partial b}=b+(y-a)1$$

• 신경망 학습 : 분류

> 데이터 준비

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
x = cancer.data
y = cancer.target
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
print(X_train.shape, X_test.shape)
```

Out : (455, 30) (114, 30)

• 신경망 학습 : 분류

> 데이터 준비

```
import numpy as np
# 가중치 초기화
w = np.ones(X_train.shape[1])
b = 1.0

# 모든 변수에 대한 가중치 적용 값
z = np.sum(X_train[0] * w) + b
print('가중치 적용 값 :', z)
```

Out : 가중치 적용 값 : 920.0319740000001

• 신경망 학습 : 분류

> 시그모이드 함수   0.5를 기준으로 0,1로 나눔

```
# 안전한 계산을 위해
z = np.clip(z, -100, None)

# 시그모이드 함수
a = 1 / (1 + np.exp(-z))
print('시그모이드 함수 통과 :', a)
```

범위를 정하고 그 안 값만 출력하는 함수:
np.clip(입력값, 최소값, 최대값)

Out : 시그모이드 함수 통과 : 1.0

신경망 학습

• 신경망 학습 : 분류

> 신경망 구조 생성하기

```
class LogisticNeuron:
    # 기본 가중치 생성
    def __init__(self):
        self.w = None
        self.b = None
    # 정방향 계산 함수
    def forpass(self, x):
        z = np.sum(x * self.w) + self.b
        return z
```

• 신경망 학습 : 분류

> 신경망 구조 생성하기

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

# 시그모이드 함수
def activation(self, z):
    z = np.clip(z, -100, None)
    a = 1 / (1 + np.exp(-z))
    return a
```

• 신경망 학습 : 분류

> 신경망 구조 생성하기

```
# 훈련을 위한 fit() 메서드 생성
def fit(self, x, y, epochs = 100):
    self.w = np.ones(x.shape[1])
    self.b = 0
    for i in range(epochs):
        for x_i, y_i in zip(x, y):
            z = self.forpass(x_i)
            a = self.activation(z)
            err = -(y_i - a)
            w_grad, b_grad = self.backprop(x_i, err)
            self.w -= w_grad
            self.b -= b_grad
```

• 신경망 학습 : 분류

> 신경망 구조 생성하기

```
# 예측 함수 생성
def predict(self, x):
    z = [self.forpass(x_i) for x_i in x]
    a = self.activation(np.array(z))
    return a > 0.5
```

신경망
학습

• 신경망 학습 : 분류

> 신경망 구조 생성하기

```
# 모델 훈련
neuron = LogisticNeuron()
neuron.fit(X_train, y_train)
print(neuron.w, neuron.b)
```

Out :

```
[ 4.44021200e+03 -1.92885000e+03  2.31902600e+04  4.82700000e+03
  1.95098500e+01 -1.11436330e+02 -2.07648288e+02 -7.92234290e+01
  2.56121000e+01  1.99302000e+01 -2.43320000e+01 -2.19092700e+02
 -7.60099000e+02 -1.49900060e+04  7.47466000e-01 -3.49603630e+01
 -5.00837511e+01 -8.96486300e+00 -4.73503200e+00 -1.33099400e+00
  4.74971700e+03 -4.81984000e+03  2.12323400e+04 -9.47580000e+03
  6.09926000e+00 -4.27425560e+02 -5.99597289e+02 -1.48915810e+02
 -4.41484000e+01 -7.82273000e+00] 569.0
```

• 신경망 학습 : 분류

> 신경망 구조 생성하기

```
# 정확도 예측
pred = neuron.predict(X_test)
print(pred[:5])
```

Out : [False True True False True]

```
np.mean(pred == y_test)
```

Out : 0.7982456140350878

신경망 학습

• 단일 신경망 구현

> 손실 함수의 결과 값 저장 기능

```
class SingleLayer:
    def __init__(self):
        self.w = None
        self.b = None
        # 손실 함수 저장하기 위한 리스트
        self.losses = []

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

• 단일 신경망 구현

> 손실 함수의 결과 값 저장 기능

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

def activation(self, z):
    z = np.clip(z, -100, None)
    a = 1 / (1 + np.exp(-z))
    return a
```

• 단일 신경망 구현

> 손실 함수의 결과 값 저장 기능

```
def fit(self, x, y, epochs = 100):
    self.w = np.ones(x.shape[1])
    self.b = 0
    for i in range(epochs):
        # 손실 초기화
        loss = 0            :x,y가 각각 섞이지않도록 인덱스를 만들어 행렬로 만들어 사용
        # x의 index 랜덤하게 반환
        indexes = np.random.permutation(np.arange(len(x)))  # 값 뒤섞는 함수
        for i in indexes:
            z = self.forpass(x[i])
            a = self.activation(z)
            err = -(y[i] - a)
```

신경망 학습

• 단일 신경망 구현

> 손실 함수의 결과 값 저장 기능

```
            w_grad, b_grad = self.backprop(x[i], err)
            self.w -= w_grad
            self.b -= b_grad
            # 안전한 로그 계산을 위한 범위 축소
            a = np.clip(a, 1e-10, 1-1e-10)
            # 손실 계산 로지스틱함수
            loss += -(y[i] * np.log(a) + (1 - y[i]) * np.log(1 - a))
        # 에포크마다 평균 손실을 저장
        self.losses.append(loss / len(y))                         결과
```

손실계산:
회귀때 사용한 로지스틱 손실함수와 같음(얼마나 오차가 생겼는지 확인,평가하는 함수)

신경망 학습

• 단일 신경망 구현

> 손실 함수의 결과 값 저장 기능

```
def predict(self, x):
    z = [self.forpass(x_i) for x_i in x]
    return np.array(z) > 0

# 정확도 계산 함수 생성
def score(self, x, y):
    return np.mean(self.predict(x) == y)
```

• 단일 신경망 구현

> 단일 신경망 훈련하기

```
layer = SingleLayer()
layer.fit(X_train, y_train)
print(layer.score(X_test, y_test))

# 손실 함수 누적값 확인하기
plt.plot(layer.losses)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

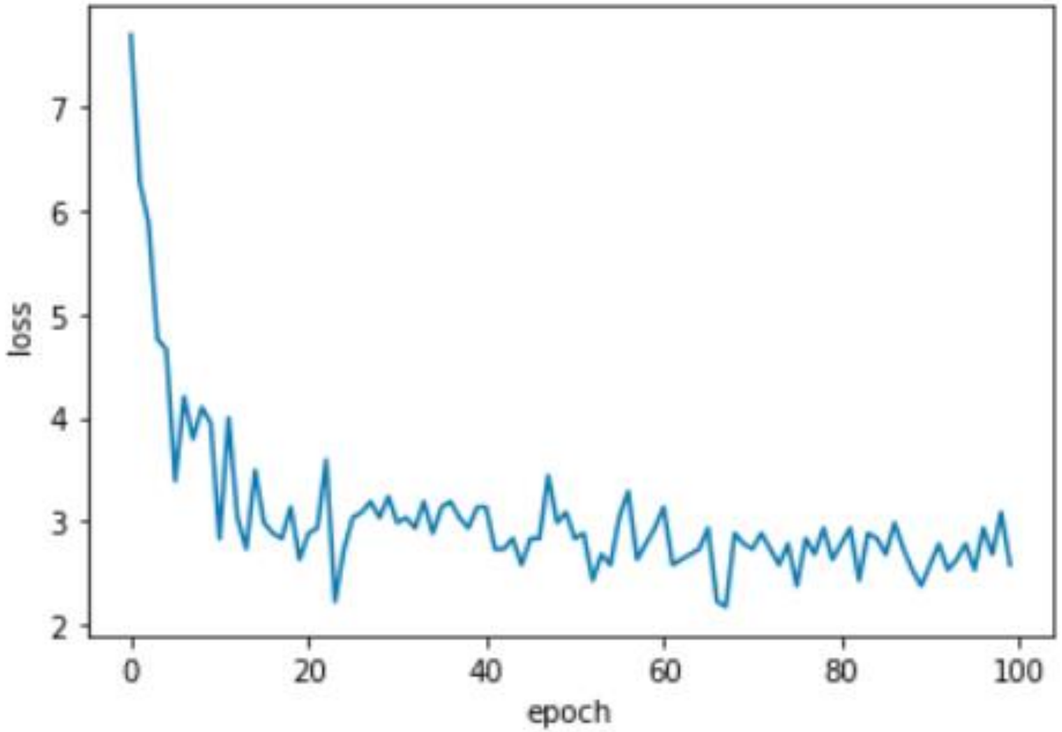
Out : 0.9122807017543859

신경망 학습

• 단일 신경망 구현

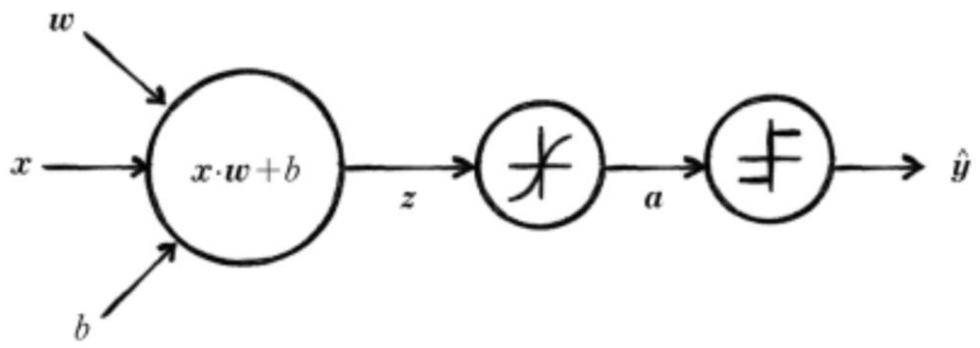
> 단일 신경망 훈련하기

Out :



• 신경망 학습 : 벡터 연산, 행렬 연산

> 딥러닝의 행렬 연산



- 위와 같은 식을 다음과 같이 표현할 수 있다.

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

• 신경망 학습 : 벡터 연산, 행렬 연산

> 딥러닝의 행렬 연산

- 입력 데이터의 개수가 증가하면 행렬의 크기도 편하게 된다.

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

가중합

- np.dot(X, W)로 나타낼 수 있다.
- 정방향 계산을 행렬곱으로 나타내면 다음과 같다.

절편

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

• 신경망 학습 : 벡터 연산, 행렬 연산

> 그레이디언트 계산하기

- 가중치를 업데이트하기 위한 기울기는 입력 데이터(X)와 오차(err)의 곱으로 나타낸다.
- 행렬 곱으로 나타내면 다음과 같다.

$$\boldsymbol{X}^T\boldsymbol{E} = 30\left\{ \overbrace{\begin{bmatrix} x_1^{(1)} & & x_1^{(364)} \\ x_2^{(1)} & \cdots & x_2^{(364)} \\ \vdots & & \vdots \\ x_{30}^{(364)} & & x_{30}^{(364)} \end{bmatrix}}^{364} \begin{bmatrix} e^{(1)} \\ e^{(2)} \\ \vdots \\ e^{(364)} \end{bmatrix} \right\} 364 = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{30} \end{bmatrix}$$

- 입력 데이터를 전치하여 오차와 곱을 해주면 모든 특성과 오차의 곱의 합을 구할 수 있다.
  평균값을 계산하면 기울기를 구할 수 있다.

원인

신경망 학습

• 신경망 학습 : 벡터 연산, 행렬 연산

> 딥러닝의 행렬 연산

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X = cancer.data
y = cancer.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

신경망 학습

• 신경망 학습 : 벡터 연산, 행렬 연산

> 딥러닝의 행렬 연산

```
# class SingleLayer 에서 변환
def forpass(self, x):
    z = np.dot(x, self.w) + self.b
    return z

def backprop(self, x, err):
    m = len(x)
    w_grad = np.dot(x.T, err) / m # 평균으로 계산
    b_grad = np.sum(err) / m
    return w_grad, b_grad
```

• 신경망 학습 : 벡터 연산, 행렬 연산

> 딥러닝의 행렬 연산

```
def fit(self, x, y, epochs = 100):
    y = y.reshape(-1,1) # 열 벡터로 변환
    m = len(x)
    self.w = np.ones((x.shape[1], 1)) # 가중치 초기화
    self.b = 0 # 절편 초기화
    for i in range(epochs):
        z = self.forpass(x)
        a = self.activation(z)
        err = -(y - a)
        w_grad, b_grad = self.backprop(x, err)
        self.w -= w_grad
        self.b -= b_grad
        a = np.clip(a, 1e-10, 1-1e-10)
        loss = np.sum(-(y*np.log(a) + (1-y)*np.log(1-a)))
        self.losses.append(loss)
```

- 신경망 학습 : 벡터 연산, 행렬 연산

> 전체 클래스

```
class SingleLayer:
    def __init__(self):
        self.w = None
        self.b = None
        self.losses = []

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

신경망 학습

- 신경망 학습 : 벡터 연산, 행렬 연산

> 전체 클래스

```
def backprop(self, x, err):
    m = len(x)
    w_grad = np.dot(x.T, err) / m
    b_grad = np.sum(err) / m
    return w_grad, b_grad

def activation(self, z):
    z = np.clip(z, -100, None)
    a = 1 / (1 + np.exp(-z))
    return a
```

신경망 학습

• 신경망 학습 : 벡터 연산, 행렬 연산

> 전체 클래스

```
    def predict(self, x):
        z = self.forpass(x)
        return z > 0

    def score(self, x, y):
        return np.mean(self.predict(x) == y.reshape(-1, 1))
```

• 신경망 학습 : 벡터 연산, 행렬 연산

> 전체 클래스

```
def fit(self, x, y, epochs = 100, random_state = None):
    y = y.reshape(-1,1) # 열 벡터로 변환
    m = len(x)
    self.w = np.ones((x.shape[1], 1)) # 가중치 초기화
    self.b = 0 # 절편 초기화
    for i in range(epochs):
        z = self.forpass(x)
        a = self.activation(z)
        err = -(y - a)
        w_grad, b_grad = self.backprop(x, err)
        self.w -= w_grad
        self.b -= b_grad
        a = np.clip(a, 1e-10, 1-1e-10)
        loss = np.sum(-(y*np.log(a) + (1-y)*np.log(1-a)))
        self.losses.append(loss)
```

• 신경망 학습 : 벡터 연산, 행렬 연산

> 모델 학습/예측/평가

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# 스케일링
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

신경망 학습

- 신경망 학습 : 벡터 연산, 행렬 연산

> 모델 학습/예측/평가

```
single_layer = SingleLayer()
single_layer.fit(X_train_scaled, y_train, epochs=1000)
single_layer.score(X_test_scaled, y_test)
```

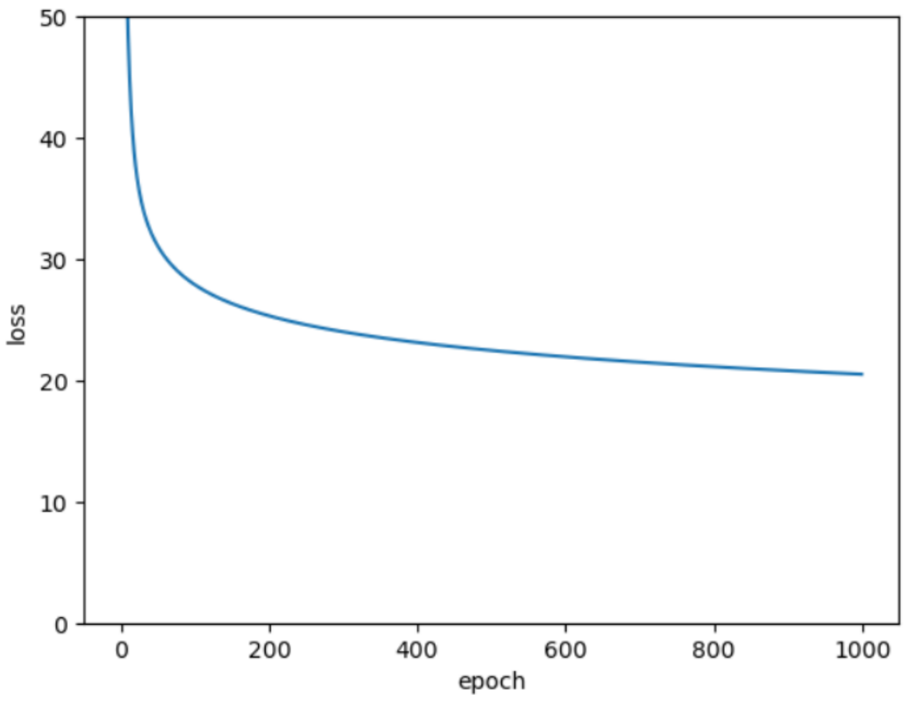
Out : 0.9649122807017544

```
plt.plot(single_layer.losses)
plt.ylim(0,50)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
print(single_layer.losses[-1])
```

• 신경망 학습 : 벡터 연산, 행렬 연산

> 모델 학습/예측/평가

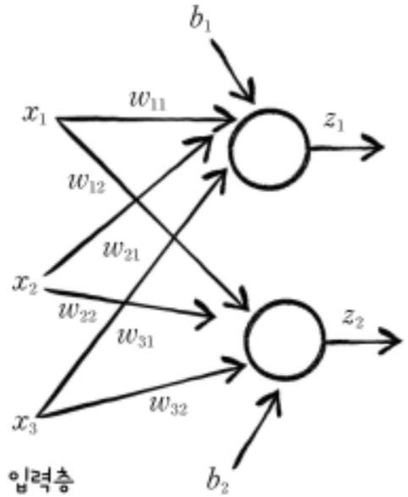
Out : 20.50999850682708



• 신경망 학습 : 다층 신경망

> **다층 퍼셉트론**

- 은닉 계층을 가지고 있는 신경망 구조
- 가중치 W가 2차원 행렬로 존재 (입력층 x 출력층)



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

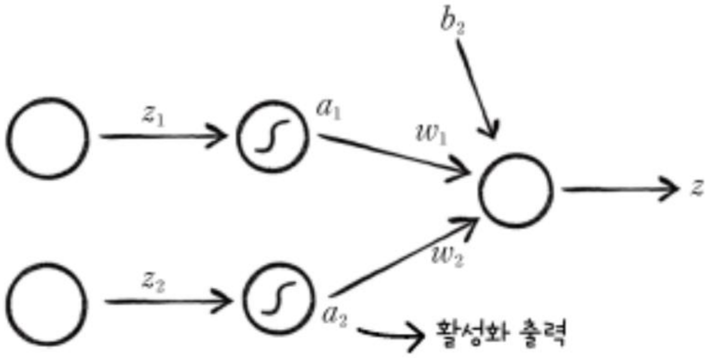
$$[x_1\ x_2\ x_3]\begin{bmatrix} w_{11} & w_{21} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}+[b_1\ b_2]=[z_1\ z_2]$$

신경망 학습

• 신경망 학습 : 다층 신경망

> 다층 퍼셉트론

- 은닉층 > 출력층의 가중치W가 또 존재



$$a_1w_1 + a_2w_2 + b_2 = z \text{ (선형 방정식)}$$

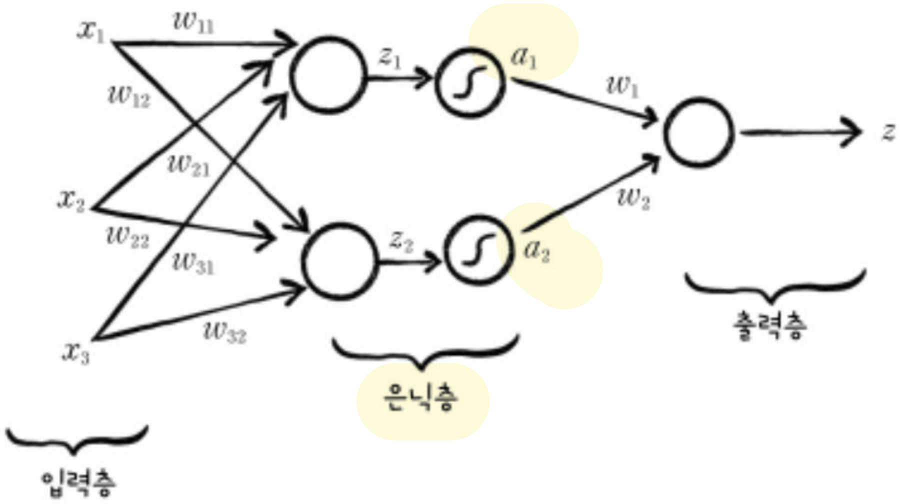
$$[a_1\ a_2]\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b_2 = z \text{ (행렬 곱셈)}$$

z는 y^

• 신경망 학습 : 다층 신경망

> **다층 퍼셉트론**

- 입력층 > 은닉층, 은닉층 > 출력층의 퍼셉트론을 하나로 합치면 다음과 같은 다층 퍼셉트론이 생성된다.
- 하나의 계층에는 같은 활성 함수를 사용해야한다.

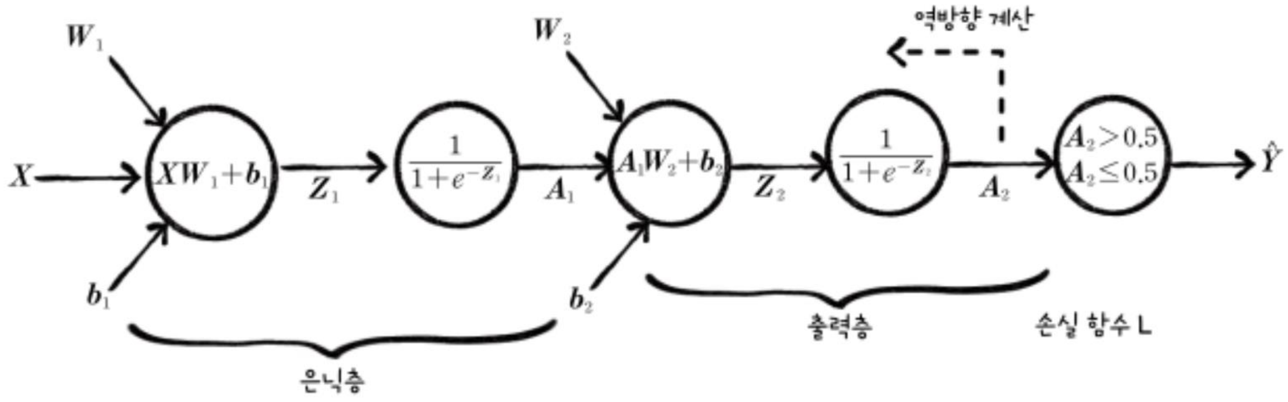


신경망
학습

• 신경망 학습 : 다층 신경망

> **경사 하강법 적용 (오차 역전파)**

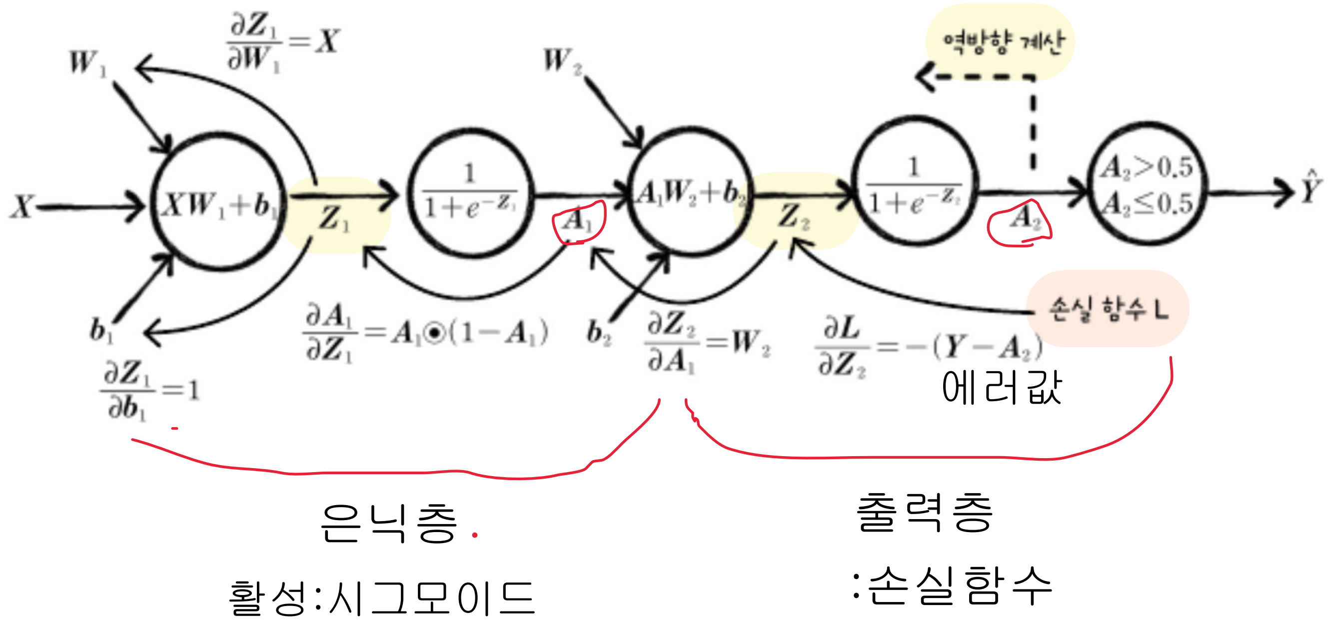
- 은닉층에서 입력받은 X는 가중 합산, 활성 함수를 통해 A라는 값을 출력에 전달하고 출력층에서 A1은 가중 합산, 활성 함수를 통해 A2로 출력된다.



• 신경망 학습 : 다층 신경망

> 경사 하강법 적용 (오차 역전파)

- 손실함수 L로부터 은닉층 > 출력층의 가중치 W2부터 업데이트를 하고 다음으로 입력층 > 은닉층의 가중치 W1을 업데이트 한다.



신경망 학습

• 신경망 학습 : 다층 신경망

> 경사 하강법 적용 (오차 역전파)

- W2의 업데이트

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

$$\frac{\partial L}{\partial Z_2} = -(Y - A_2) = \underbrace{\begin{bmatrix} 0.7 \\ 0.3 \\ \vdots \\ 0.6 \end{bmatrix}}_{m\text{개}}$$

$$\frac{\partial Z_2}{\partial W_2} = A_1 = \underbrace{\begin{bmatrix} -1.37 & 0.96 \\ & \vdots \\ 2.10 & -0.17 \end{bmatrix}}_{m\text{개}}$$

$$A_1^T(-(Y - A_2)) = \underbrace{\begin{bmatrix} -1.37 \cdots 2.10 \\ 0.96 \quad -0.17 \end{bmatrix}}_{m\text{개}} \underbrace{\begin{bmatrix} 0.7 \\ 0.3 \\ \vdots \\ 0.6 \end{bmatrix}}_{\text{타깃과 예측의 차이}} = \overbrace{\begin{bmatrix} -0.12 \\ 0.36 \end{bmatrix}}^{\text{그레이디언트의 총 합}}$$

신경망 학습

• 신경망 학습 : 다층 신경망

> **경사 하강법 적용 (오차 역전파)**

- b2의 업데이트

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{b}_2}=\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{Z}_2}\frac{\partial \boldsymbol{Z}_2}{\partial \boldsymbol{b}_2}=\mathbf{1}^T(-(\boldsymbol{Y}-\boldsymbol{A}_2))=\underbrace{[1 \cdots 1]}_{m\text{개}}\begin{bmatrix}0.7\\0.3\\\vdots\\0.6\end{bmatrix}=0.18$$

타깃과 예측의 차이

• 신경망 학습 : 다층 신경망

> 경사 하강법 적용 (오차 역전파)

- W1의 업데이트

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{W}_1} = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{Z}_2} \frac{\partial \boldsymbol{Z}_2}{\partial \boldsymbol{A}_1} \frac{\partial \boldsymbol{A}_1}{\partial \boldsymbol{Z}_1} \frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{W}_1}$$

$$\frac{\partial \boldsymbol{Z}_2}{\partial \boldsymbol{A}_1} = \boldsymbol{W}_2 = \begin{bmatrix} 0.02 \\ 0.16 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{W}_1} = \boldsymbol{X} = \left.\begin{bmatrix} 3 & 6 & 2 \\ 1 & 10 & 7 \\ & \vdots & \\ 4 & 8 & 1 \end{bmatrix}\right\} m\text{개}$$

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{Z}_2} = -(\boldsymbol{Y} - \boldsymbol{A}_2) = \left.\begin{bmatrix} 0.7 \\ 0.3 \\ \vdots \\ 0.6 \end{bmatrix}\right\} m\text{개}$$

$$\frac{\partial \boldsymbol{A}_1}{\partial \boldsymbol{Z}_1} = \boldsymbol{A}_1 \odot (1 - \boldsymbol{A}_1) = \left.\begin{bmatrix} 2.37 & 6.10 \\ \vdots & \\ 1.81 & 4.82 \end{bmatrix}\right\} m\text{개}$$

신경망 학습

• 신경망 학습 : 다층 신경망

> **경사 하강법 적용 (오차 역전파)**

- W1의 업데이트

에러 X 전치

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

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

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

신경망 학습

• 신경망 학습 : 다층 신경망

> 경사 하강법 적용 (오차 역전파)

- b1의 업데이트

$$\frac{\partial L}{\partial \boldsymbol{b}_1} = \frac{\partial L}{\partial \boldsymbol{Z}_2}\frac{\partial \boldsymbol{Z}_2}{\partial \boldsymbol{A}_1}\frac{\partial \boldsymbol{A}_1}{\partial \boldsymbol{Z}_1}\frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{b}_1} = \mathbf{1}^T(-(\boldsymbol{Y}-\boldsymbol{A}_2)\boldsymbol{W}_2^T \odot \boldsymbol{A}_1 \odot (1-\boldsymbol{A}_1))$$

$$= [1 \quad 1 \cdots 1]\underbrace{\begin{bmatrix} -0.045 & -3.48 \\ & \vdots \\ -0.018 & -1.768 \end{bmatrix}}_{m\text{개}}$$

$$= [0.121 \quad -0.034]$$

• 신경망 학습 : 다층 신경망

> 다층 신경망 구현

```
class DualLayer(SingleLayer):
    def __init__(self, units = 10):
        self.units = units # 은닉층의 뉴런 개수
        self.w1 = None # 입력 > 은닉 가중치
        self.b1 = None # 입력 > 은닉 절편
        self.w2 = None # 은닉 > 출력 가중치
        self.b2 = None # 은닉 > 출력 절편
        self.a1 = None # 은닉층의 활성화 출력
        self.losses = []

    def forpass(self, x):
        z1 = np.dot(x, self.w1) + self.b1
        self.a1 = self.activation(z1)
        z2 = np.dot(self.a1, self.w2) + self.b2
        return z2
```

• 신경망 학습 : 다층 신경망

> 다층 신경망 구현

```
def backprop(self, x, err):
    m = len(x)
    # 은닉층 > 출력층 가중치, 절편 업데이트
    w2_grad = np.dot(self.a1.T, err) / m
    b2_grad = np.sum(err) / m

    # 은닉층 오차
    err_to_hidden = np.dot(err, self.w2.T) * self.a1 * (1 - self.a1)
    # 입력층 > 은닉층 가중치, 절편 업데이트
    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
```

• 신경망 학습 : 다층 신경망

> 다층 신경망 구현

```
def init_weights(self, n_features):
    self.w1 = np.ones((n_features, self.units))
    self.b1 = np.zeros(self.units)
    self.w2 = np.ones((self.units, 1))
    self.b2 = 0
def training(self, x, y, m):
    z = self.forpass(x)
    a = self.activation(z)
    err = -(y - a)
    w1_grad, b1_grad, w2_grad, b2_grad = self.backprop(x, err)
    self.w1 -= w1_grad; self.b1 -= b1_grad
    self.w2 -= w2_grad; self.b2 -= b2_grad
    return a
```

• 신경망 학습 : 다층 신경망

> 다층 신경망 구현

```
def fit(self, x, y, epochs = 100):
    y = y.reshape(-1,1)
    m = len(x)
    self.init_weights(x.shape[1])

    for i in range(epochs):
        a = self.training(x, y, m)
        a = np.clip(a, 1e-10, 1-1e-10)
        loss = np.sum(-(y * np.log(a) + (1 - y) * np.log(1 - a)))
        self.losses.append(loss / m)
```

• 신경망 학습 : 다층 신경망

> 다층 신경망 구현

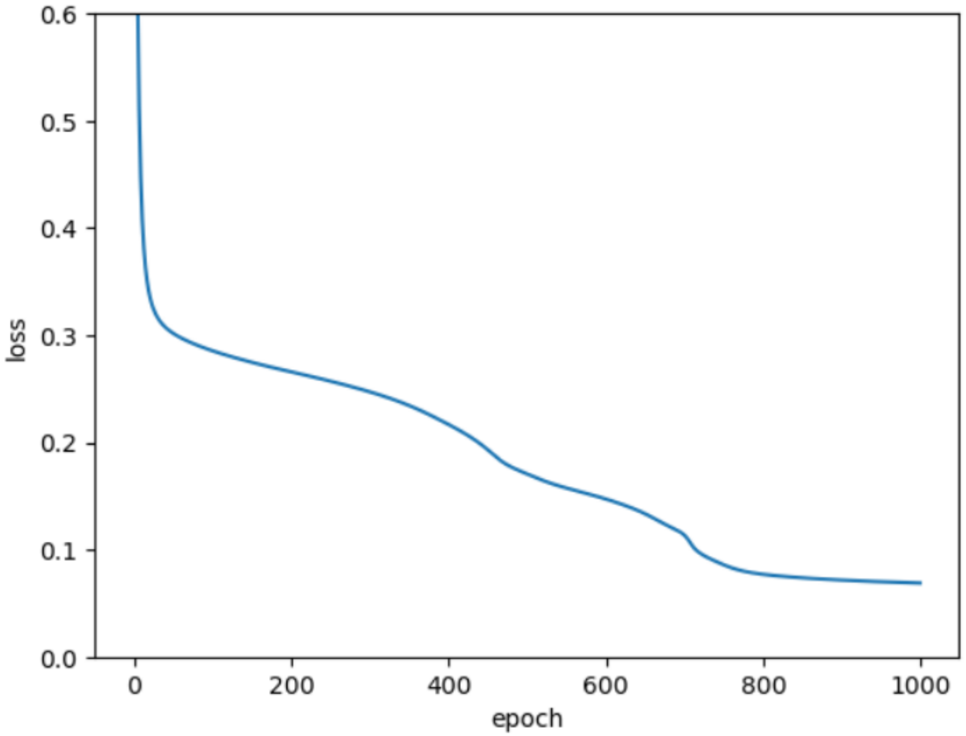
```
dual_layer = DualLayer()
dual_layer.fit(X_train_scaled, y_train, epochs=1000)
print(dual_layer.score(X_test_scaled, y_test))

plt.plot(dual_layer.losses)
plt.ylim(0,0.6)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
print(dual_layer.losses[-1])
```

• 신경망 학습 : 다층 신경망

> 다층 신경망 구현

Out : 0.956140350877193

0.06883682133902377



• 신경망 학습 : 다층 신경망

> 가중치 초기화 개선

```
class RandomInitNetwork(DualLayer):

    def init_weights(self, n_features):
        # 랜덤값 고정
        np.random.seed(0)
        # 평균이 0, 표준편차가 1인 수로 랜덤하게 생성
        self.w1 = np.random.normal(0, 1, (n_features, self.units))
        self.b1 = np.zeros(self.units)
        self.w2 = np.random.normal(0, 1, (self.units, 1))
        self.b2 = 0
```

• 신경망 학습 : 다층 신경망

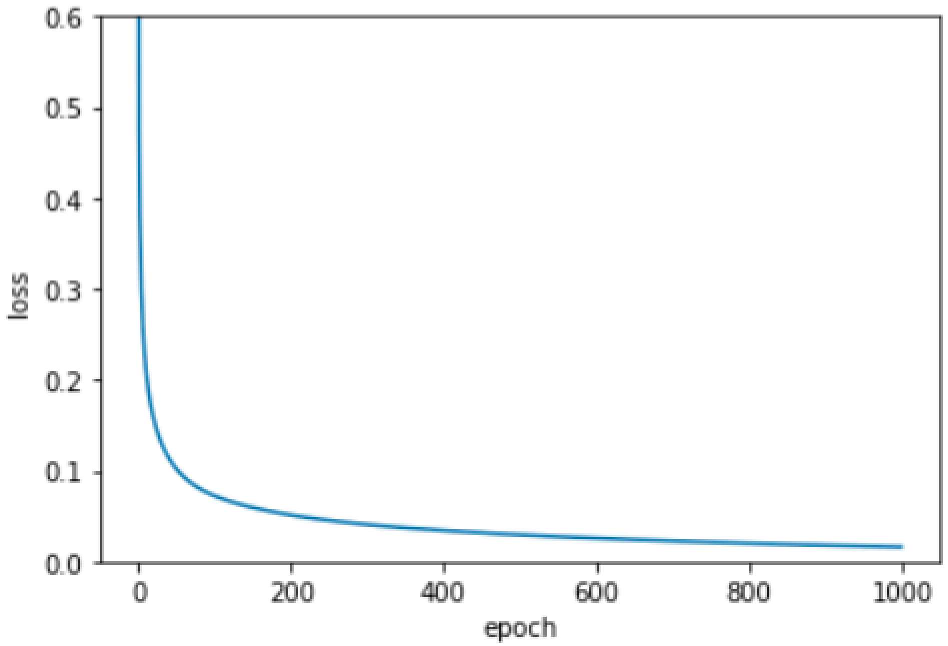
> 가중치 초기화 개선

```
random_init_net = RandomInitNetwork()
random_init_net.fit(X_train_scaled, y_train, epochs = 1000)
plt.plot(random_init_net.losses)
plt.ylim(0,0.6)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
print(random_init_net.losses[-1])
```

• 신경망 학습 : 다층 신경망

> 가중치 초기화 개선

Out : 0.020966117311711403



신경망 학습

• 신경망 학습 : 다층 신경망

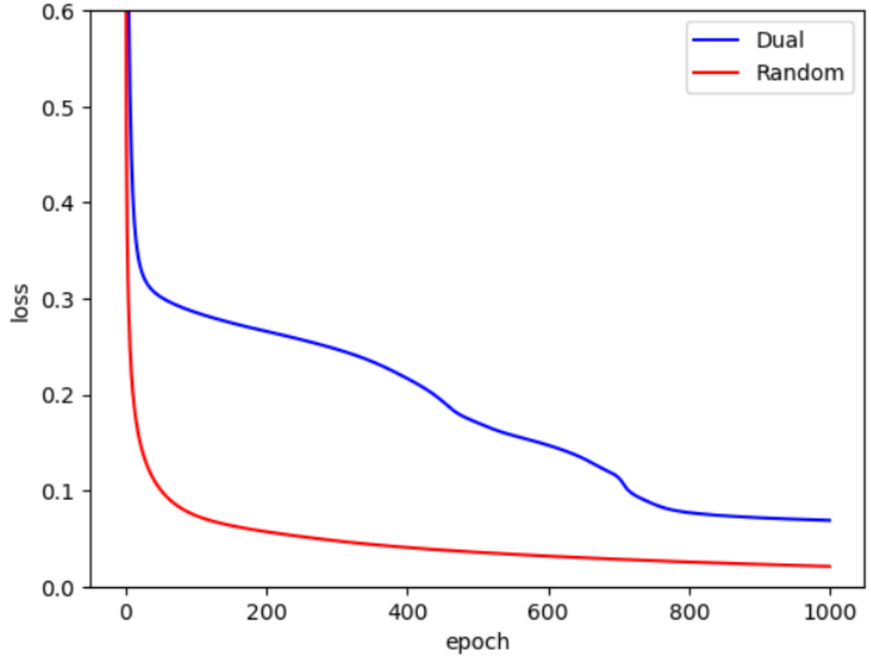
> 가중치 초기화 개선

```
plt.plot(dual_layer.losses, 'b', label = 'Dual')
plt.plot(random_init_net.losses, 'r', label = 'Random')
plt.ylim(0,0.6)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```

• 신경망 학습 : 다층 신경망

> 가중치 초기화 개선

Out :

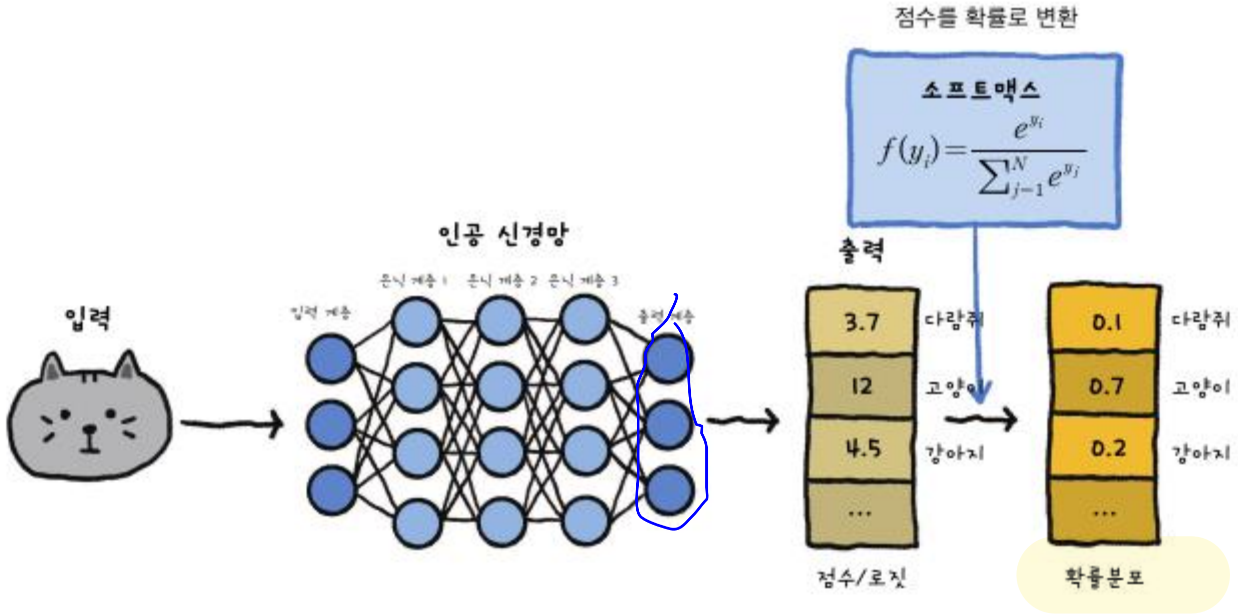


가중치를 1로 고정하고 학습시키는 것보다 무작위로 초기화하는 것이 훨씬 빠르고 매끄럽게 손실 함수 값이 줄어든 것을 확인할 수 있다.

• 신경망 학습 : 다중 분류

> **다중 분류** 마지막 활성함수만 다름 / 소프트맥스

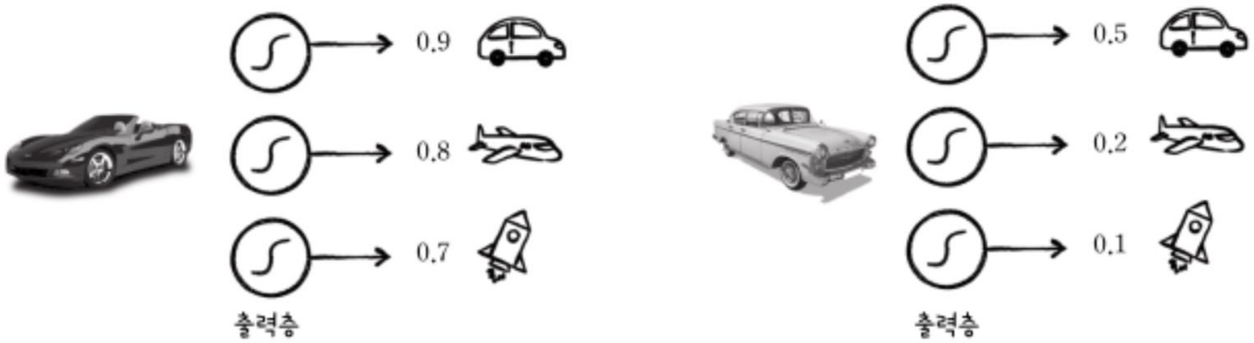
- 타깃 데이터가 2개로 분류하지 않고 여러 클래스로 분류하는 작업 (3개 이상)
- 시그모이드 함수를 사용하지 않고 소프트맥스 함수를 사용하여 출력 결과가 K 개의 클래스를 가진다.



• 신경망 학습 : 다중 분류

> **다중 분류의 문제점**

- 다음은 왼쪽은 시그모이드 함수를 활성 함수로 사용하여 출력한 결과이다.



서로 비교하기가 애매함.

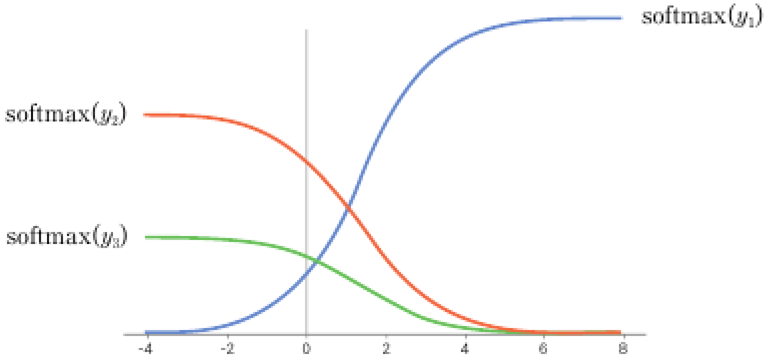
- 왼쪽은 자동차를 0.9 확률로, 오른쪽은 0.5 확률로 예측하였다.
- 0.9 가 0.5보다 크므로 왼쪽이 정확하다 라고 말할 수 없다.

신경망 학습

• 신경망 학습 : 다중 분류

> **소프트맥스** 다중분류일때 활성화 함수: 소프트맥스

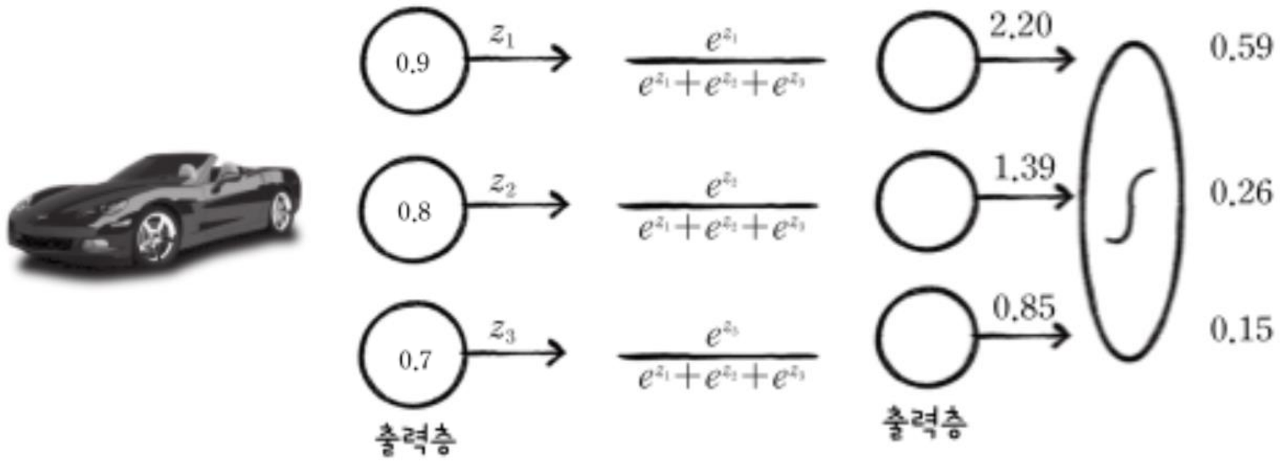
- 소프트맥스 함수는 K개의 클래스를 각각의 확률 벡터로 변환하여 반환한다.
- 다음과 같은 식과 그래프로 나타낼 수 있으면 모든 확률의 합은 1이 된다.



$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}}$$

• 신경망 학습 : 다중 분류

> **소프트맥스를 사용한 정규화 방법**

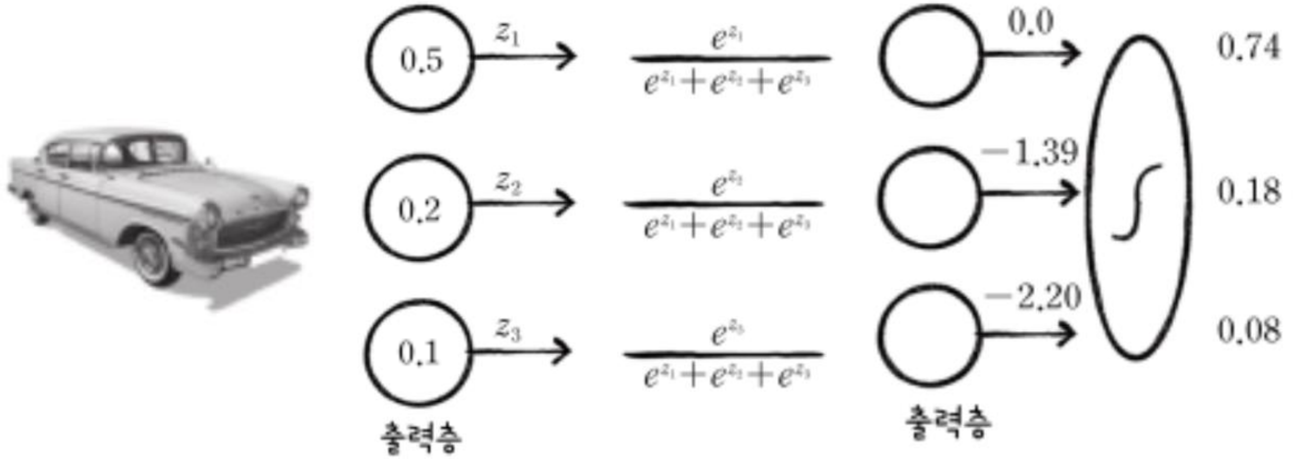


$$z_1=-ln(\frac{1}{0.9}-1)=2.20 \quad z_2=-ln(\frac{1}{0.8}-1)=1.39 \quad z_3=-ln(\frac{1}{0.7}-1)=0.85$$

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

• 신경망 학습 : 다중 분류

> 소프트맥스를 사용한 정규화 방법



$$z_1 = -ln\left(\frac{1}{0.5} - 1\right) = 0.0 \quad z_2 = -ln\left(\frac{1}{0.2} - 1\right) = -1.39 \quad z_3 = -ln\left(\frac{1}{0.1} - 1\right) = -2.20$$

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

• 신경망 학습 : 다중 분류

> **소프트맥스를 사용한 정규화 방법**

- 앞에서 같이 시그모이드 함수보다 소프트맥스 함수를 사용하여 비교하는 것이 공정하게 예측된 결과를 확인할 수 있다.
- 다중 분류에서는 로지스틱 손실 함수의 일반화 버전인 크로스 엔트로피(cross entropy) 손실 함수를 사용하여 소프트맥스 함수 결과값으로 계산한다.

• 신경망 학습 : 다중 분류

> 크로스 엔트로피 손실 함수

- 다음은 크로스 엔트로피 손실 함수의 식이다.

크로스 엔트로피 손실 함수

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

- C는 클래스의 개수이다.
- 이를 로지스틱 손실 함수로 변환하면 y2는 (1 - y1)으로 a2는 (1 – a1)으로 나타낼 수 있다.

로지스틱 손실 함수

$$L=-(ylog(a)+(1-y)log(1-a))$$ 이진분류

- 따라서 위와 같은 식의 손실 함수를 얻을 수 있다.

• 신경망 학습 : 다중 분류

> **엔트로피란?**

- 불확실성의 척도이다.
- 정보이론에서 엔트로피가 높다는 것은 정보가 많고, 확률이 낮다는 것을 의미한다.

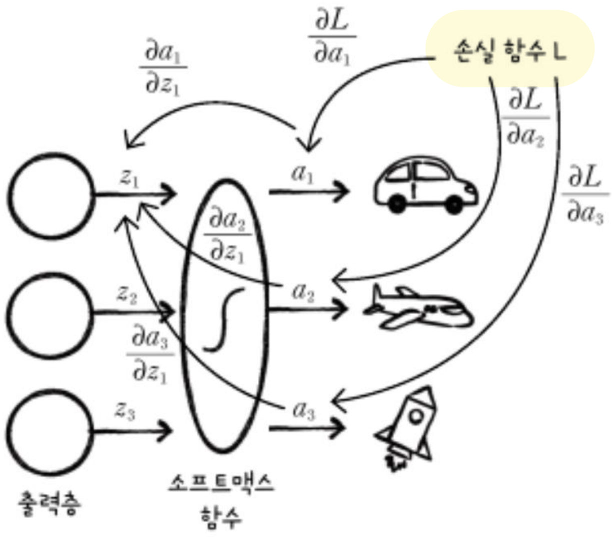
$$H_p(q) = -\sum_{c=1}^{C} q(y_c)log(p(y_c))$$

- 위와 같은 식으로 나타낼 수 있고, 실제 분포 q를 모르는 경우에 확률 분포 p를 통해서 q를 예측하는 것이다.
- 실제 값과 예측 값이 맞는 경우에 0으로 수렴하고 다른 경우 값이 커지기에 값을 줄이기 위한 손실 함수로 사용된다.

• 신경망 학습 : 다중 분류

> 크로스 엔트로피 손실 함수 미분

- 앞선 자동차처럼 3개의 클래스로 분류하는 모델이면 다음과 같이 미분할 수 있다.



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

신경망 학습

- 신경망 학습 : 다중 분류

> **크로스 엔트로피 손실 함수 미분**

- z1에 대한 미분은 다음 순서로 계산된다.

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

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

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

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

• 신경망 학습 : 다중 분류

> **크로스 엔트로피 손실 함수 미분**

- z1에 대한 미분은 다음 순서로 계산된다.

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

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

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

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

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

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

신경망
학습

• 신경망 학습 : 다중 분류

> **크로스 엔트로피 손실 함수 미분**

- z1에 대한 미분은 다음 순서로 계산된다.

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

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

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

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

-error 뭘 하든 에러값이 나옴

신경망 학습

• 신경망 학습 : 다중 분류

> 시그모이드, 소프트맥스 함수 준비

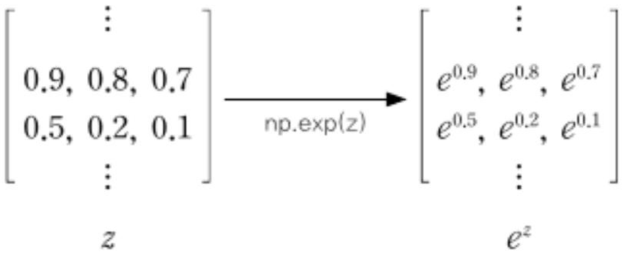
```
def sigmoid(self, z):
    z = np.clip(z, -100, None)
    a = 1 / (1 + np.exp(-z))
    return a

def softmax(self, z):
    z = np.clip(z, -100, None)
    exp_z = np.exp(z)
    return exp_z / np.sum(exp_z, axis = 1).reshape(-1, 1)
```

• 신경망 학습 : 다중 분류

> **소프트맥스 함수 이해**

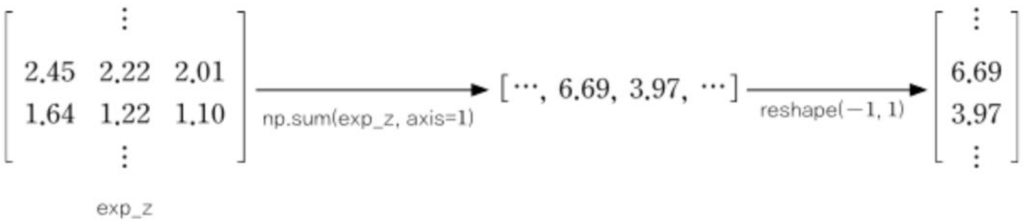
- z의 원소를 e^z로 만들어 주기

```
exp_z = np.exp(z)
```



- 가로의 $e^{0.9}$, $e^{0.8}$, $e^{0.7}$ 다 더해주기(분모)

```
np.sum(exp_z, axis = 1).reshape(-1, 1)
```



• 신경망 학습 : 다중 분류

> 다중 분류 클래스 만들기

```
class MultiClassNetwork:
    def __init__(self, units = 10): # 전과 동일
        self.units = units
        self.w1 = None
        self.b1 = None
        self.w2 = None
        self.b2 = None
        self.a1 = None
        self.losses = []

    def forpass(self, x):
        z1 = np.dot(x, self.w1) + self.b1
        self.a1 = self.sigmoid(z1) # 활성 함수 이름 변경
        z2 = np.dot(self.a1, self.w2) + self.b2
        return z2
```

• 신경망 학습 : 다중 분류

> 다중 분류 클래스 만들기

```
    def backprop(self, x, err): # 전과 동일
        m = len(x)
        w2_grad = np.dot(self.a1.T, err) / m
        b2_grad = np.sum(err) / m
        err_to_hidden = np.dot(err, self.w2.T) * self.a1 * (1 - self.a1)
        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

    def sigmoid(self, z): # 시그모이드 함수
        z = np.clip(z, -100, None)
        a = 1 / (1 + np.exp(-z))
        return a
```

• 신경망 학습 : 다중 분류

> 다중 분류 클래스 만들기

```
def softmax(self, z): #소프트맥스 함수
    z = np.clip(z, -100, None)
    exp_z = np.exp(z)
    return exp_z / np.sum(exp_z, axis = 1).reshape(-1, 1)

def init_weights(self, n_features, n_classes): # 클래스 개수 받음
    np.random.seed(0)
    self.w1 = np.random.normal(0, 1, (n_features, self.units))
    self.b1 = np.zeros(self.units)
    self.w2 = np.random.normal(0, 1, (self.units, n_classes)) # 클래스 개수 포함
    self.b2 = np.zeros(n_classes) # 클래스 개수 포함
```

• 신경망 학습 : 다중 분류

> 다중 분류 클래스 만들기

```
def training(self, x, y, m):
    z = self.forpass(x)
    a = self.softmax(z) # 출력 계층 활성 함수 변경
    err = -(y - a)
    w1_grad, b1_grad, w2_grad, b2_grad = self.backprop(x, err)
    self.w1 -= w1_grad
    self.b1 -= b1_grad
    self.w2 -= w2_grad
    self.b2 -= b2_grad
    return a
```

• 신경망 학습 : 다중 분류

> 다중 분류 클래스 만들기

```
def fit(self, x, y, epochs = 100):
    m = len(x)
    self.init_weights(x.shape[1], y.shape[1]) # 가중치 초기화시 클래스 개수 포함
    for I in range(epochs):
        print('.', end='')   # epochs 1번마다 . 찍음
        a = self.training(x, y, m)
        a = np.clip(a, 1e-10, 1-1e-10)
        loss = np.sum(-y * np.log(a))  # 크로스 엔트로피 손실 함수
        self.losses.append(loss / m)
```

• 신경망 학습 : 다중 분류

> 다중 분류 클래스 만들기

```
def predict(self, x):
    z = self.forpass(x)
    return np.argmax(z, axis = 1)  # 예측한 결과에서 가장 큰 확률 인덱스
def score(self, x, y):
    # 정답 인덱스 확인
    return np.mean(self.predict(x) == np.argmax(y, axis = 1))
```

• 신경망 학습 : 다중 분류

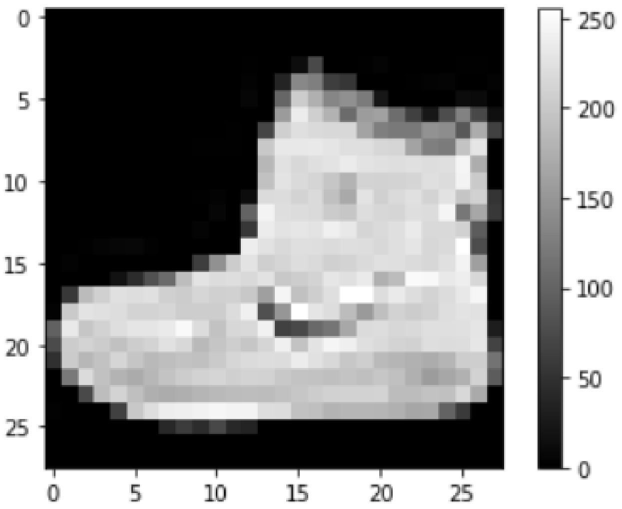
> 다중 클래스로 MNIST 패션 이미지 분류

```
import tensorflow as tf
import matplotlib.pyplot as plt

# 데이터 불러오기
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

# 데이터 확인하기
plt.imshow(X_train[0], cmap='gray')
plt.colorbar()
plt.show()
print(X_train[0][:5])
```

• 신경망 학습 : 다중 분류

> 다중 클래스로 MNIST 패션 이미지 분류

Out :




```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0
    0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
   54   0   0   0   1   3   4   0   0   3]]
```

• 신경망 학습 : 다중 분류

> 다중 클래스로 MNIST 패션 이미지 분류

```
# 데이터 스케일링
X_train = X_train / 255
X_test = X_test / 255

# 28 x 28 > 784 로 변환
X_train = X_train.reshape(-1,784)
X_test = X_test.reshape(-1,784)
print(X_train.shape)
```

Out : (60000, 784)

• 신경망 학습 : 다중 분류

> 다중 클래스로 MNIST 패션 이미지 분류

```
# 타깃 데이터 확인
print(y_train[:5])
print(np.unique(y_train))

# 클래스 이름
class_name = ['티셔츠/윗도리', '바지', '스웨터', '드레스',
              '코트', '샌들', '셔츠', '스니커즈', '가방', '앵클부츠']
```

Out : [9 0 0 3 0]

[0 1 2 3 4 5 6 7 8 9]

• 신경망 학습 : 다중 분류

> 다중 클래스로 MNIST 패션 이미지 분류

```
# 원-핫 인코딩 (텐서플로우 지원 함수)
y_train_encoded = tf.keras.utils.to_categorical(y_train)
y_test_encoded = tf.keras.utils.to_categorical(y_test)
print(y_train_encoded)
```

Out :

```
[[0. 0. 0. ... 0. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

• 신경망 학습 : 다중 분류

> 다중 클래스로 MNIST 패션 이미지 분류

```
# 다중 클래스 학습
multiclass = MultiClassNetwork(units = 100)
multiclass.fit(X_train, y_train_encoded, epochs = 100)
multiclass.score(X_test, y_test_encoded)
```

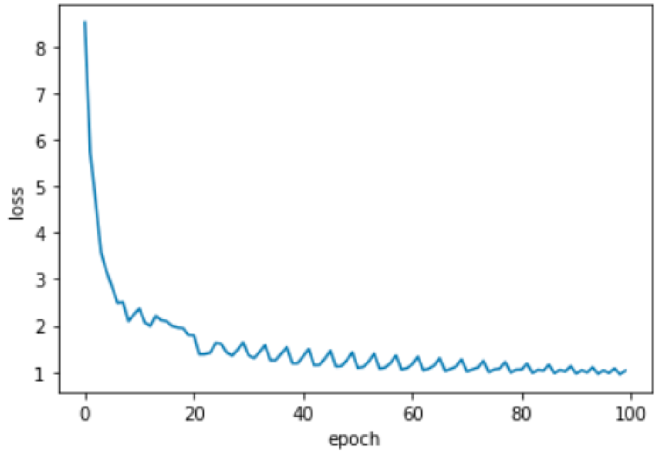
Out : 0.6959

• 신경망 학습 : 다중 분류

> 다중 클래스로 MNIST 패션 이미지 분류

```
plt.plot(multiclass.losses)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
print(multiclass.losses[-1])
```

Out : 1.0323766749003016



신경망 학습

• 신경망 학습 : 다중 분류

> 다중 클래스로 MNIST 패션 이미지 분류

```
pred = multiclass.predict(X_test[0].reshape(1,-1))
plt.imshow(X_test[0].reshape(28,28), cmap='gray')
print('예측 :', class_name[pred[0]])
print('실제 :', class_name[y_test[0]])
```

Out : 예측 : 스니커즈 실제 : 앵클부츠