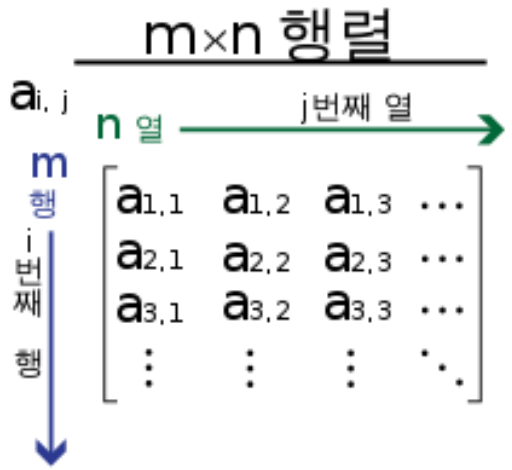
밑바닥부터 시작하는 딥러닝2

CHAPTER 1. 신경망 복습

20180376 안제준

1-1 수학과 파이썬 복습

- 1. 벡터와 행렬(Matrix)




```
1 import numpy as np
2 A = np.array([[1,2],[3,4],[5,6]])
```

```
[17]  1 A.shape
```

```
(3, 2)
```

1-1 수학과 파이썬 복습

- 2. 행렬의 원소별 연산(element-wise)

```
1 import numpy as np
2 A = np.array([[1,2],[3,4]])
3 B = np.array([[5,6],[7,8]])
```

```
1 A + B
```

```
array([[ 6,  8],
       [10, 12]])
```

```
1 A * B
```

```
array([[ 5, 12],
       [21, 32]])
```

1-1 수학과 파이썬 복습

- 3. 브로드캐스트

```
[1]   1 import numpy as np
      2 A = np.array([[1,2],[3,4]])
      3 B = np.array([[5,6],[7,8]])
```

```
[6]   1 A * 10
```

```
array([[10, 20],
       [30, 40]])
```

```
[9]   1 A = np.arange(10).reshape(2,1,5)
      2 B = np.arange(20).reshape(2,2,5)
      3 A * B
```

```
array([[[ 0,  1,  4,  9, 16],
        [ 0,  6, 14, 24, 36]],

       [[ 50,  66,  84, 104, 126],
        [ 75,  96, 119, 144, 171]]])
```

```
[10]  1 A = np.arange(10).reshape(2,5,1)
      2 B = np.arange(20).reshape(2,2,5)
      3 A * B
```

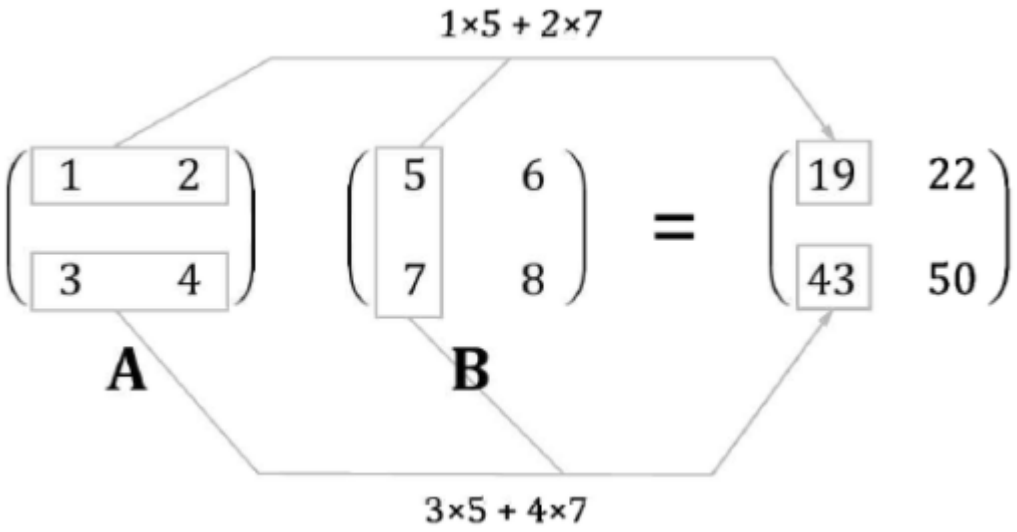
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-83e04aa0484f> in <module>()
      1 A = np.arange(10).reshape(2,5,1)
      2 B = np.arange(20).reshape(2,2,5)
----> 3 A * B

ValueError: operands could not be broadcast together with shapes (2,5,1) (2,2,5)
```

1-1 수학과 파이썬 복습

- 4. 벡터의 내적과 행렬의 곱

벡터의 내적 $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$

행렬의 곱셈



1-1 수학과 파이썬 복습

- 4. 벡터의 내적과 행렬의 곱

```
[11]  1 import numpy as np
      2 A = np.array([[1,2],[3,4]])
      3 B = np.array([[5,6],[7,8]])
```

```
[12]  1 np.matmul(A, B)
```

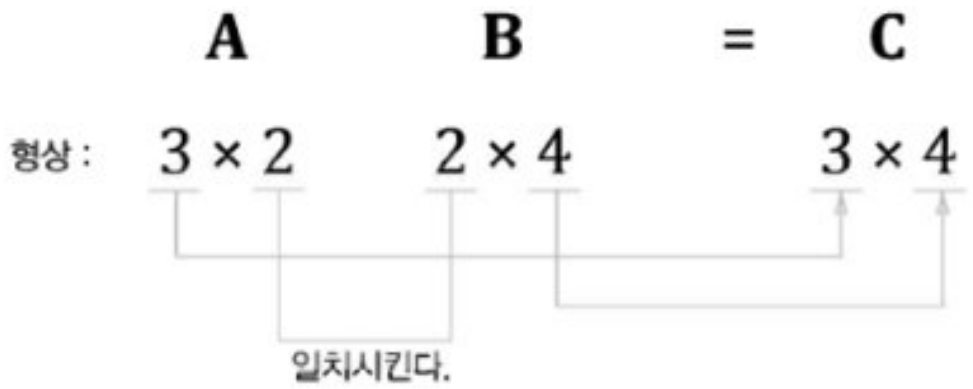
```
array([[19, 22],
       [43, 50]])
```

```
[13]  1 A @ B
```

```
array([[19, 22],
       [43, 50]])
```

1-1 수학과 파이썬 복습

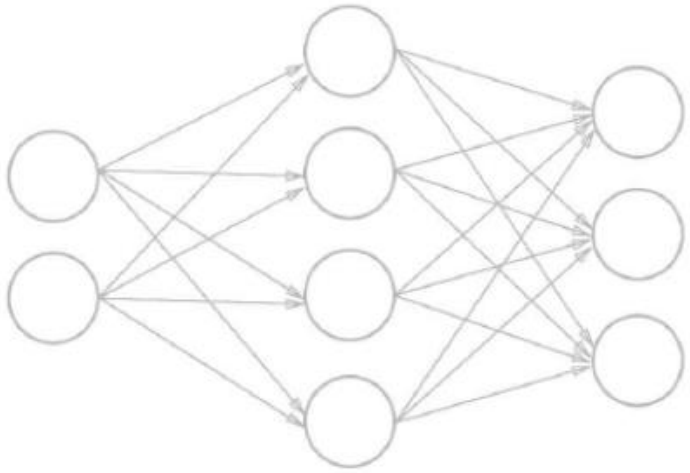
- 5. 행렬의 형상 확인



1-2 신경망의 추론

- 1. 신경망 추론 전체 그림

그림 1-7 신경망의 예




```
[18]  1 # Inpur -> hidden
      2 W1 = np.random.randn(2, 4)  # 가중치
      3 b1 = np.random.randn(4)
      4 x = np.random.randn(10, 2)
      5 h = np.matmul(x, W1) + b1  # b1 은 브로드캐스팅 됨
```

```
[19]  1 h.shape
```

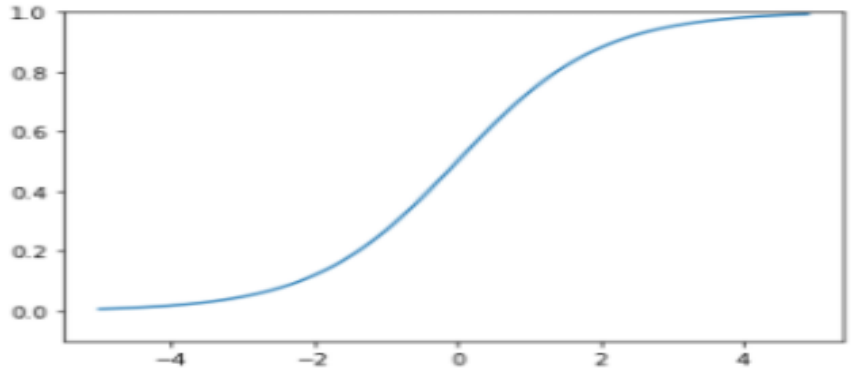
```
(10, 4)
```

h = x@W + b

1-2 신경망의 추론

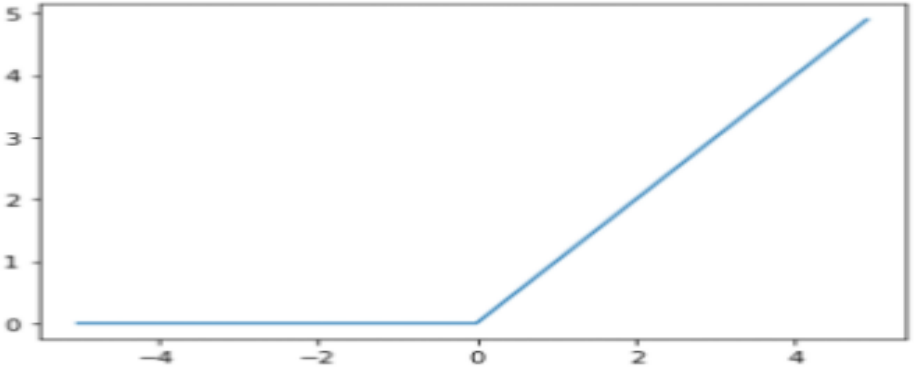
- 1. 신경망 추론 전체 그림

Sigmoid Function



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

ReLU Function

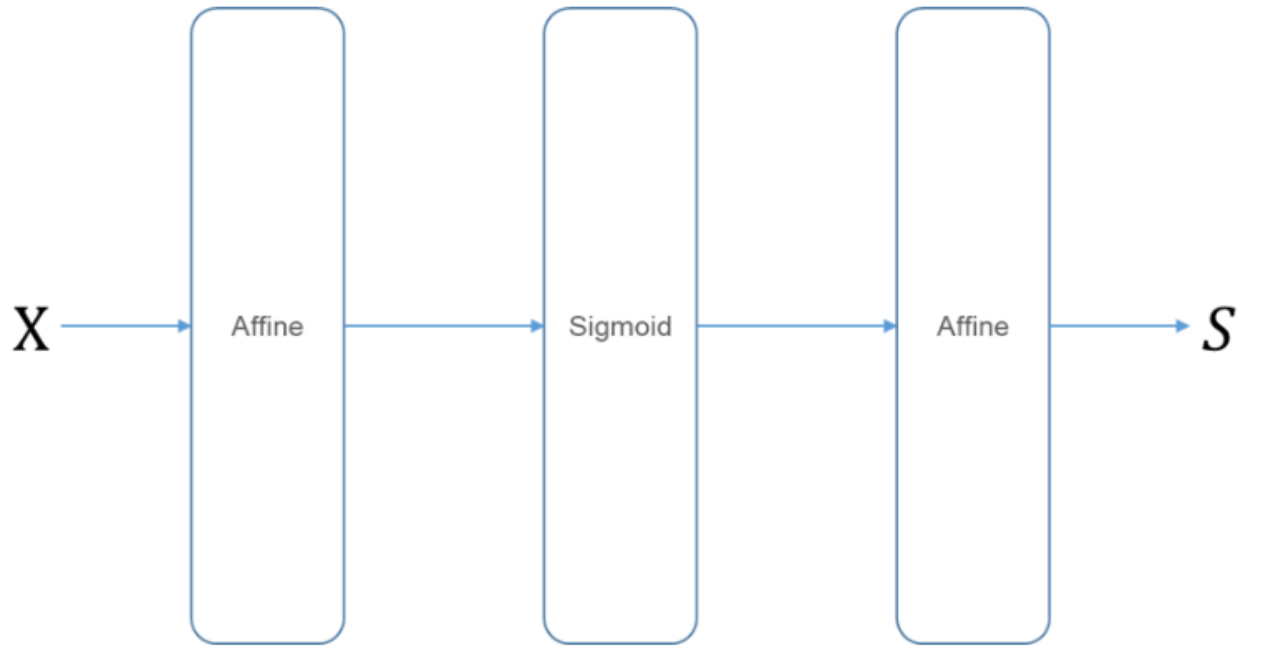


y = x (when x>=0), 0 (when x<0)

1-2 신경망의 추론

- 2. Forward propagation(순전파)

```python
# 시그모이드(Sigmoid) 레이어 구현
class Sigmoid:
    def __init__(self):
        self.params = []

    def forward(self, x):
        return 1 / (1 + np.exp(-x))
# 완전연결계층(Affine) 구현
class Affine:
    def __init__(self, W, b):
        self.params = [W, b]

    def forward(self, x):
        W, b = self.params
        out = np.matmul(x, W) + b
        return out
```



1-3 신경망의 학습

- 1. 손실 함수(Loss Function)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$\text{Loss} = -\frac{1}{\text{output size}}\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

1-3 신경망의 학습

- 2. 미분과 기울기

$$f(x_1, x_2, \ldots, x_n) = (f_1, f_2, \ldots, f_m)$$

$$J_f = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

1-3 신경망의 학습

- 3. 연쇄 법칙(Chain Rule)
  합성함수 미분법

$$t = x + y$$

$$z = t^2$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

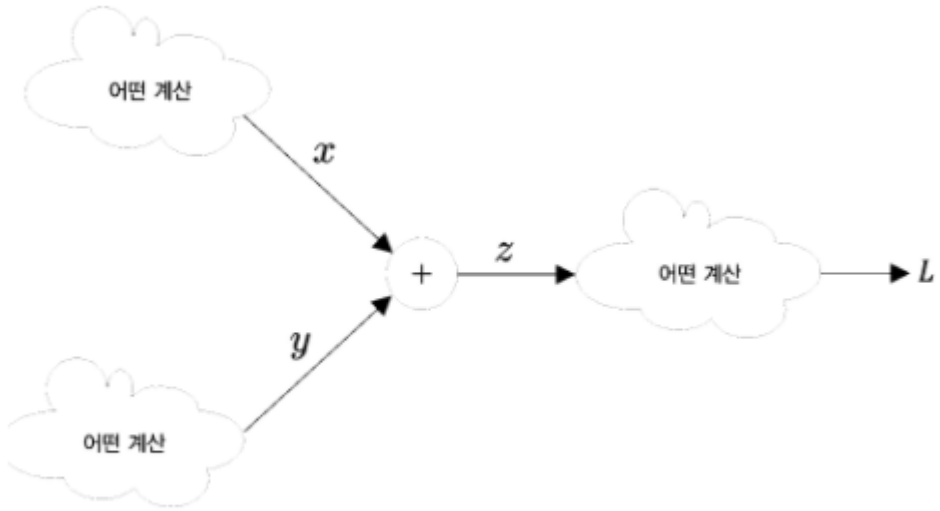
$$\frac{\partial z}{\partial t} = 2t$$

$$\frac{\partial t}{\partial x} = 1$$

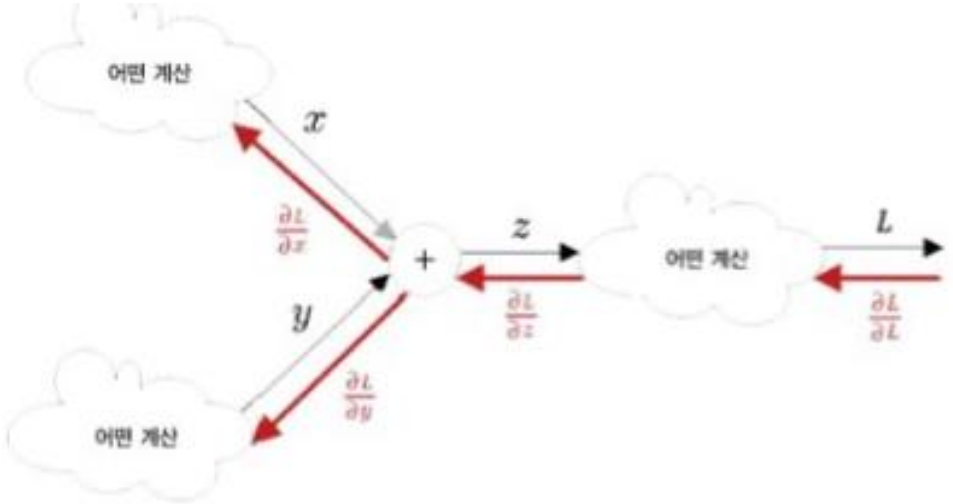
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x + y)$$

1-3 신경망의 학습

- 4. 계산 그래프



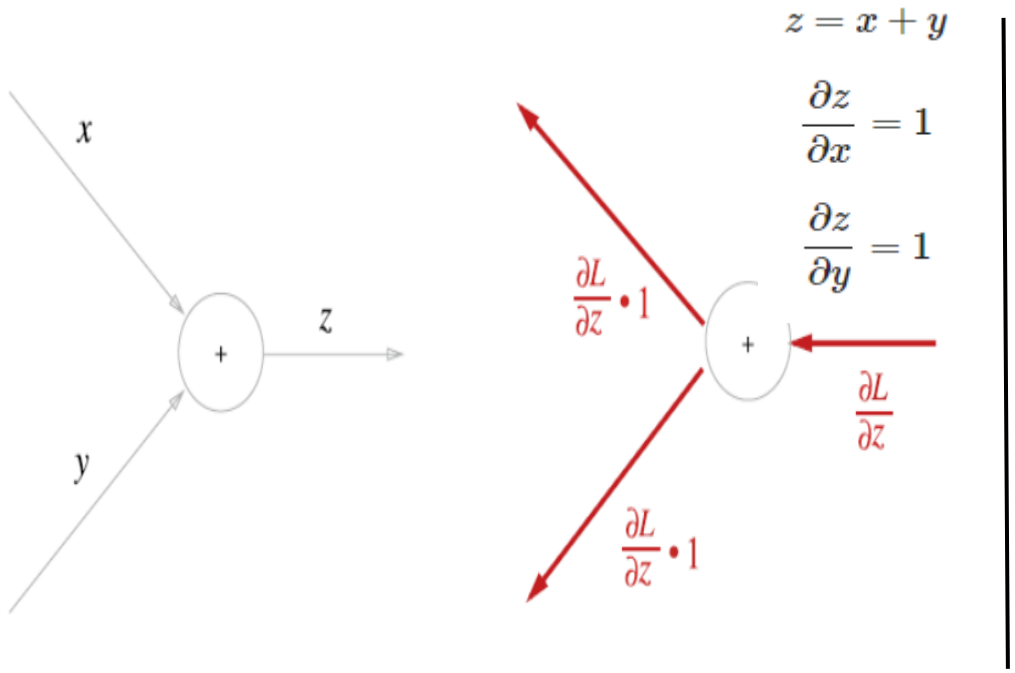
순전파의 계산 그래프



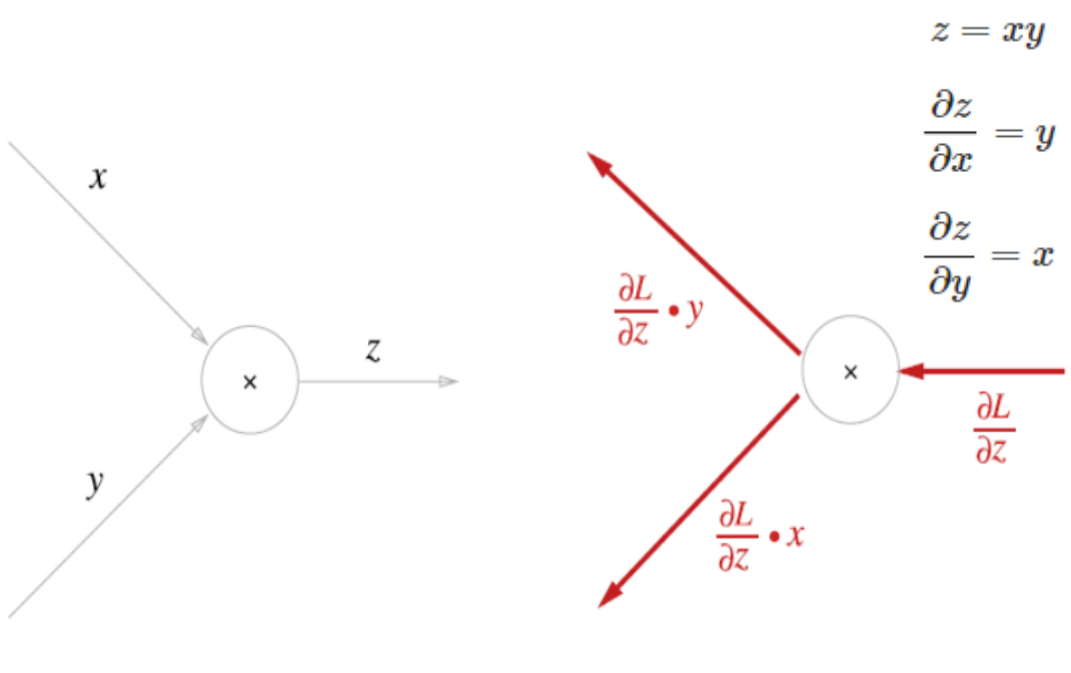
역전파가 이루어지는 과정

1-3 신경망의 학습

- 4. 계산 그래프 (덧셈 노드, 곱셈 노드)



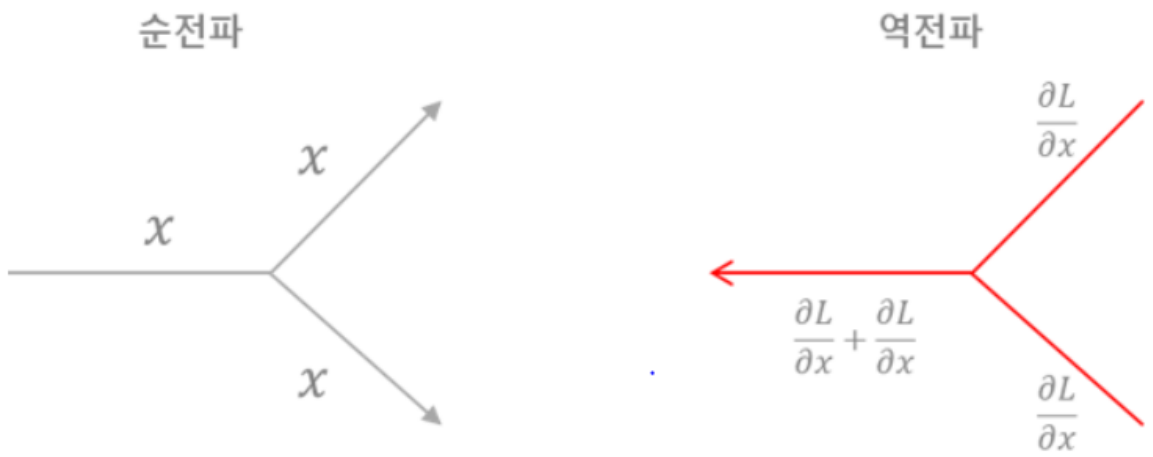
덧셈 노드



곱셈 노드

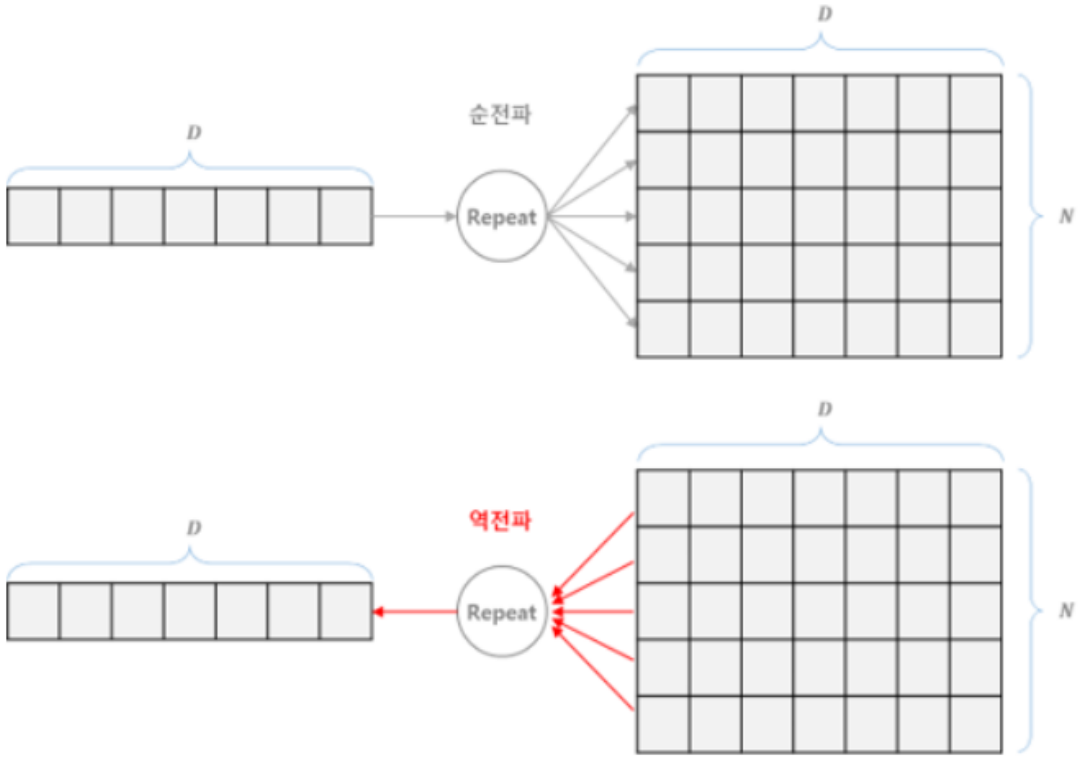
1-3 신경망의 학습

- 4. 계산 그래프 (분기 노드)



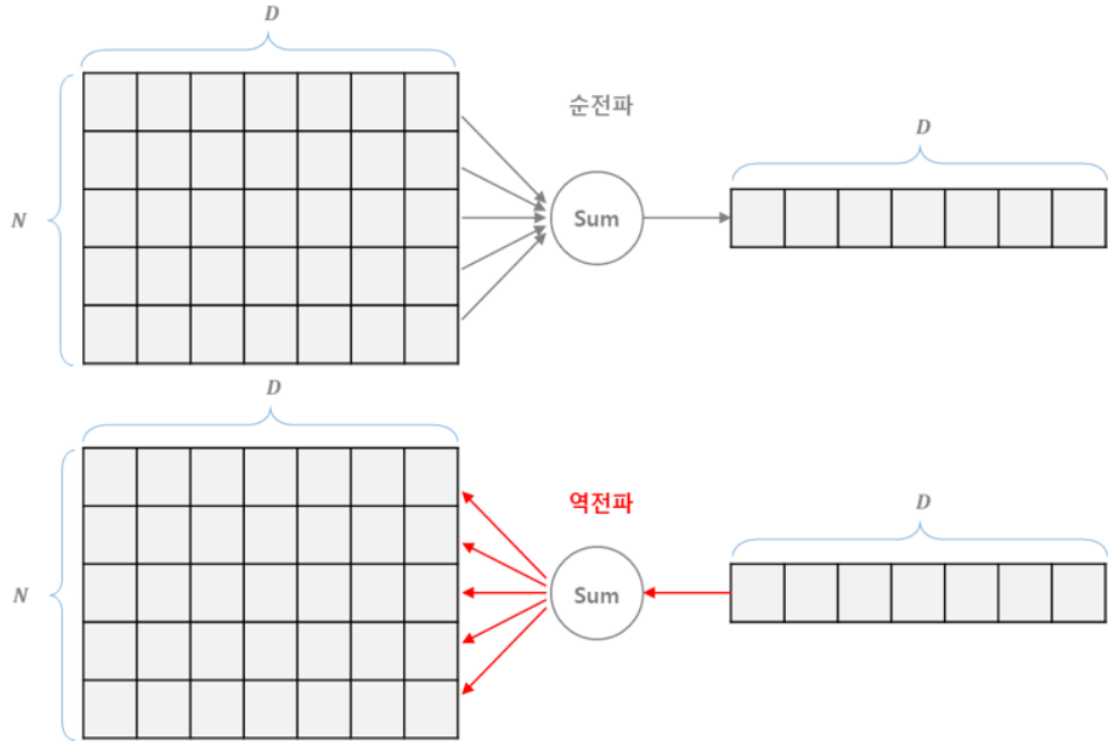
1-3 신경망의 학습

- 4. 계산 그래프 (Repeat 노드)



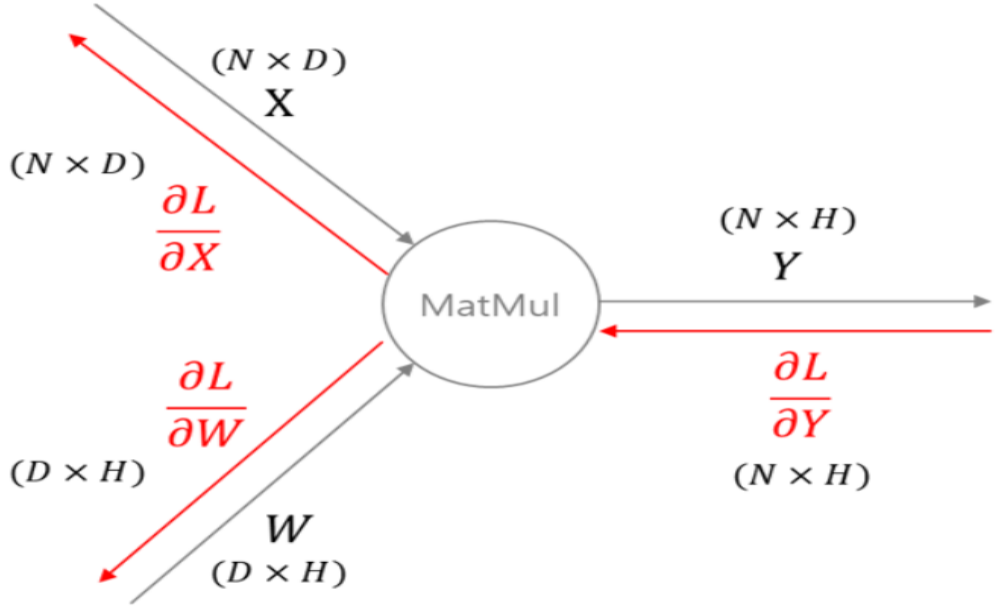
1-3 신경망의 학습

- 4. 계산 그래프 (Sum 노드)



1-3 신경망의 학습

- 4. 계산 그래프 (MatMul 노드)



$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}}\mathbf{W}^T$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{x}^T\frac{\partial L}{\partial \mathbf{y}}$$

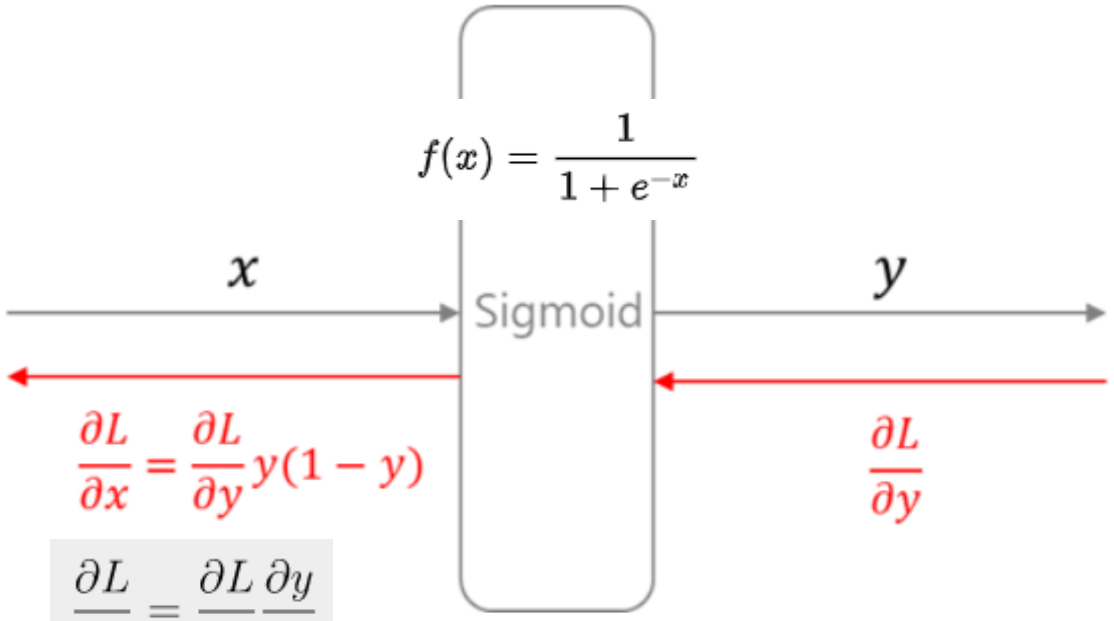
$$\underset{N \times D}{\frac{\partial L}{\partial \mathbf{x}}} = \underset{N \times H}{\frac{\partial L}{\partial \mathbf{y}}} \underset{H \times D}{\mathbf{W}^{\mathrm{T}}}$$

$$\underset{D \times H}{\frac{\partial L}{\partial \mathbf{W}}} = \underset{D \times N}{\mathbf{x}^{\mathrm{T}}} \underset{N \times H}{\frac{\partial L}{\partial \mathbf{y}}}$$

1-3 신경망의 학습

- 5. 기울기 도출과 역전파 구현(Sigmoid 계층)

Sigmoid 계층

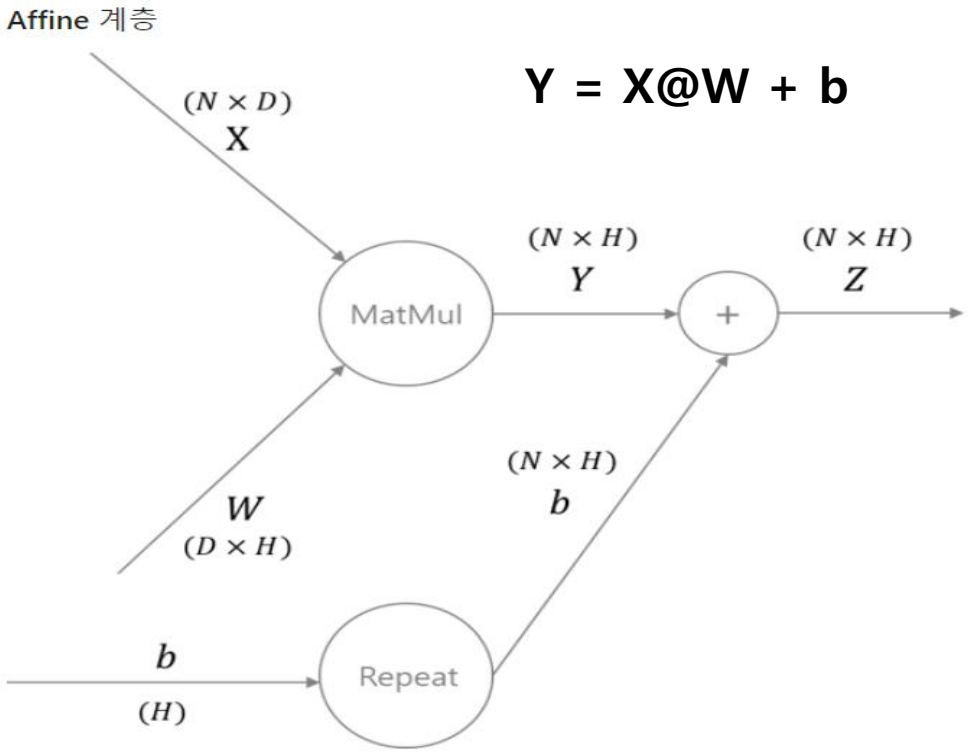


$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

```
def backward(self, dout):
    dx = dout * (1.0 - self.out) * self.out
    return dx
```

1-3 신경망의 학습

- 5. 기울기 도출과 역전파 구현(Affine 계층)

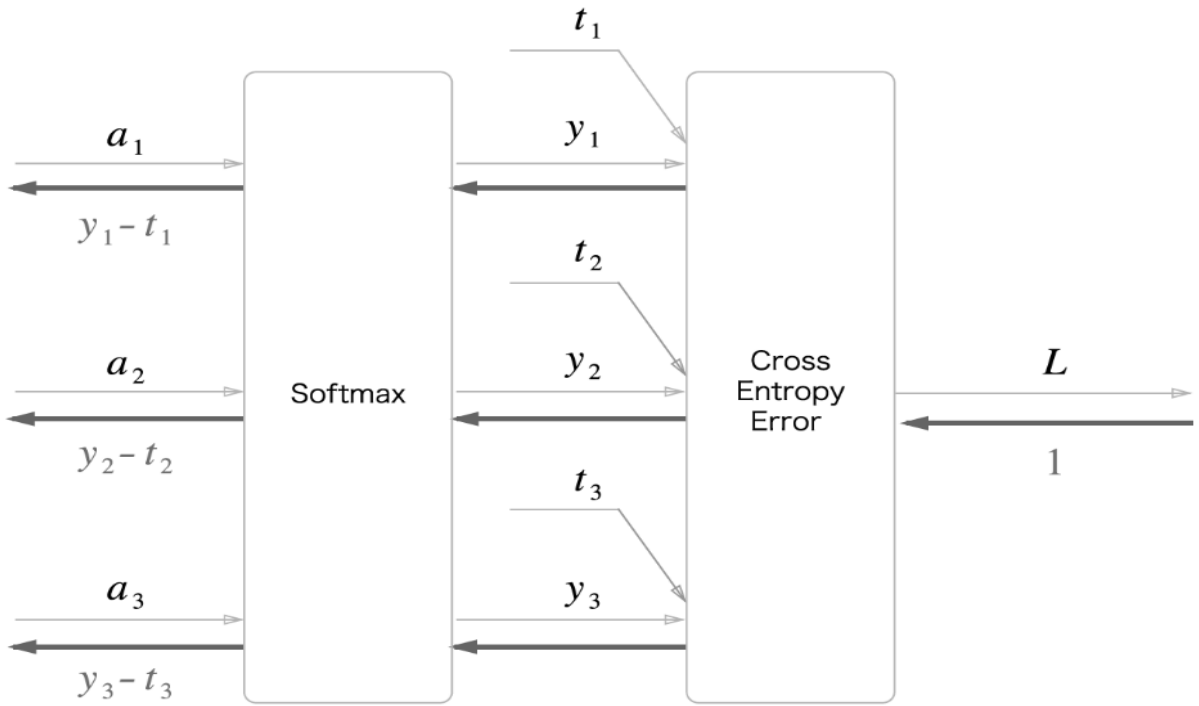



```
def backward(self, dout):
    W, b = self.params
    dx = np.matmul(dout, W.T)
    dW = np.matmul(self.x.T, dout)
    db = np.sum(dout, axis=0)

    self.grads[0][...] = dW
    self.grads[1][...] = db
    return dx
```

1-3 신경망의 학습

- 5. 기울기 도출과 역전파 구현(Softmax with Loss 계층)

Softmax with Loss 계층




```python
def backward(self, dout=1):
    batch_size = self.t.shape[0]

    dx = self.y.copy()
    dx[np.arange(batch_size), self.t] -= 1
    dx *= dout
    dx /= batch_size

    return dx
```

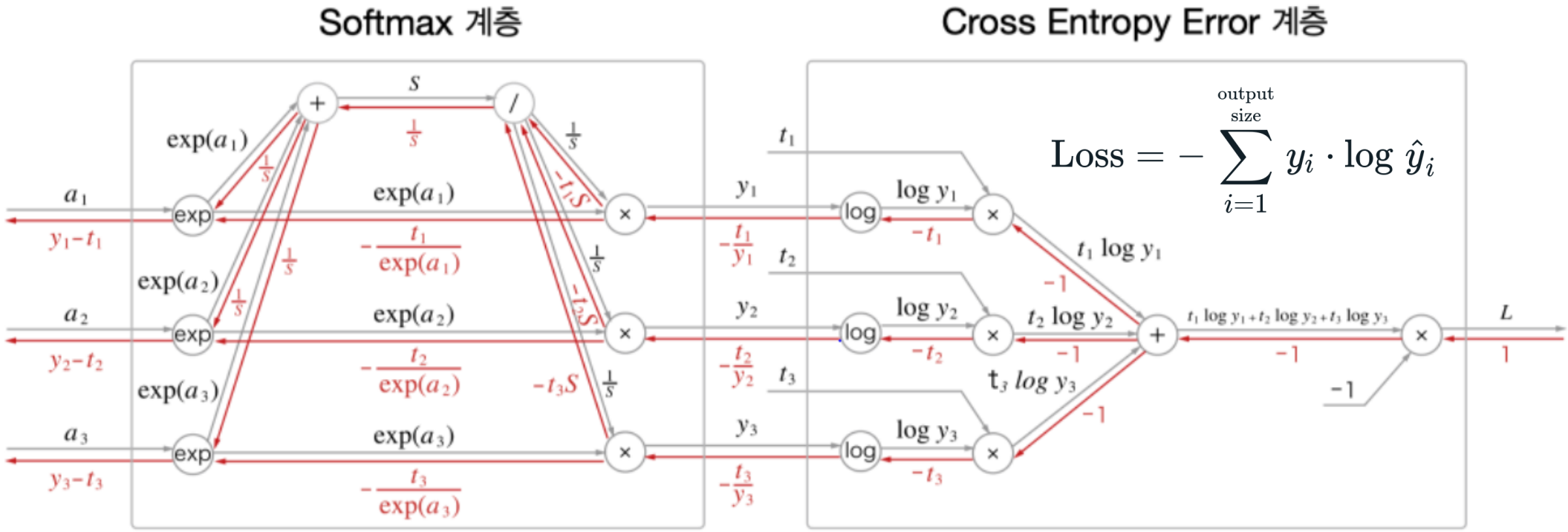
$$p_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

Softmax Function

$$= \frac{e^{x_j}}{e^{x_1}+e^{x_2}+\cdots+e^{x_K}} \quad for\ j = 1,\ldots,K$$

1-3 신경망의 학습

- 5. 기울기 도출과 역전파 구현(Softmax with Loss 계층)



1-3 신경망의 학습

- 6. 가중치 갱신

1단계 -> mini-batch => 훈련 데이터 중에서 무작위로 다수의 데이터를 선정

2단계 -> 기울기 계산 => backpropagation으로 각 가중치 매개변수에 대한 손실 함수의 기울기 계산

3단계 -> 매개변수 갱신 => 기울기를 사용하여 매개변수 갱신

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

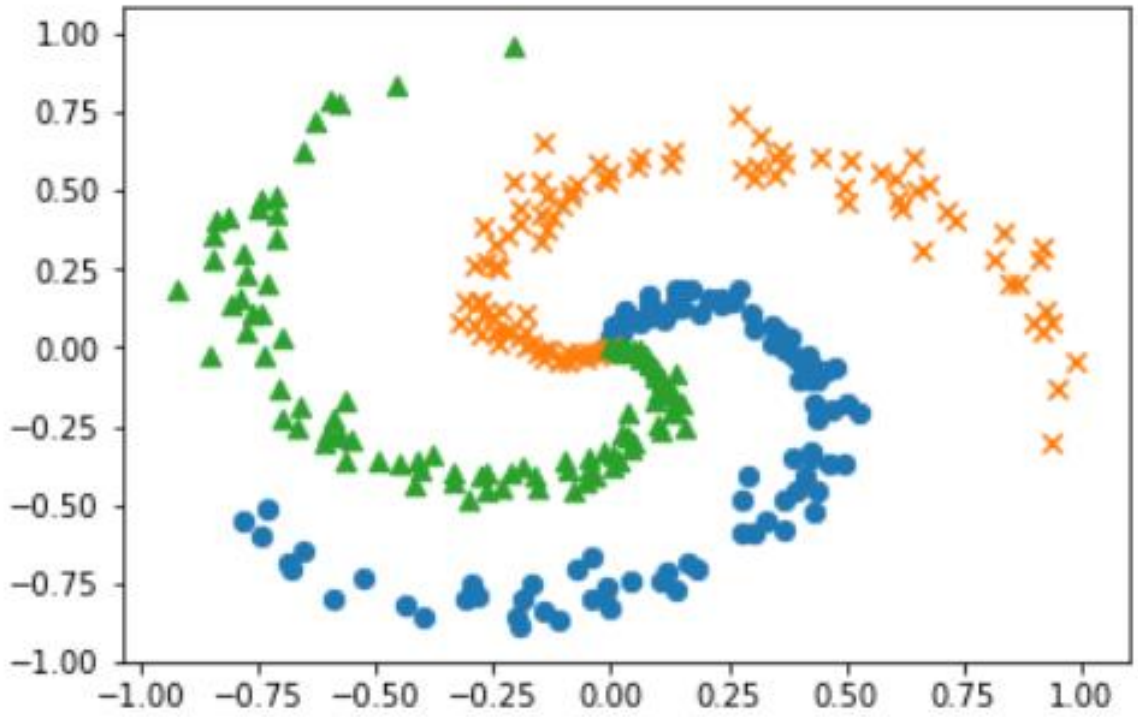
4단계 -> 반복

```python
class SGD:
    '''
    확률적 경사하강법(SGD, Stochastic Gradient Descent)
        W <- W - lr * (dL/dW)
    '''
    def __init__(self, lr=0.01):
        self.lr = lr  # learning rate

    def update(self, params, grads):
        for i in range(len(params)):
            params[i] -= self.lr * grads[i]
```

1-4 신경망으로 문제를 풀다

- 1. 스파이럴 데이터셋

```python
import sys
sys.path.append('..')  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import matplotlib.pyplot as plt

from dataset import spiral

x, t = spiral.load_data()
print('x', x.shape)  # (300, 2)
print('t', t.shape)

# 데이터점 플롯
N = 100
CLS_NUM = 3
markers = ['o', 'x', '^']
for i in range(CLS_NUM):
    plt.scatter(x[i*N:(i+1)*N, 0], x[i*N:(i+1)*N, 1], s=40, marker=markers[i])
plt.show()
```



1-4 신경망으로 문제를 풀다

- 2. 신경망 구현

```python
import sys
sys.path.append('..')
from common.np import *
from common.layers import Affine, Sigmoid, SoftmaxWithLoss


class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size):
        I, H, O = input_size, hidden_size, output_size

        # 가중치와 편향 초기화
        W1 = 0.01 * np.random.randn(I, H)
        b1 = np.zeros(H)
        W2 = 0.01 * np.random.randn(H, O)
        b2 = np.zeros(O)

        # 레이어 생성
        self.layers = [
            Affine(W1, b1),
            Sigmoid(),
            Affine(W2, b2)
        ]
        self.loss_layer = SoftmaxWithLoss()

        # 모든 가중치와 기울기를 리스트에 모은다.
        self.params, self.grads = [], []
        for layer in self.layers:
            self.params += layer.params
            self.grads += layer.grads
```

```python
def predict(self, x):
    for layer in self.layers:
        x = layer.forward(x)
    return x

def forward(self, x, t):
    score = self.predict(x)
    loss = self.loss_layer.forward(score, t)
    return loss

def backward(self, dout=1):
    dout = self.loss_layer.backward(dout)
    for layer in reversed(self.layers):
        dout = layer.backward(dout)
    return dout
```

1-4 신경망으로 문제를 풀다

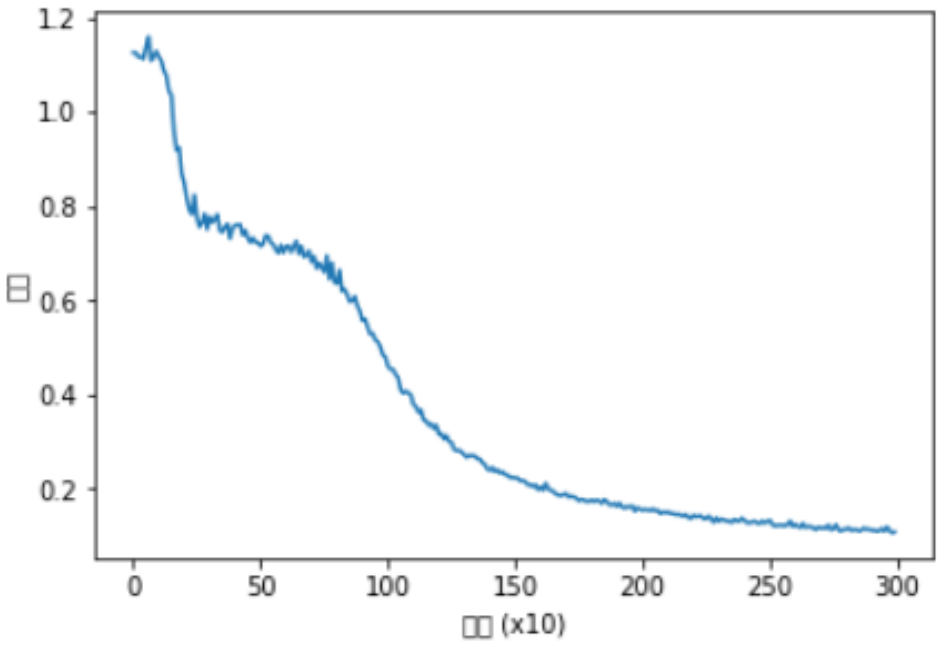
- 3. 학습용 코드

```
# 1. 하이퍼파라미터 설정
max_epoch = 300
batch_size = 30
hidden_size = 10
learning_rate = 1.0

# 2. 데이터 읽기, 모델과 옵티마이저 생성
x, t = spiral.load_data()
model = TwoLayerNet(input_size=2,
                    hidden_size=hidden_size,
                    output_size=3)
optimizer = SGD(lr=learning_rate)

# 학습에 사용하는 변수
data_size = len(x)
max_iters = data_size // batch_size
total_loss = 0
loss_count = 0
loss_list = []
```

```
for epoch in tqdm(range(max_epoch)):
    # 3. 데이터 셔플링
    idx = np.random.permutation(data_size)
    x = x[idx]
    t = t[idx]

    for iters in range(max_iters):
        batch_x = x[iters*batch_size:(iters+1)*batch_size]
        batch_t = t[iters*batch_size:(iters+1)*batch_size]

        # 기울기를 구해 매개변수 갱신
        loss = model.forward(batch_x, batch_t)
        model.backward()
        optimizer.update(model.params, model.grads)

        total_loss += loss
        loss_count += 1

        # 정기적으로 학습 경과 출력
        if (iters+1) % 10 == 0:
            avg_loss = total_loss / loss_count
            print(f'| 에폭 {epoch+1} | 반복{iters+1}/{max_iters} | 손실 {avg_loss:.2f}')
            loss_list.append(avg_loss)
            total_loss, loss_count = 0, 0
```

1-4 신경망으로 문제를 풀다

- 3. 학습용 코드 -결과



1-4 신경망으로 문제를 풀다

- .4 Trainer클래스

```
trainer = Trainer(model, optimizer)
trainer.fit(x, t, max_epoch, batch_size, eval_interval=10)
trainer.plot()
```

1-5 계산 고속화

- 1. 비트 정밀도

  numpy의 datatype은 float 64bit이다.
  그러나 신경망의 추론과 학습은 32bit로도 문제없이 수행이 가능하다.

  32bit로 바꾸면 메모리 측면이나 계산 속도 측면에서도 빠르다.

  사실 16bit도 상관없지만, 일반적으로 CPU와 GPU는 연산 자체를 32비트로 수행합니다.

  그래서 학습된 데이터 정도에서는 16bit로 수행하는 것이 효과를 볼 수 있지만, 나머지 연산에서는 그렇게 큰 효과를 보지 못합니다.

1-5 계산 고속화

- 2. GPU(쿠파이)

딥러닝의 계산은 대량의 곱하기 연산으로 구성되는데 이를 병렬 계산으로 지원해주는 것 -> CUPY

하지만, 아쉽게도 CUPY는 NVIDIA의 GPU에서만 동작함

사용법은 NUMPY와 동일 (연산만 GPU이용해서 병렬계산)

1-6 정리

- 신경망은 입력층, 은닉층(중간층), 출력층을 지닌다.
- 완전연결계층에 의해 선형변환이 이뤄지고, 활성화 함수에 의해 비선형 변환이 이뤄진다.
- 완전연결계층이나 미니배치 처리는 행렬로 모아 한꺼번에 계산할 수 있다.
- 오차역전파법을 사용하여 신경망의 손실에 관한 기울기를 효율적으로 구할 수 있다.
- 신경망이 수행하는 처리는 계산 그래프로 시각화할 수 있으며, 순전파와 역전파를 이해하는 데 도움이 된다.
- 신경망의 구성요소를 '계층(layer)'으로 모듈화해두면 이를 조립하여 신경망을 쉽게 구성할 수 있다.
- 신경망 고속화에는 GPU를 이용한 병렬계산과 데이터의 비트 정밀도가 중요하다.