밑바닥부터 시작하는 딥러닝 2

1. 신경망 복습

최혜원

1.1 수학과 파이썬 복습

.1 백터와 행렬

그림 1-1 벡터와 행렬의 예

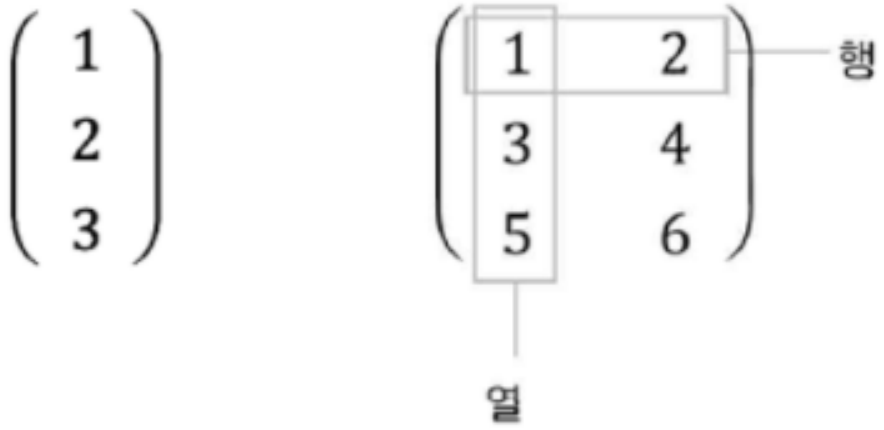


그림 1-2 벡터의 표현법

1.1 수학과 파이썬 복습

.2 행렬의 원소별 연산

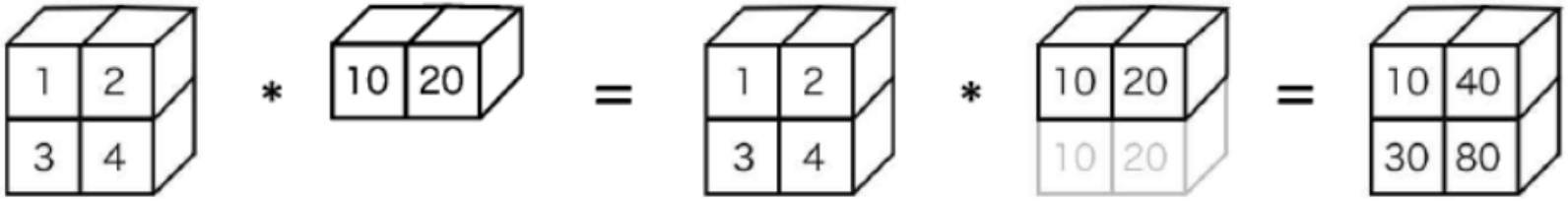
```
In [1]:
1 import numpy as np

In [2]:
1 A = np.array([[1,2,3],[4,5,6]])
2 X = np.array([[0,1,2],[3,4,5]])

In [3]:
1 A + X

Out[3]:
array([[ 1,  3,  5],
       [ 7,  9, 11]])

In [4]:
1 A * X

Out[4]:
array([[ 0,  2,  6],
       [12, 20, 30]])
```

1.1 수학과 파이썬 복습

.3 브로드캐스트

```
In [1]:  1  import numpy as np

In [2]:  1  A = np.array([[1,2],[3,4]])

In [3]:  1  A * 10

Out[3]: array([[10, 20],
               [30, 40]])
```

그림 1-4 브로드캐스트의 예 2



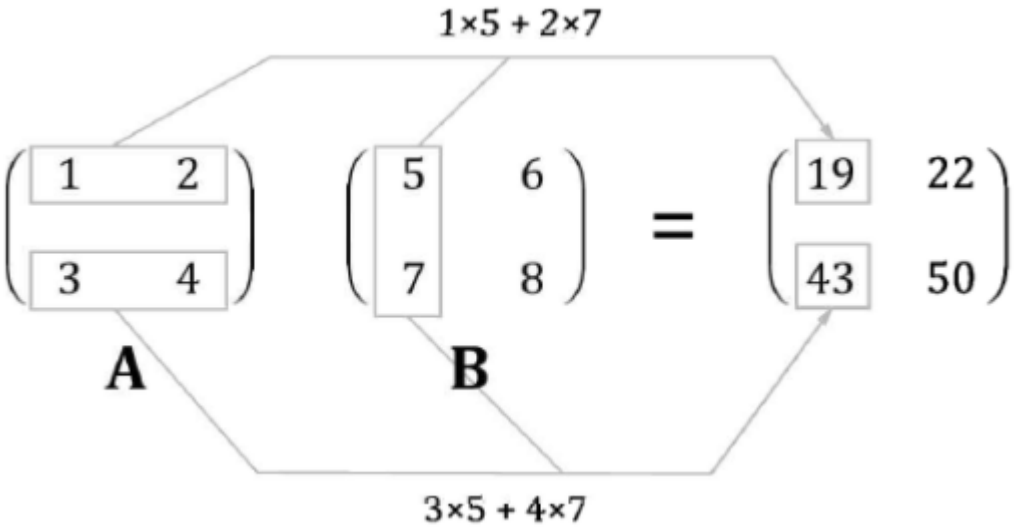
1.1 수학과 파이썬 복습

.4 벡터의 내적과 행렬의 곱

벡터의 내적

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n \qquad [식\ 1.1]$$

행렬의 곱셈



1.1 수학과 파이썬 복습

.4 벡터의 내적과 행렬의 곱

In [1]:
```
import numpy as np
```

In [2]:
```
a = np.array([1,2,3])
b = np.array([4,5,6])
np.dot(a,b)
```
벡터의 내적

Out[2]: 32

In [3]:
```
A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])
np.matmul(A,B)
```
행렬의 곱

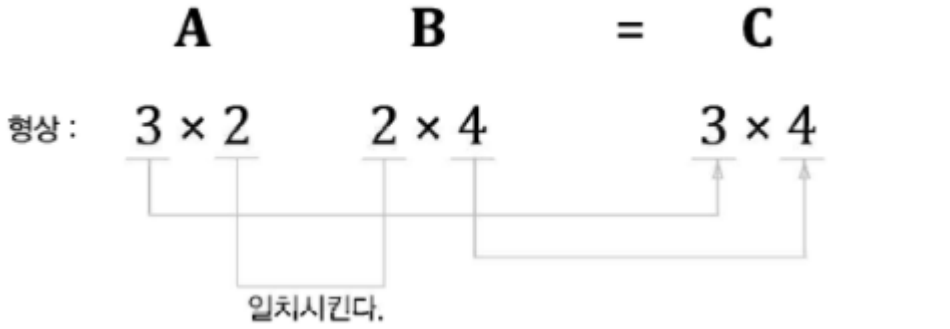
Out[3]:
```
array([[19, 22],
       [43, 50]])
```

벡터의 내적과 행렬의 곱 모두에 dot 연산을 사용 가능.
그러나 의도를 구분하기 위해...

1.1 수학과 파이썬 복습

.5 행렬 형상 확인

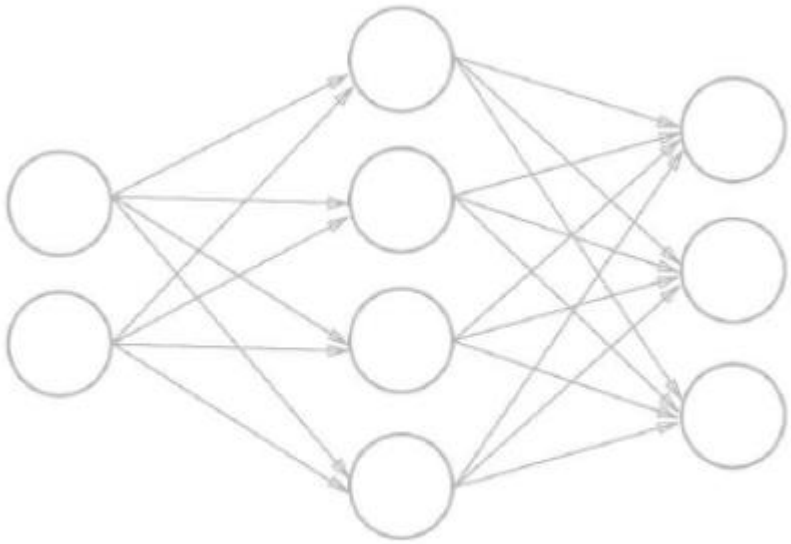
그림 1-6 형상 확인: 행렬의 곱에서는 대응하는 차원의 원소 수를 일치시킨다.



1.2 신경망의 추론

.1 신경망 추론 전체 그림

그림 1-7 신경망의 예



신경망이 계산하는 수식

$$h_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

↓ 간소화

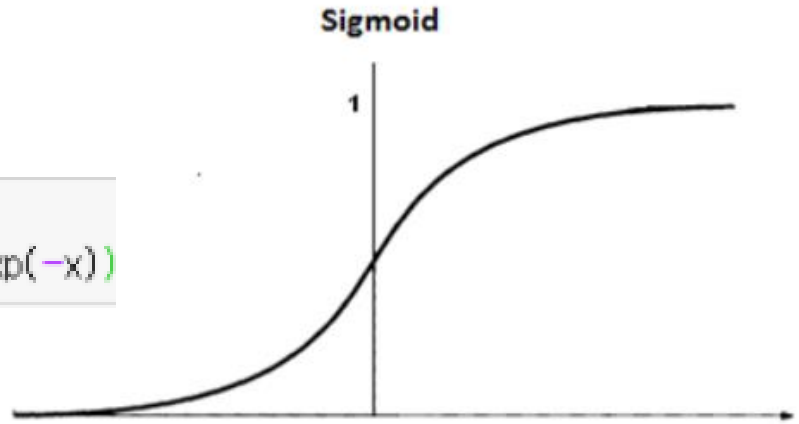
$$h = xW + b$$

```
In [1]:
1 import numpy as np

In [2]:
1 W1 = np.random.randn(2,4)
2 b1 = np.random.randn(4)
3 x = np.random.randn(10,2)
4 h = np.matmul(x,W1) + b1

In [3]:
1 h.shape

Out[3]:
(10, 4)
```

1.2 신경망의 추론

.1 신경망 추론 전체 그림

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$



코드

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

In [1]:
```
1  import numpy as np
```

In [2]:
```
1  def sigmoid(x):
2      return 1/(1+np.exp(-x))
```

In [3]:
```
1  x = np.random.randn(10,2)
2  W1 = np.random.randn(2,4)
3  b1 = np.random.randn(4)
4  W2 = np.random.randn(4,3)
5  b2 = np.random.randn(3)
```

In [4]:
```
1  h = np.matmul(x,W1) + b1
2  a = sigmoid(h)
3  s = np.matmul(a,W2) + b2
```

In [5]:
```
1  s.shape
```

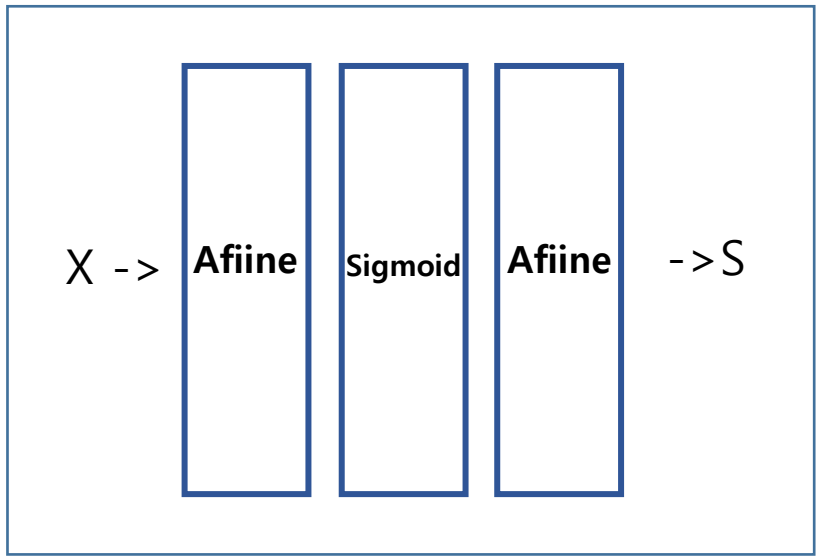
Out[5]: (10, 3)

1.2 신경망의 추론

.2 계층으로 클래스화 및 순전파 구현

```python
class Sigmoid:
    def __init__(self):
        self.params = []

    def forward(self, x):
        return 1 / (1 + np.exp(-x))
```

```python
class Affine:
    def __init__(self, W, b):
        self.params = [W, b]

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




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

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

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

        # 모든 가중치를 리스트에 모은다.
        self.params = []
        for layer in self.layers:
            self.params += layer.params

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

1.3 신경망의 학습

.1 손실 함수

: 학습 데이터(정답데이터)와 신경망이 예측한 결과를 비교하여 예측이 얼마나 나쁜가 산출한 단일 값(스칼라)

그림 1-12 손실 함수를 적용한 신경망의 계층 구성

x - 입력 데이터, t - 정답 레이블, L - 손실

Cross Entropy Error - 확률, 정답 레이블 입력

$$y_k = \frac{\exp(s_k)}{\sum_{i=1}^{n}\exp(s_i)} \quad \text{[식 1.6]}$$

softmax 함수

$$L = -\sum_k t_k \log y_k \quad \text{[식 1.7]}$$

Cross Entropy Error

$$L = -\frac{1}{N}\sum_n\sum_k t_{nk}\log y_{nk} \quad \text{[식 1.8]}$$

미니배치 처리를 고려한 Cross Entropy Error 식

1.3 신경망의 학습

.2 미분과 기울기

미분 : 조금의 변화를 극한까지 줄일 때 y 값의 변화 정도

기울기 : 각 x에서의 변화 정도

: 벡터의 각 원소에 대해 미분을 정리한 것.

1.3 신경망의 학습

.3 연쇄 법칙

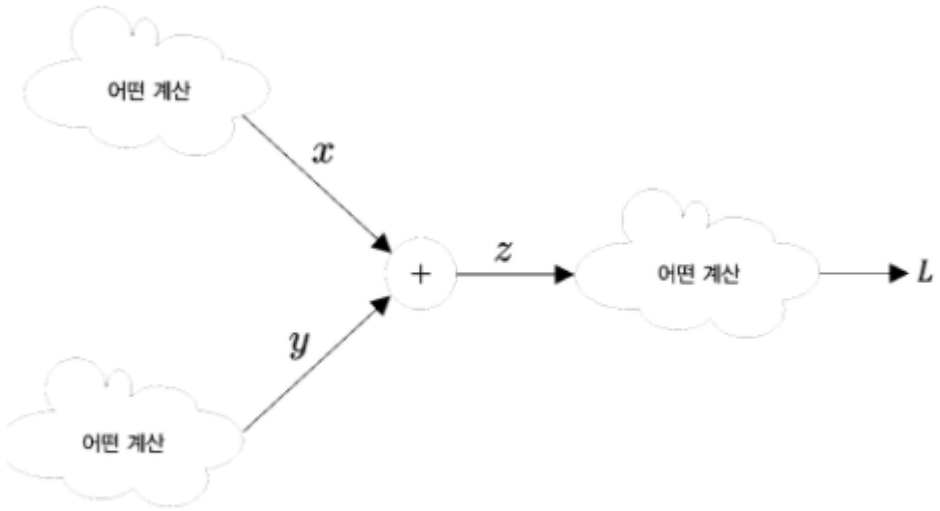
: 합성함수에 대한 미분의 법칙

ex) y=f(x), z=g(y) 일 때 z = g(f(x)) 를 합성 함수라 한다.

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

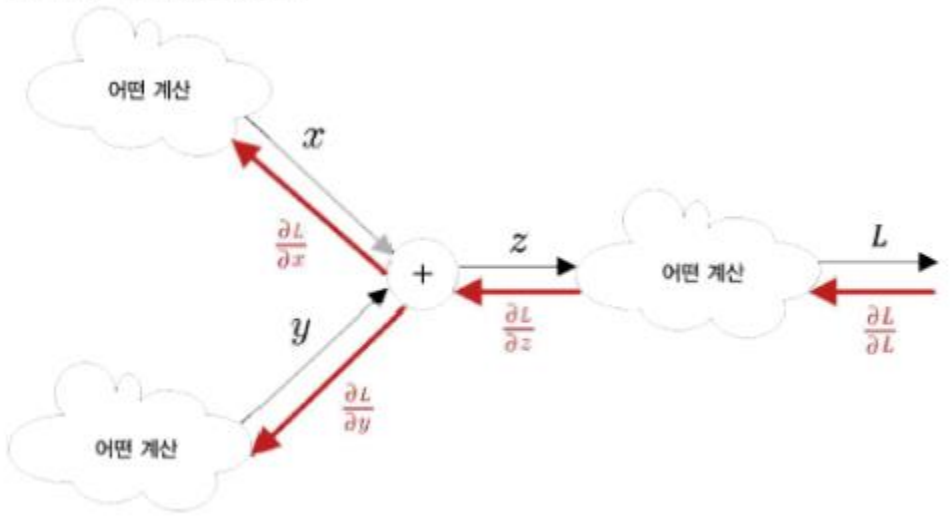
1.3 신경망의 학습

.4 계산 그래프



순전파의 계산 그래프

그림 1-17 계산 그래프의 역전파

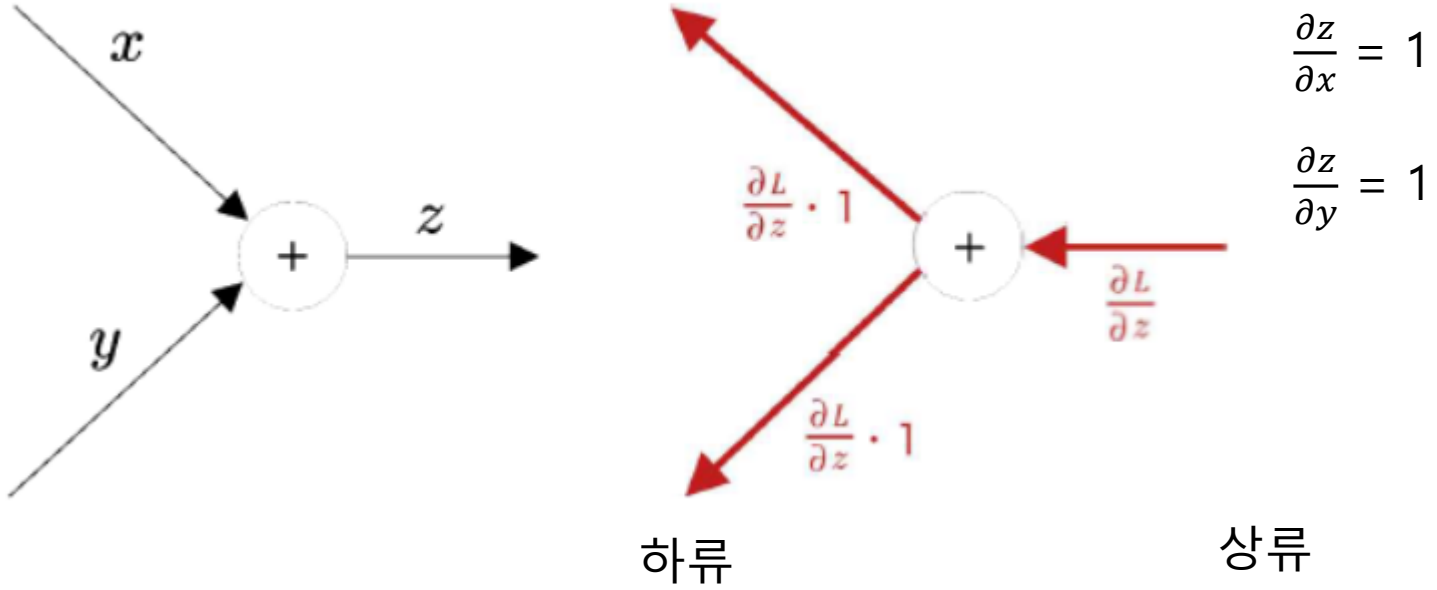


역전파가 이루어지는 과정

1.3 신경망의 학습

.4 계산 그래프- 덧셈노드

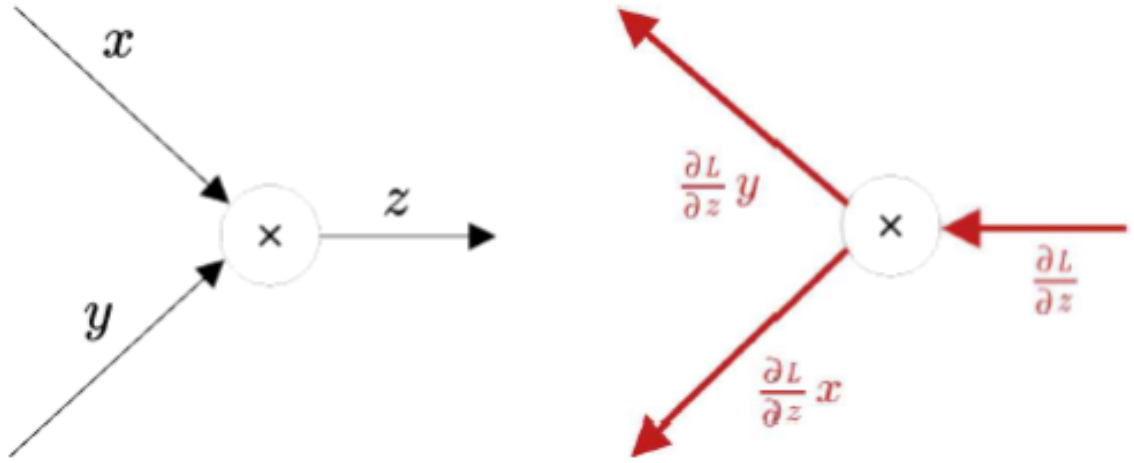
그림 1-18 덧셈 노드의 순전파(왼쪽)와 역전파(오른쪽)



1.3 신경망의 학습

.4 계산 그래프- 곱셈노드

그림 1-19 곱셈 노드의 순전파(왼쪽)와 역전파(오른쪽)



z = xy

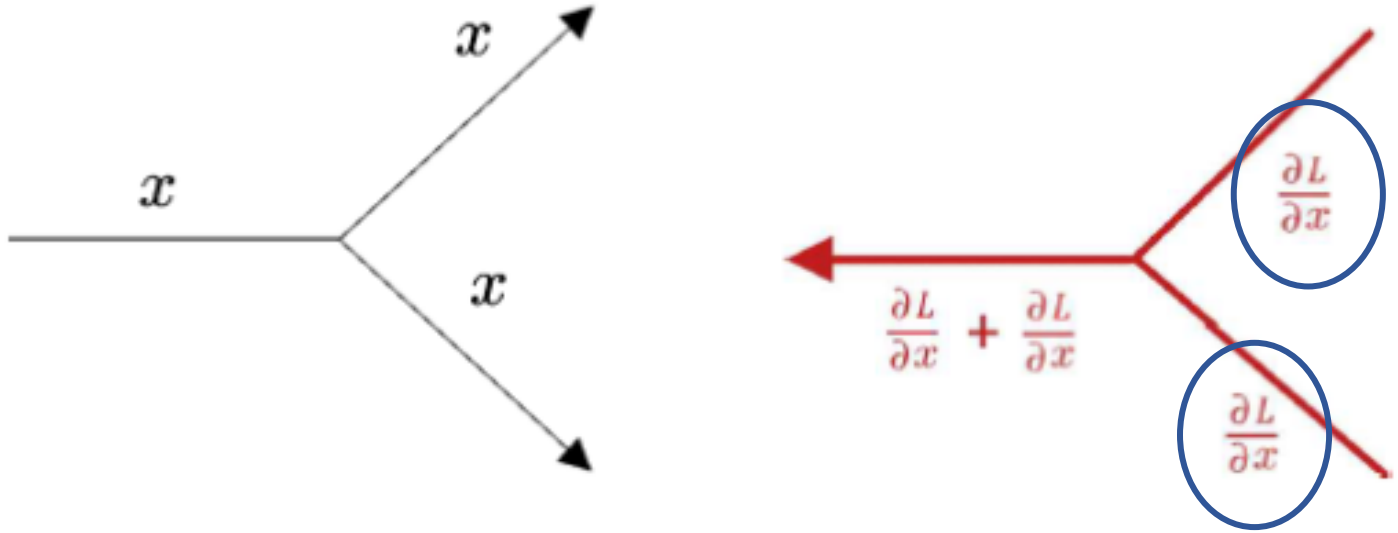
$\frac{\partial z}{\partial x}$ = y

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

1.3 신경망의 학습

.4 계산 그래프- 분기노드

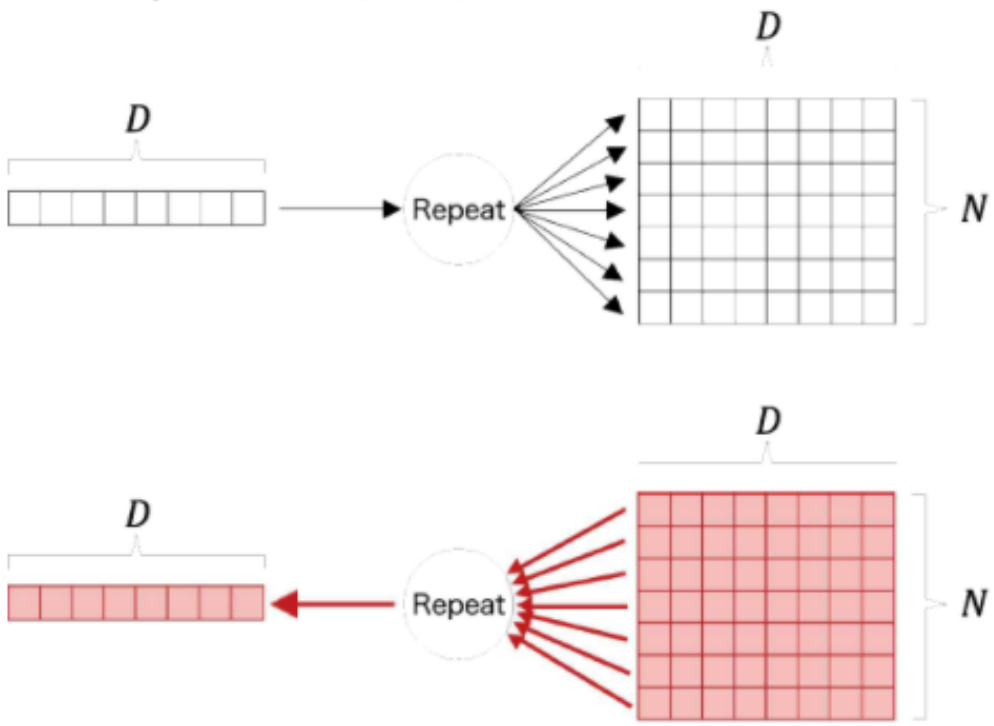
그림 1-20 분기 노드의 순전파(왼쪽)와 역전파(오른쪽)



1.3 신경망의 학습

.4 계산 그래프- Repeat노드

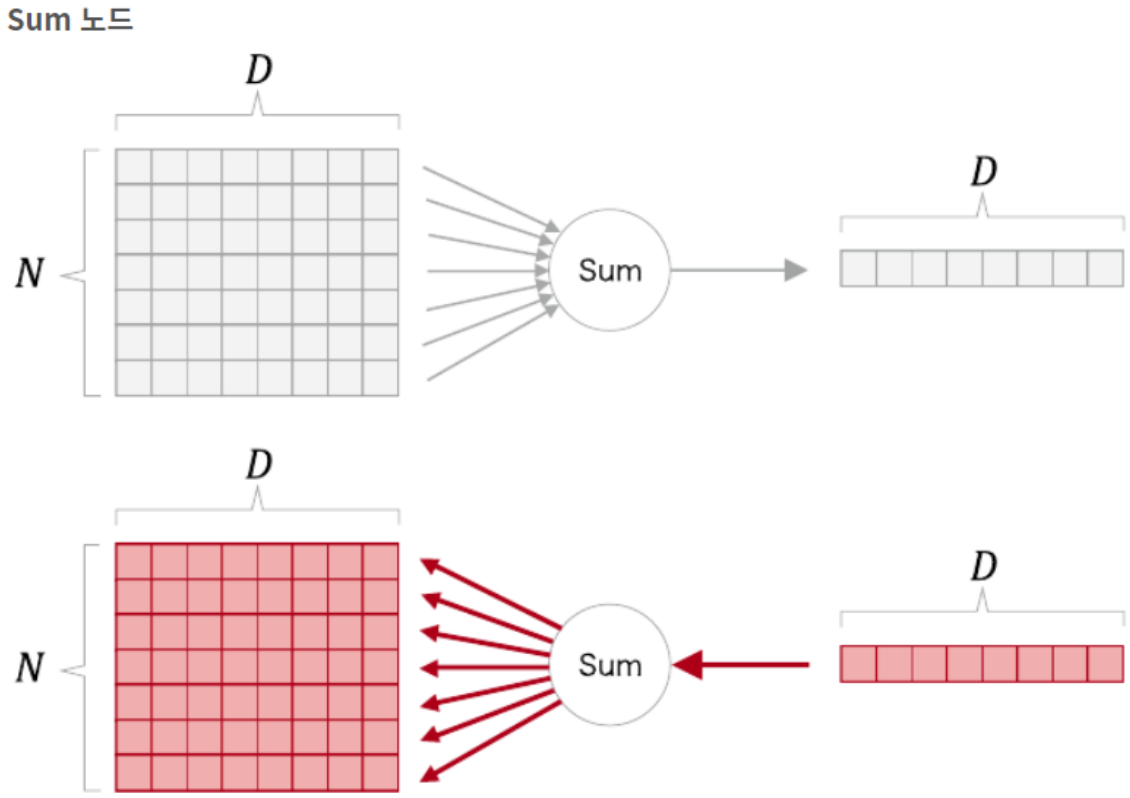
그림 1-21 Repeat 노드의 순전파(위)와 역전파(아래)




```
import numpy as np
D,N = 8,7
x = np.random.randn(1,D)
y = np.repeat(x,N,axis=0) #원소 복제, 순전파
dy = np.random.randn(N,D)
dx = np.sum(dy, axis = 0, keepdims = True)
 #keepdims 배열의 차원 유지 True - (1,N) False - (N,)
```

1.3 신경망의 학습

.4 계산 그래프- Sum노드




```
import numpy as np
D,N = 8,7
x = np.random.randn(N,D) #입력
Y = np.sum(x,axis=0 , keepdims = True) #\순전파
Dy = np.random.randn(1,D) #무작위기울기
dx = np.repeat(dy, axis = 0, keepdims = True)
 #역전파
```

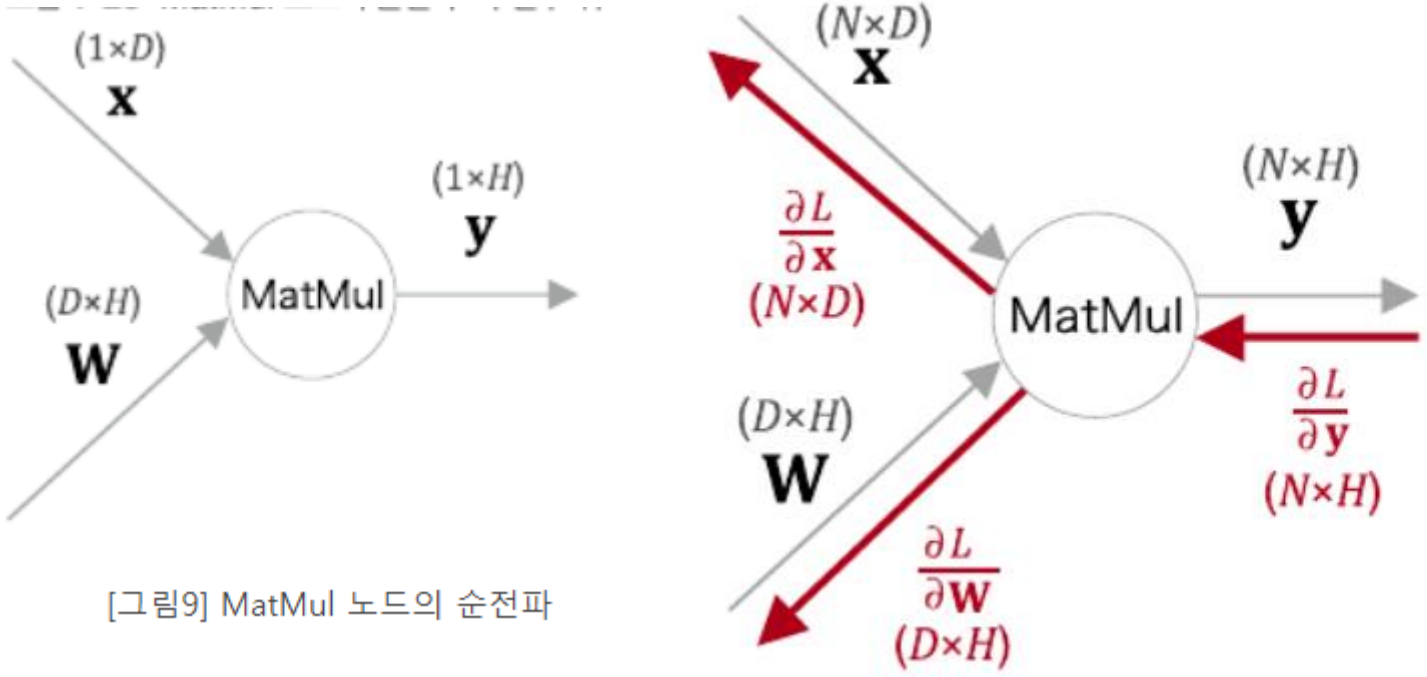
1.3 신경망의 학습

.4 계산 그래프- MatMul노드(Matrix Multiply)

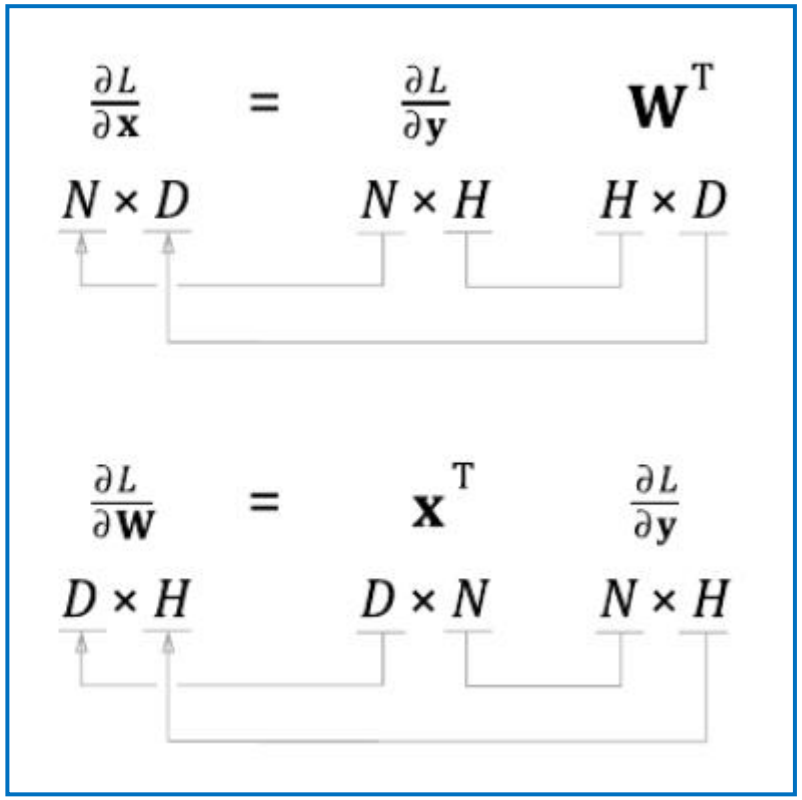
식1 $$\frac{\partial L}{\partial xi} = \sum_j \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial xi}$$

$$\frac{\partial L}{\partial x_i} = \sum_j \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial x} = \sum_j \frac{\partial L}{\partial y_j} W_{ij}$$

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



[그림9] MatMul 노드의 순전파

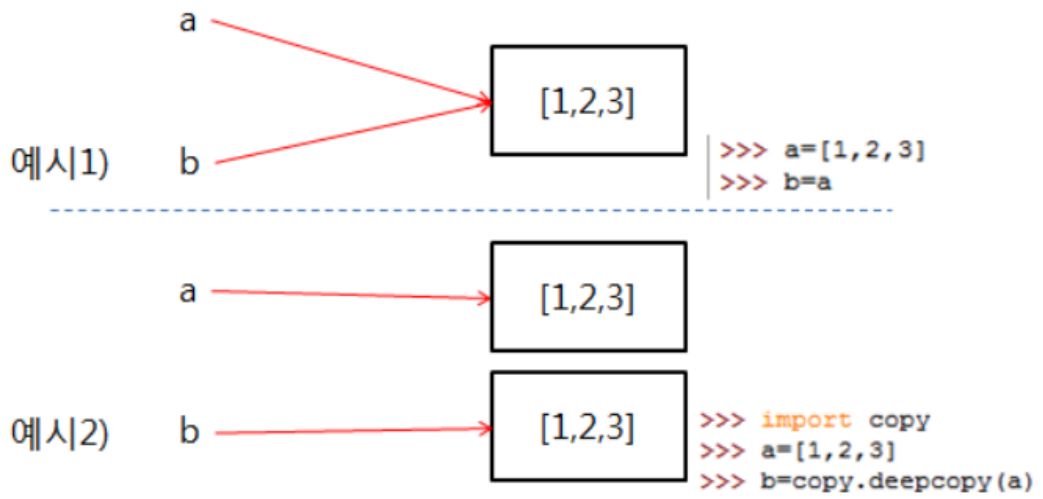


1.3 신경망의 학습

.4 계산 그래프- MatMul노드(Matrix Multiply)

```python
class MatMul:
    def __init__(self, W):
        self.params = [W]
        self.grads = [np.zeros_like(W)]
        self.x = None

    def forward(self, x):
        W, = self.params
        out = np.dot(x, W)
        self.x = x
        return out

    def backward(self, dout):
        W, = self.params
        dx = np.dot(dout, W.T)
        dW = np.dot(self.x.T, dout)
        self.grads[0][...] = dW
        return dx
```

깊은 복사



1.3 신경망의 학습

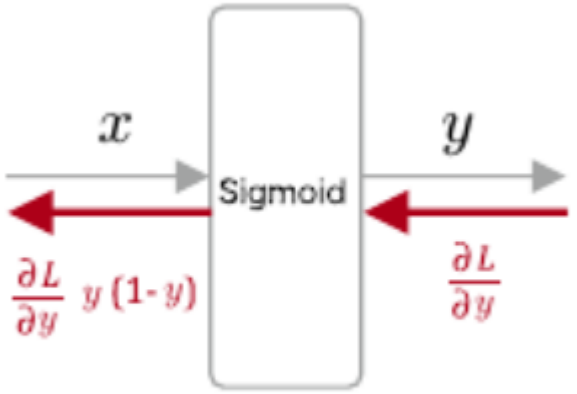
.5 기울기 도출과 역전파 구현-Sigmoid 계층

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

미분

$$\frac{\partial y}{\partial x} = y(1-y)$$

Sigmoid 계층의 계산 그래프



[그림12] Sigmoid 계층의 계산 그래프

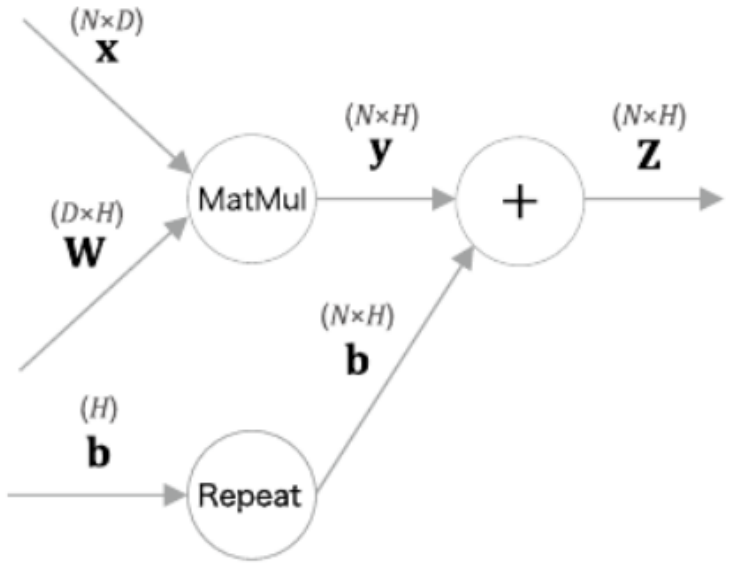
```python
class Sigmoid:
    def __init__(self):
        self.params, self.grads = [], []
        self.out = None

    def forward(self, x):
        out = 1 / (1 + np.exp(-x))
        self.out = out
        return out

    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

y = np.matmul(x, W) + b 로 구현 가능

Affine 계층 계산 그래프



[그림13] Affine 계층 계산 그래프

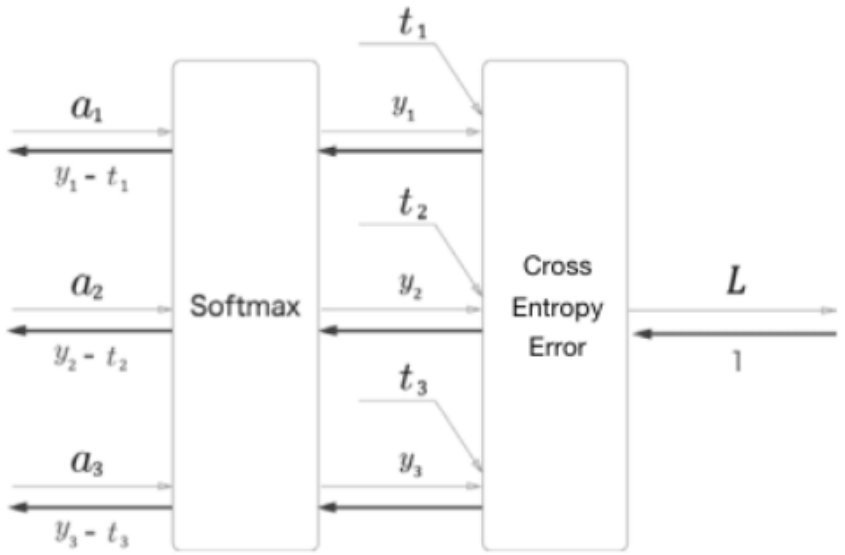
```python
class Affine:
    def __init__(self, W, b):
        self.params = [W, b]
        self.grads = [np.zeros_like(W), np.zeros_like(b)]
        self.x = None

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

    def backward(self, dout):
        W, b = self.params
        dx = np.dot(dout, W.T)
        dW = np.dot(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 + cross entropy Loss

Softmax with Loss 계층의 계산 그래프



[그림14] Softmax with Loss 계층 계산 그래프

1.3 신경망의 학습

.6 가중치 갱신

- 신경망 학습의 절차 정리

1단계 – 미니배치 : 훈련데이터 중 일부를 무작위로 가져온 데이터

2단계 – 기울기 산출 : 오차역전파법으로 각 가중치 매개변수에 대한 손실함수의 기울기를 구한다.

3단계 – 매개변수 갱신 :기울기를 사용하여 가중치 매개변수를 갱신한다.

4단계 – 반복 : 1~3단계를 반복.

- 확률적 경사 하강법 (SGD, stochastic gradient descent) : 무작위로 선택된 데이터(미니배치)에 대한 기울기를 이용한다.

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

[그림15] SGD 수식

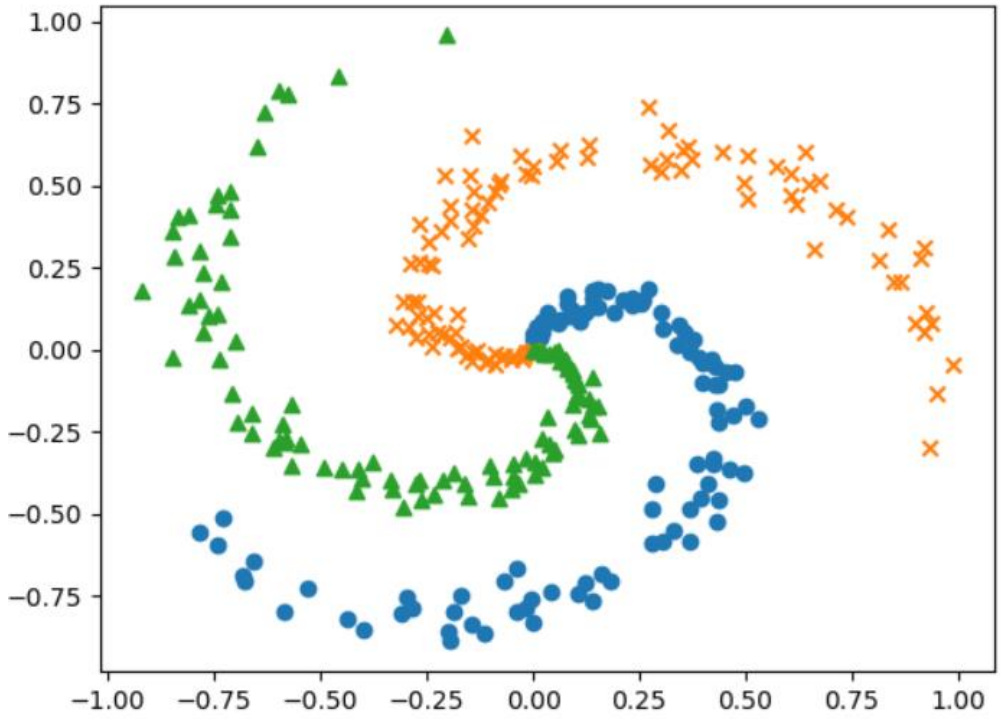
```python
class SGD:
    '''
    확률적 경사하강법(Stochastic Gradient Descent)
    '''
    def __init__(self, lr=0.01):
        self.lr = lr

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

이외 : Momentum, AdaGrad, Adam..

1.4 신경망으로 문제를 풀다

.1 스파이럴 데이터셋

```
# coding: utf-8
import sys
sys.path.append('..')  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
from dataset import spiral
import matplotlib.pyplot as plt



x, t = spiral.load_data()
print('x', x.shape)  # (300, 2)
print('t', t.shape)  # (300, 3)

# 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
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)

        # 계층 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

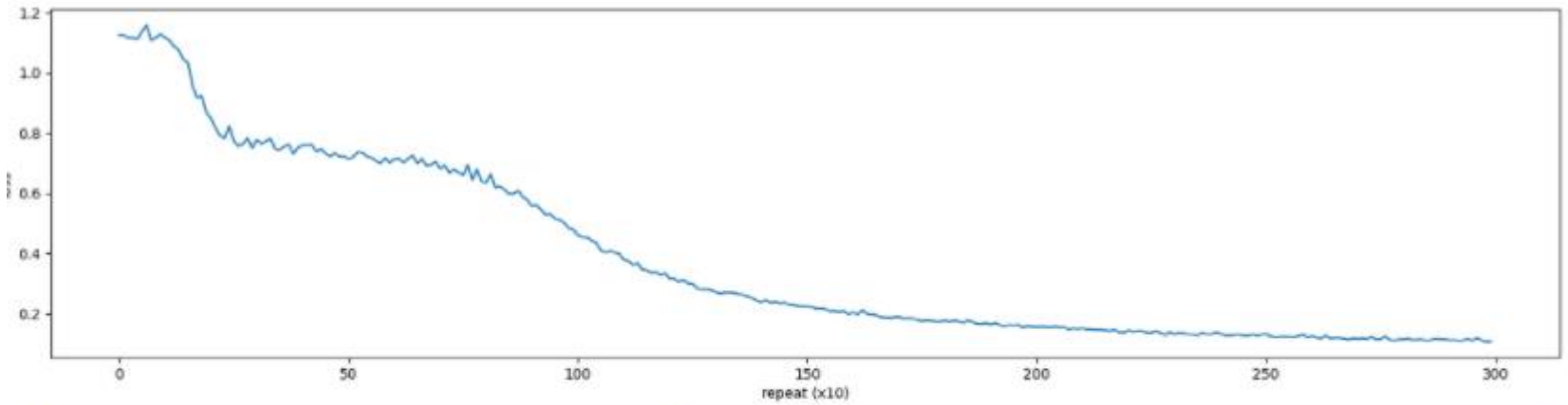
```
# coding: utf-8
import sys
sys.path.append('..')  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.optimizer import SGD
from dataset import spiral
import matplotlib.pyplot as plt
from two_layer_net import TwoLayerNet


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

# 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 = []
```

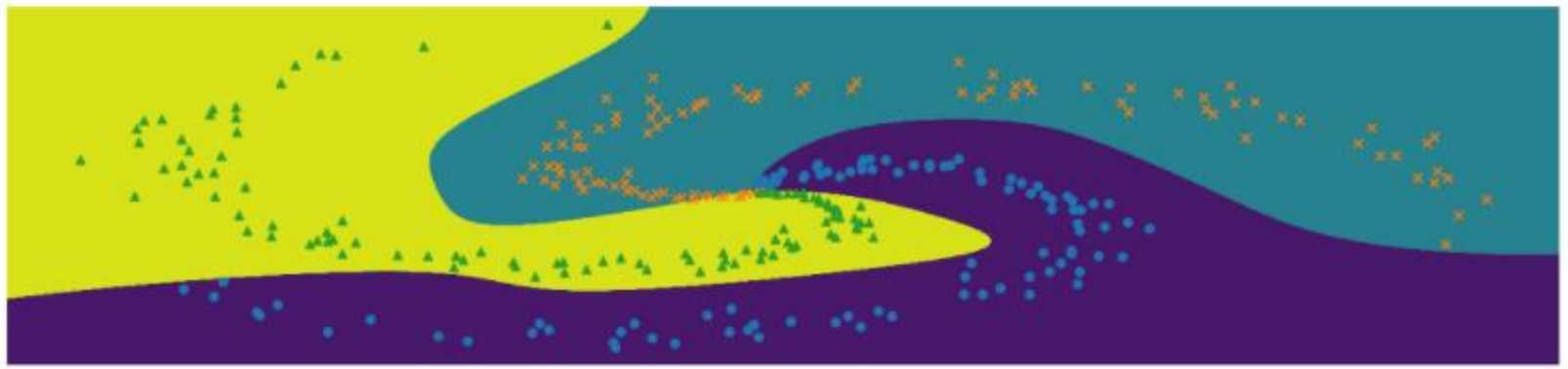
```
for epoch in range(max_epoch):
    # 데이터 뒤섞기
    idx = np.random.permutation(data_size)
    x = x[idx]
    t = t[idx]

    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('| 에폭 %d |  반복 %d / %d | 손실 %.2f'
                  % (epoch + 1, iters + 1, max_iters, avg_loss))
            loss_list.append(avg_loss)
            total_loss, loss_count = 0, 0
```

1.4 신경망으로 문제를 풀다

3. 학습용 코드 -결과

```
# 학습 결과 플롯
plt.plot(np.arange(len(loss_list)), loss_list, label='train')
plt.xlabel('반복 (x10)')
plt.ylabel('손실')
plt.show()

# 경계 영역 플롯
h = 0.001
x_min, x_max = x[:, 0].min() - .1, x[:, 0].max() + .1
y_min, y_max = x[:, 1].min() - .1, x[:, 1].max() + .1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
X = np.c_[xx.ravel(), yy.ravel()]
score = model.predict(X)
predict_cls = np.argmax(score, axis=1)
Z = predict_cls.reshape(xx.shape)
plt.contourf(xx, yy, Z)
plt.axis('off')

# 데이터점 플롯
x, t = spiral.load_data()
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

.4 Trainer클래스

```
import sys
sys.path.append('..')  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
from common.optimizer import SGD
from common.trainer import Trainer
from dataset import spiral
from two_layer_net import TwoLayerNet


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

x, t = spiral.load_data()
model = TwoLayerNet(input_size=2, hidden_size=hidden_size, output_size=3)
optimizer = SGD(lr=learning_rate)

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

1.5 계산 고속화

.1 비트 정밀도

np.dtype : 데이터 타입 확인
Ex) float64 : 64비트 부동소수점 수.

But, 신경망의 추론, 학습은 32비트로도 문제 없이 수행할 수 있다.
-> 메모리를 줄이는 관점에서 32비트가 유리
-> 버스대역폭 병목현상 감소
-> 계산속도 빨라짐

학습된 가중치를 파일에 저장할 때는 16비트여도 상관없다.
-> 학습된 가중치를 저장할 때는 16비트로 진행

1.5 계산 고속화

.2 GPU(쿠파이)

- 딥러닝은 대량의 곱하기 연산으로 구성
- 병렬 연산은 CPU < GPU
- 예제, 쿠파이라는 파이썬 라이브러리 사용할 수 있다.
- 쿠파이 : GPU를 이용해 병렬 계산을 수행해주는 라이브러리(NVIDIA의 GPU에서만 동작)
- 쿠파이의 사용법은 넘파이와 동일 (연산만 GPU이용하는 것)

1.6 정리

- 신경망은 **입력층, 은닉층(중간층), 출력층**을 지닌다.
- **완전연결계층**에 의해 **선형변환**이 이뤄지고, **활성화 함수**에 의해 **비선형 변환**이 이뤄진다.
- 완전연결계층이나 미니배치 처리는 행렬로 모아 한꺼번에 계산할 수 있다.
- **오차역전파법**을 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 ' 으로 모듈화해두면 이를 조립하여 신경망을 쉽게 구성할 수 있다.
- 신경망 고속화에는 GPU를 이용한 병렬계산과 데이터의 비트 정밀도가 중요하다.