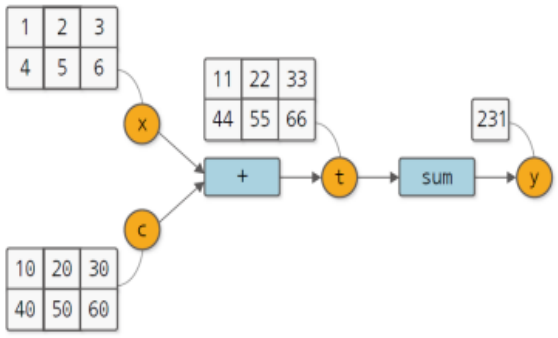
Ch.07

- 텐서(tensor)
  - 스칼라, 벡터, 행렬 같은 개념을 아우르는 상위 개념이다.
  - 다차원의 배열을 통칭한다.
  - 텐서(tensor)의 축의 개수(rank)와 각 축의 길이(dimension)가 모양을 정의한다. 특히 축이 없이 숫자 하나만 있는 것을 스칼라(Scalar), 축이 1개이고 그 축 안에 n 개의 숫자가 나열된 것을 n차원 벡터(Vector), 2개의 축에 테이블 형식으로 숫자가 나열된 것을 n * m 행렬(Matrix) 그런 행렬이 모인 것은 별도의 이름 없이 그냥 퉁 쳐서 텐서(tensor)라고 부른다.
- 텐서 사용 시의 역전파
  - Dezero 함수에 텐서를 건내면 텐서의 원소마다 스칼라로 계산한다.

그림 37-1 텐서를 사용한 계산 그래프




```
import numpy as np
from dezero import Variable
import dezero.functions as F

x = Variable(np.array([[1, 2, 3], [4, 5, 6]]))
c = Variable(np.array([[10, 20, 30], [40, 50, 60]]))
t = x + c
y = F.sum(t)

print(y)
```

```
variable(231)
```

  - 기울기(역전파)의 shape과 순전파 때의 데이터의 shape이 일치한다.

```
y.backward(retain_grad=True)
print(y.grad)
print(t.grad)
print(x.grad)
print(c.grad)
```

```
variable(1)
variable([[1 1 1]
          [1 1 1]])
variable([[1 1 1]
          [1 1 1]])
variable([[1 1 1]
          [1 1 1]])
```

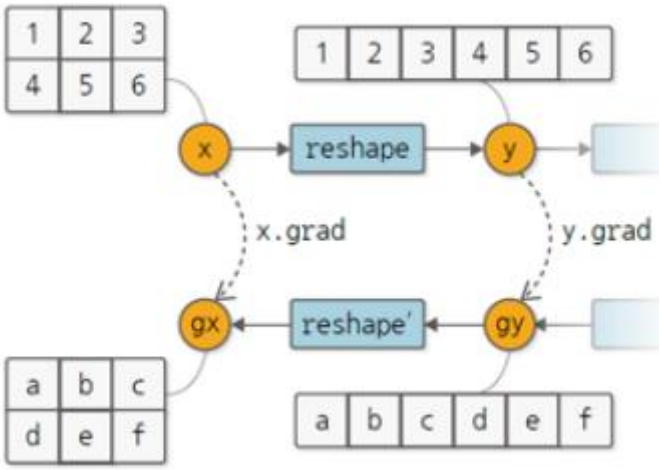
형상 변환 함수

- 원소별로 계산하지 않는 함수
  - 텐서의 형상을 변환하는 reshape 함수

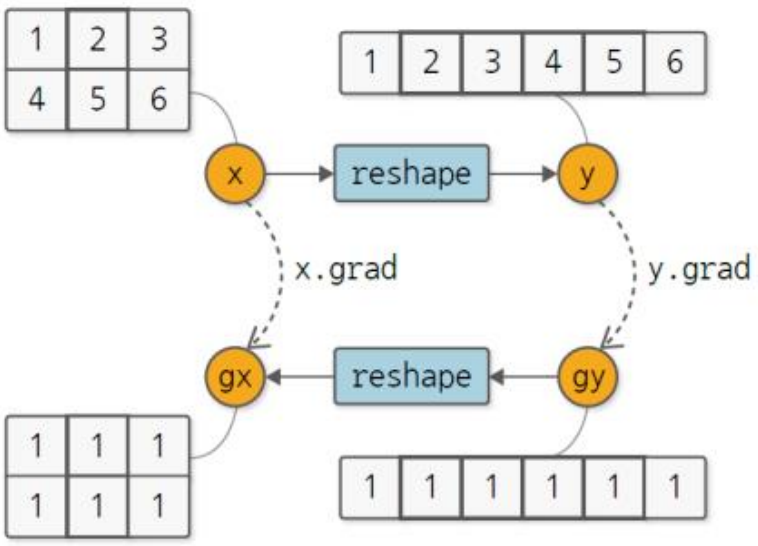
```
x = np.array([[1,2,3],[4,5,6]])
y = np.reshape(x, (6,))
print(y)
```

```
[1 2 3 4 5 6]
```

➔ np.reshape(x, shape) 형태로 쓰며 x를 shape 인수로 지정한 형상으로 변환



➔ 기울기의 형상이 입력의 형상과 같아지도록 변환



➔ y.backward(retain_grad = True)를 수행하여 x의 기울기를 구한다.

➔ y의 기울기도 자동으로 채워진다.

➔ 채워진 기울기의 형상은 y와 같다 (y.grad.shape == y.shape)

■ 행렬을 전치하는 transpose 함수

$$\mathbf{x} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix} \qquad \mathbf{x}^{\mathrm{T}} = \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ x_{13} & x_{23} \end{pmatrix}$$

```
x = np.array([[1,2,3],[4,5,6]])
y = np.transpose(x)
print(y)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

(2, 3) → (3, 2)

➔ 행렬을 전치하면 행렬의 형상이 변함

➔ 역전파는 출력 쪽에서 전해지는 기울기를 transpose 함수를 사용하여 반환

➔ 역전파에서는 순전파와는 반대의 변환이 이루어짐

```
class Variable:
    ....
    def transpose(self):
        return dezero.functions.transpose(self)

    @property
    def T(self):
        return dezero.functions.transpose(self)
```

```
x = Variable(np.random.rand(2, 3))
y = x.transpose()
y = x.T
```

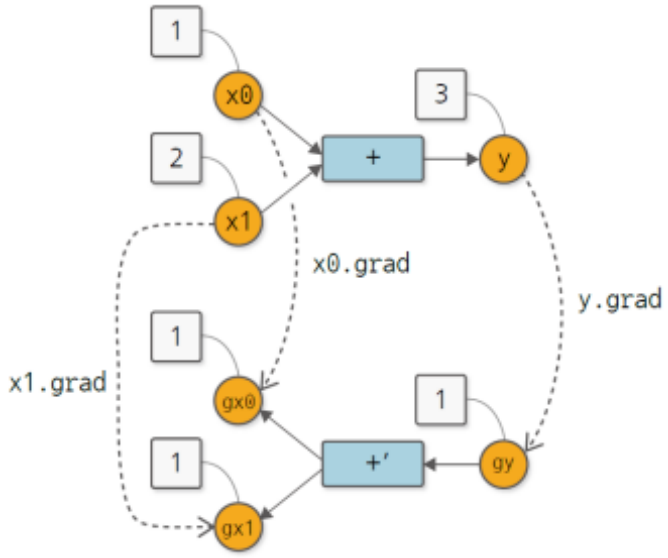
➔ 인스턴스 메서드로 이용하기 위함

➔ @property 데코레이터가 붙어 인스턴스 변수로 사용

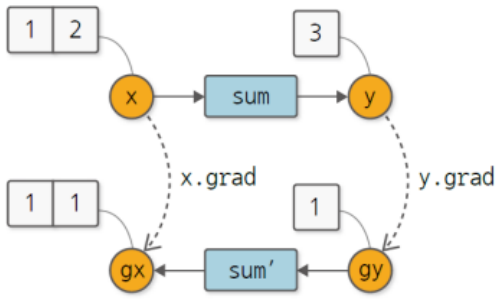
■ 두 함수 모두 텐서의 형상을 바꾸는 함수

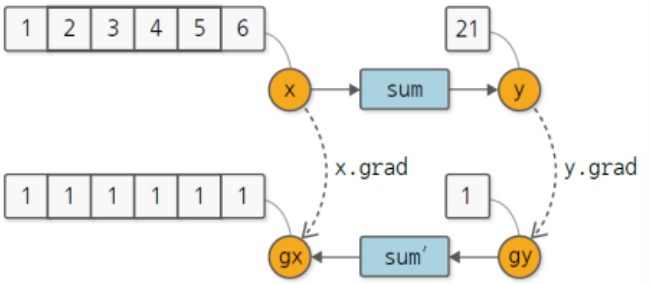
Sum 함수의 역전파

- 원소가 1개



  - 역전파는 출력 쪽에서 전해지는 기울기를 그대로 입력 쪽으로 흘려보냄
  - x0, x1에는 출력 쪽에서 전해준 1이라는 기울기를 두 개로 복사하여 전달
- 원소가 2개 이상





  - 기울기 벡터의 원소 수 만큼 복사
  - 기울기를 입력 변수의 형상과 같아지도록 복사한다.

✓ broadcast_to 함수

  - Variable 인스턴스인 x의 원소를 복사하여 shape인수로 지정한 형상이 되도록 만들어 주는 함수
  - 입력 변수와 형상이 같아지도록 기울기 gy의 원소를 복사함

```
x = Variable(np.array([1, 2, 3, 4, 5, 6]))
y = F.sum(x)
y.backward()
print(y)
print(x.grad)
```

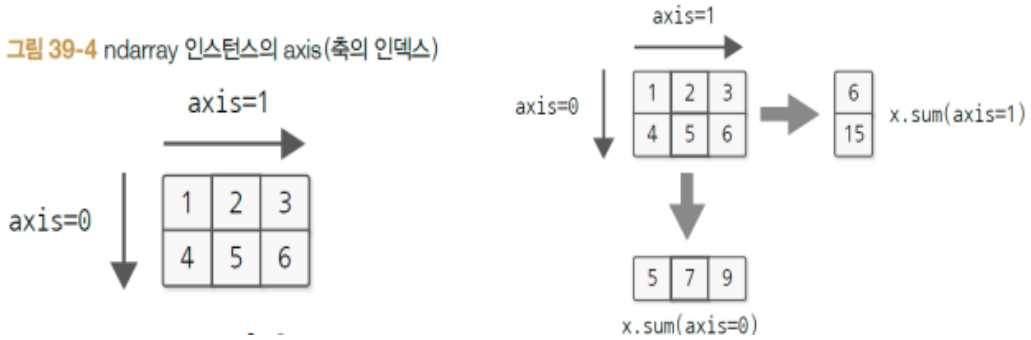
```
variable(21)
variable([1 1 1 1 1 1])
```

```
x = Variable(np.array([[1, 2, 3], [4, 5, 6]]))
y = F.sum(x)
y.backward()
print(y)
print(x.grad)
```

```
variable(21)
variable([[1 1 1]
          [1 1 1]])
```

Axis와 keepdims

- axis(축)



➔ axis는 축을 뜻하며, 다차원 배열에서 화살표의 방향을 의미

- keepdims
  - 입력과 출력의 차원 수(축 수)를 똑같게 유지할지 정하는 플래그이다.

```
x = np.array([[1,2,3],[4,5,6]])
y = np.sum(x, keepdims=True)
print(y)
print(y.shape)
```

```
[[21]]
(1, 1)
```

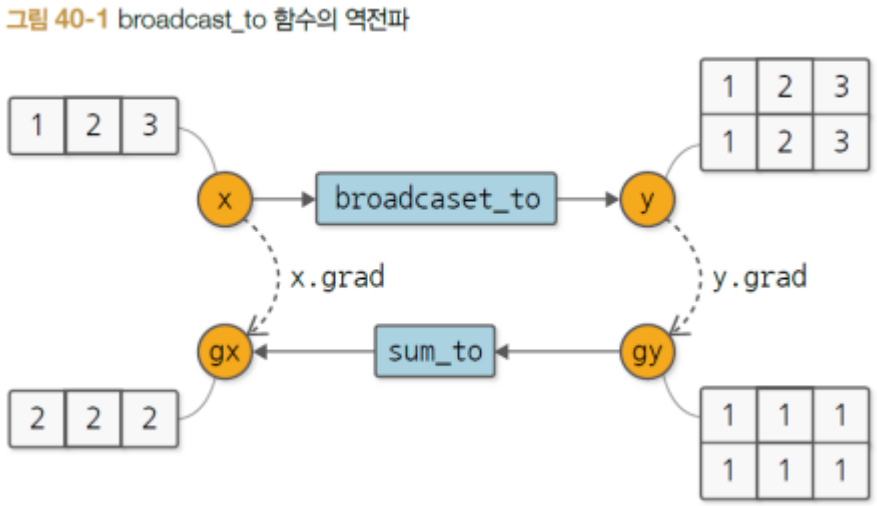
->1차원 벡터

```
x = np.array([[1,2,3],[4,5,6]])
y = np.sum(x, keepdims=False)
print(y)
print(y.shape)
```

```
21
()
```

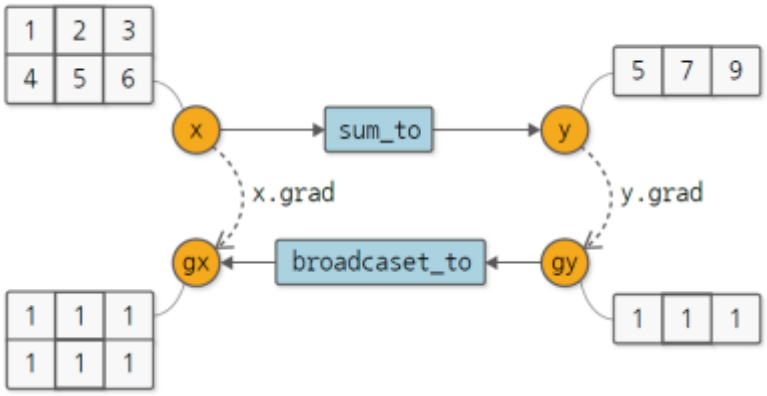
-> 스칼라

Broadcast_to 함수와 sum_to 함수



➔ sum_to 함수는 x의 원소의 합을 구해 shape 형상으로 만들어주는 함수이다.

그림 40-2 sum_to 함수의 역전파는 broadcast_to 함수



벡터의 내적과 행렬의 곱

- 벡터의 내적

$$\mathbf{ab} = a_1b_1 + a_2b_2 + \cdots + a_nb_n$$

a = (a0,…,an) 와 b = (b0,….,bn)

➔ 두 벡터 사이의 대응 원소의 곱을 모두 합한 값

- 행렬의 곱

그림 41-1 행렬의 곱 계산 방법

$$1 \times 5 + 2 \times 7$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

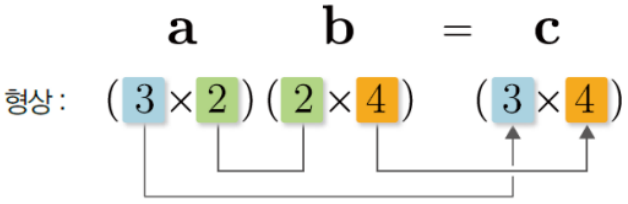
a **b** **c**

➔ 왼쪽 행렬의 가로 방향 벡터와 오른쪽 행렬의 세로 방향 벡터 사이의 내적을 계산

* 벡터의 내적과 행렬의 곱 계산은 **np.dot** 함수로 해결 가능 *

- 행렬과 벡터 사용한 계산시 주의사항

그림 41-2 행렬의 곱에서는 대응하는 차원(축)의 원소 수를 일치시킨다.



■ 행렬 a와 b의 대응하는 차원(축)의 원소 수가 일치해야 한다.

- 결과로 만들어진 행렬 c의 형상은 행렬 a와 같은 수의 행을, 행렬 b와 같은 수의 열을 갖음

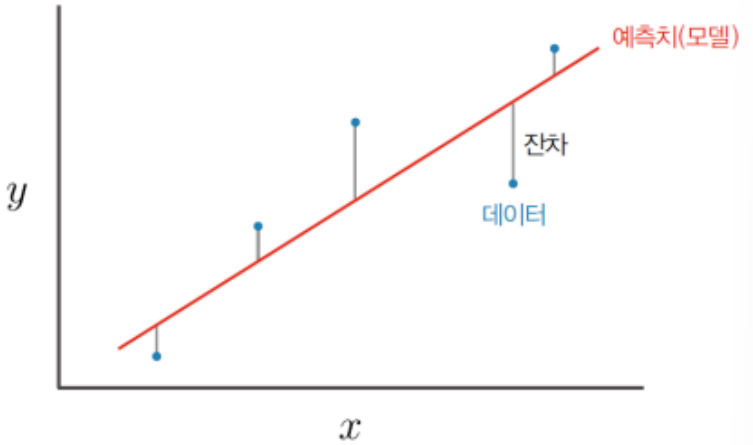
Ch.08

토이 데이터셋

- 실험용으로 만든 작은 데이터셋

선형 회귀 이론

그림 42-2 선형 회귀 예



- 예측 모델: $y = Wx + b$ → y와 x가 선형 관계라 가정
- 평균 제곱 오차(Mean Squared Error)
  - $L = \frac{1}{N}\sum_{i=0}^{N}(f(x_i) - y_i)^2$
  - 예측치(모델)와 데이터의 오차를 나타내는 지표

```
import numpy as np
import matplotlib.pyplot as plt
from dezero import Variable
import dezero.functions as F

# Generate toy dataset
np.random.seed(0)
x = np.random.rand(100, 1)
y = 5 + 2 * x + np.random.rand(100, 1)
x, y = Variable(x), Variable(y)

W = Variable(np.zeros((1, 1)))
b = Variable(np.zeros(1))
```

```
def predict(x):
    y = F.matmul(x, W) + b
    return y
```

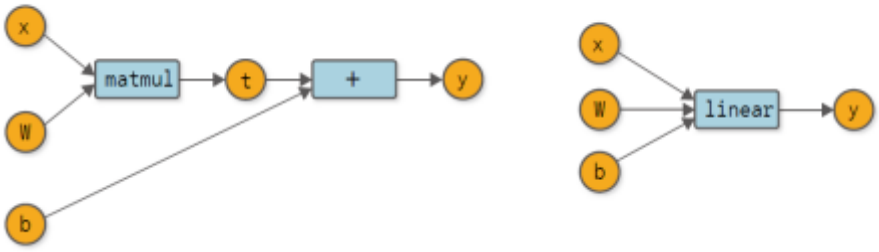
```
def mean_squared_error(x0, x1):
    diff = x0 -x1
    return F.sum(diff ** 2) / len(diff)
```

```
lr = 0.1
iters = 100

for i in range(iters):
    y_pred = predict(x)
    loss = mean_squared_error(y, y_pred)

    W.cleargrad()
    b.cleargrad()
    loss.backward()

    # Update .data attribute
    W.data -= lr * W.grad.data
    b.data -= lr * b.grad.data
    print(W, b, loss)
```

Linear 함수

- 선형 변환 혹은 아핀 변환(Affine Transformation)
  - 입력 x와 매개변수 W사이에서 행렬 곱을 구하고 거기에 b를 더함

그림 43-1 선형 변환의 두 가지 구현 방식



비선형 데이터셋

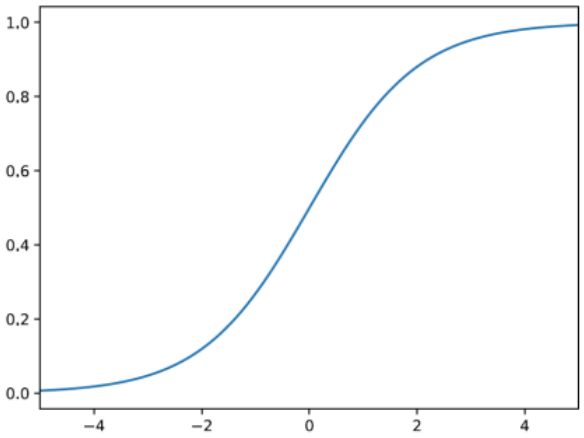
- 선형 회귀로는 문제를 풀수 없다.
- 신경망을 이용하여 해결할 수 있다.

활성화 함수

- 선형 변환은 입력 데이터를 선형으로 변환
- 신경망은 선형 변환의 출력에 비선형 변환을 수행
- 이 비선현 변환이 활성화 함수임(ReLU, Sigmoid 함수 등)

시그모이드(Sigmoid)

그림 43-3 시그모이드 함수의 그래프



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

- 활성화 함수

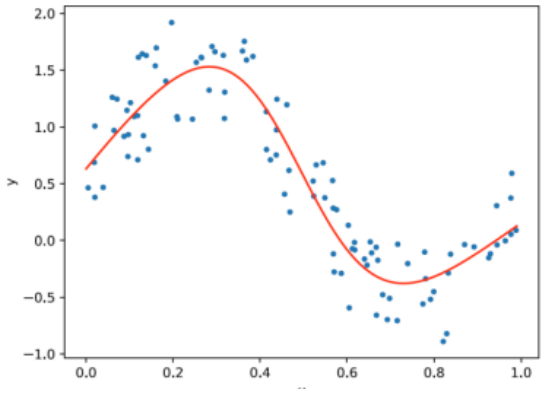
신경망 구현

- 선형 변환 -> 활성화 함수(비선형) -> 선형 변환 -> 활성화 함수 -> ....)
- 선형 변환이나 활성화 함수 등에 의한 변환을 층(Layer)라고 함

```python
import numpy as np
from dezero import Variable
import dezero.functions as F

np.random.seed(0)
x = np.random.rand(100, 1)
y = np.sin(2 * np.pi * x) + np.random.rand(100, 1

# (1) 가중치 초기화
I, H, O = 1, 10, 1
W1 = Variable(0.01 * np.random.randn(I, H))
b1 = Variable(np.zeros(H))
W2 = Variable(0.01 * np.random.randn(H, O))
b2 = Variable(np.zeros(O))

# (2) 신경망 추론
def predict(x):
    y = F.linear(x, W1, b1)
    y = F.sigmoid(y)
    y = F.linear(y, W2, b2)
    return y
```

```python
lr = 0.2
iters = 10000

# (3) 신경망 학습
for i in range(iters):
    y_pred = predict(x)
    loss = F.mean_squared_error(y, y_pred)

    W1.cleargrad()
    b1.cleargrad()
    W2.cleargrad()
    b2.cleargrad()
    loss.backward()

    W1.data -= lr * W1.grad.data
    b1.data -= lr * b1.grad.data
    W2.data -= lr * W2.grad.data
    b2.data -= lr * b2.grad.data
    if i % 1000 == 0:
        print(loss)
```

그림 43-4 학습이 완료된 신경망



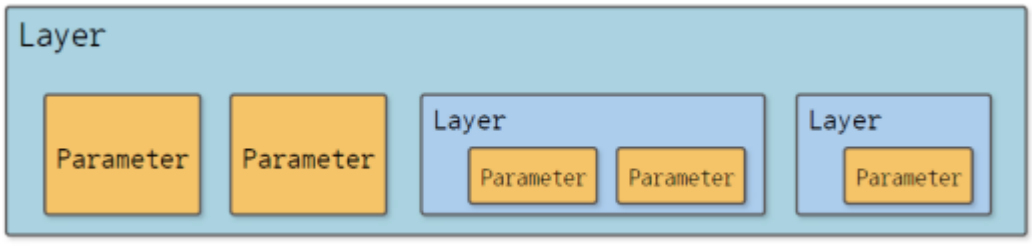
(1) 매개변수 초기화

- I는 입력층의 차원 수, H는 은닉층의 차원 수
- O는 출력층의 차원 수, 편향(bias)은 0 벡터로 초기화

(2) 신경망 추론을 수행

(3) 매개변수 갱신

Parmeter와 Layer



- 매개변수를 담는 구조
- 두 클래스를 사용하여 매개변수 관리를 자동화할 수 있다.