5장
오차역전파법

오차역전파법

가중치 매개 변수의 기울기를 효율적으로 계산하는 방법

계산그래프

계산 과정을 그래프로 나타낸 것

노드(Node)와 에지(Edge)로 표현

문제 1 : 현빈 군은 슈퍼에서 1개에 100원인 사과를 2개 샀습니다. 이때 지불 금액을 구하세요. 단, 소비세가 10% 부과됩니다.

그림 5-1 계산 그래프로 풀어본 문제 1의 답

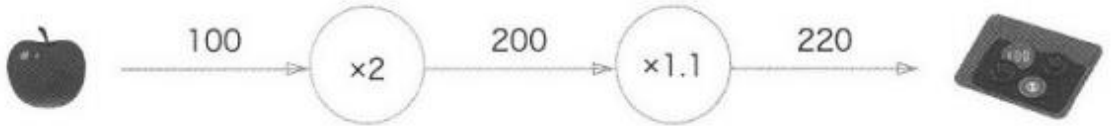
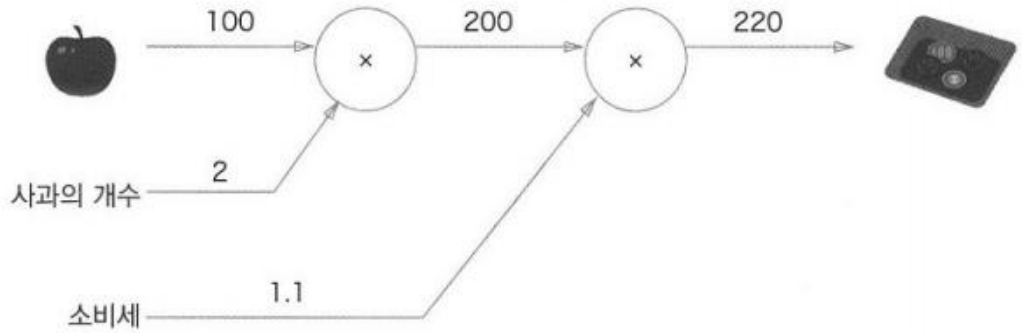
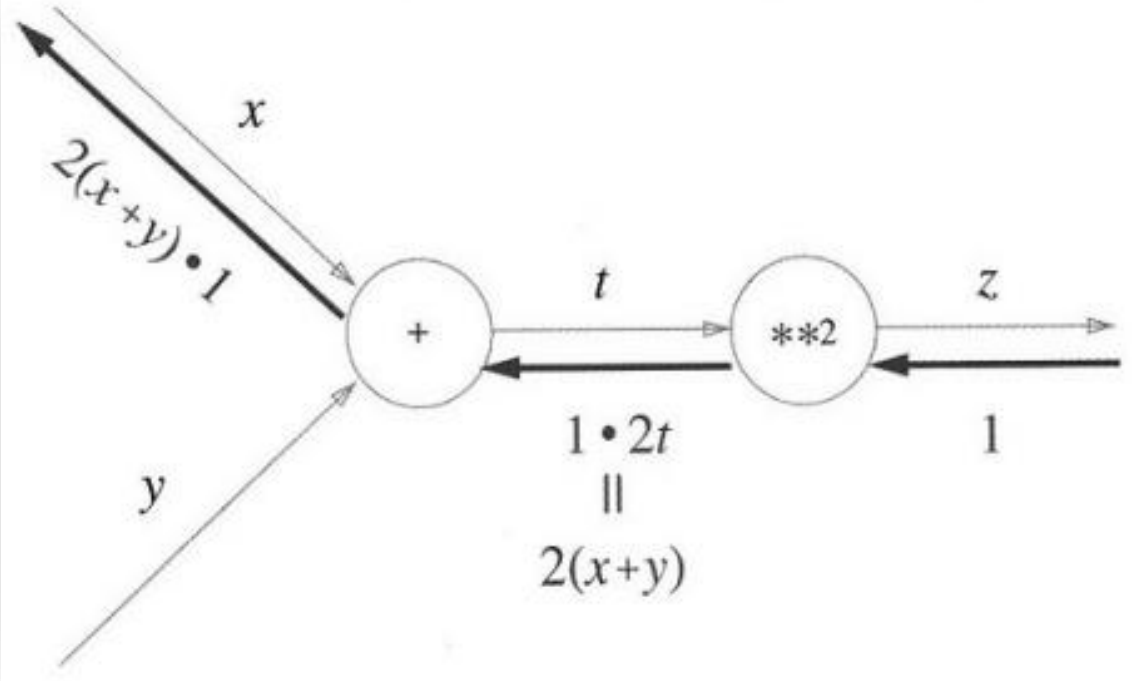


그림 5-2 계산 그래프로 풀어본 문제 1의 답 : '사과의 개수'와 '소비세'를 변수로 취급해 원 밖에 표기



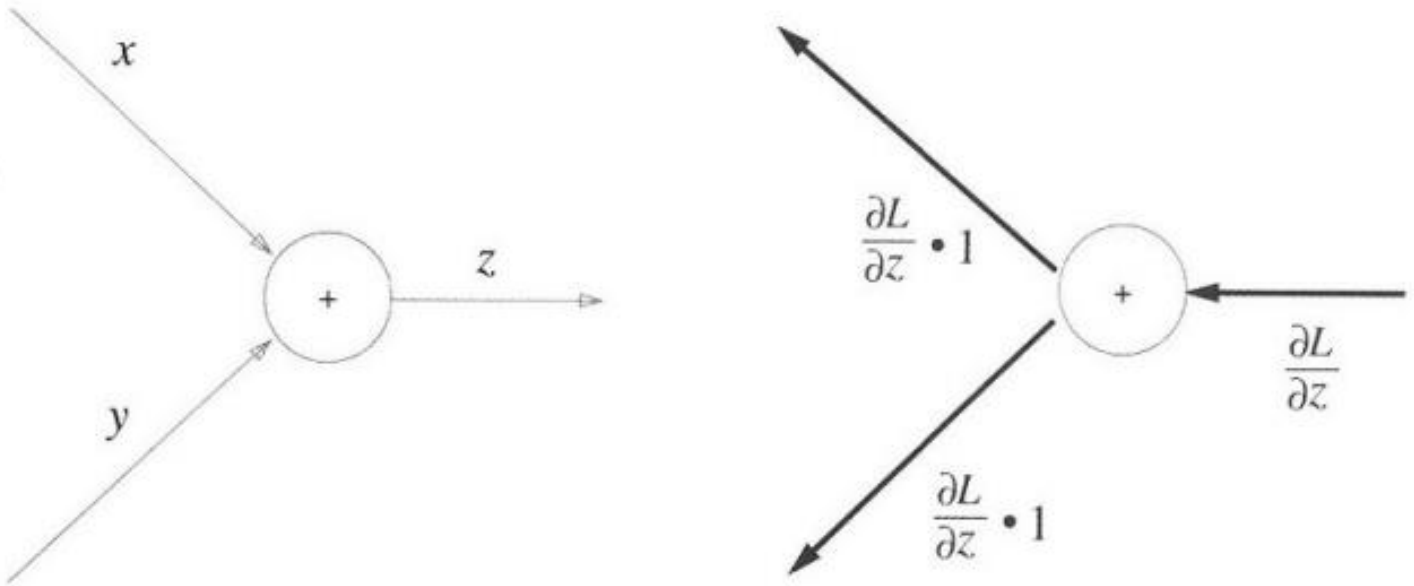
국소적 계산과 역전파

그림 5-8 계산 그래프의 역전파 결과에 따르면 $\frac{\partial z}{\partial x}$ 는 $2(x + y)$가 된다.



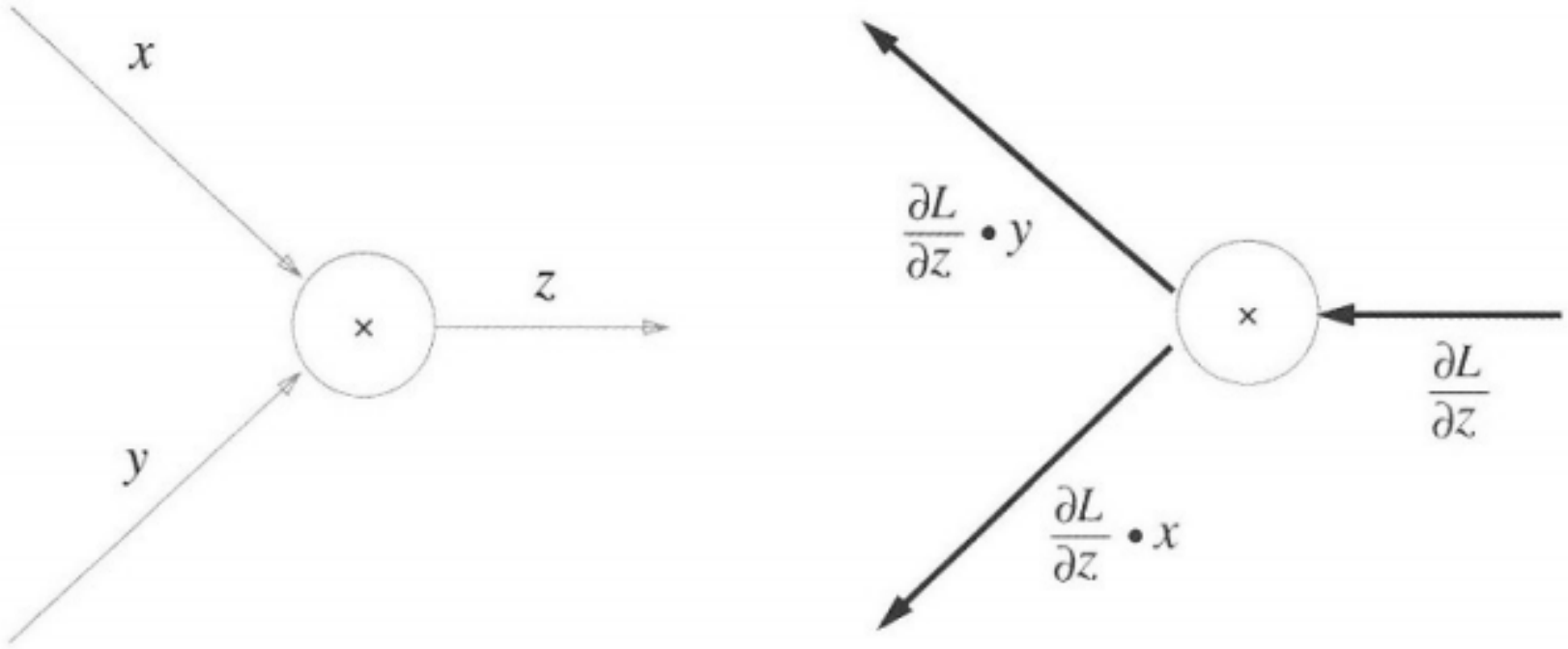
덧셈 노드의 역전파

그림 5-9 덧셈 노드의 역전파 : 왼쪽이 순전파, 오른쪽이 역전파다. 덧셈 노드의 역전파는 입력 값을 그대로 흘려보낸다.



곱셈 노드의 역전파

그림 5-12 곱셈 노드의 역전파 : 왼쪽이 순전파, 오른쪽이 역전파다.



MulLayer 구현

- __init__(): 인스턴스 변수인 x와 y 초기화
- forward()에서 인수를 받고 곱해서 반환
- backward(): 순전파 출력에 대한 미분값(dout)에 연쇄법칙 적용
- backward() 호출 순서는 forward()와 반대

```
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout):
        dx = dout * self.y # x와 y를 ㅣ
        dy = dout * self.x

        return dx, dy
```

AddLayer 구현

- __init__()은 아무 일도 하지 않음
  - 덧셈 계층에서는 초기화 불필요
  - pass가 아무것도 하지 말라는 뜻

```
class AddLayer:
    def __init__(self):
        pass

    def forward(self, x, y):
        out = x + y
        return out

    def backward(self, dout):
        dx = dout * 1
        dy = dout * 1
        return dx, dy
```

ReLU 계층 구현

```
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()
        out[self.mask] = 0

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx
```

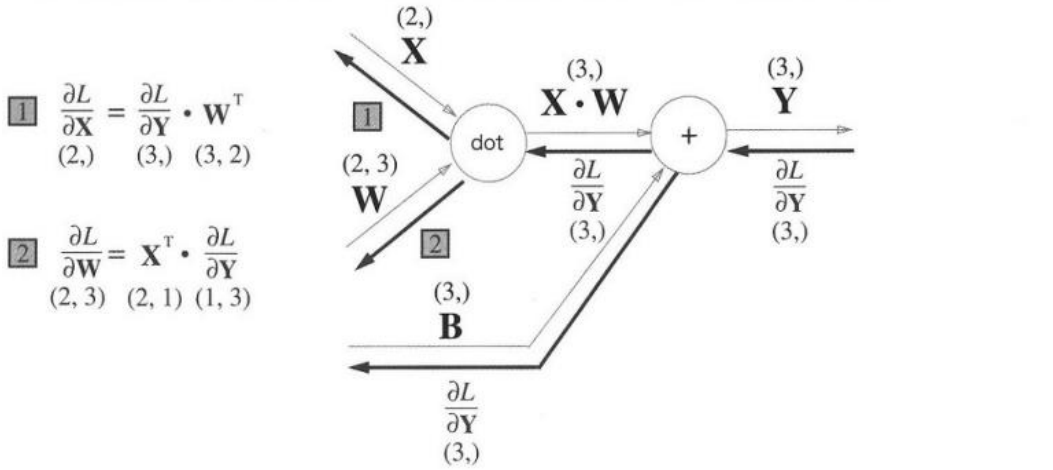
- 인스턴스 변수 mask
  - True/False로 구성된 넘파이 배열
- 순전파의 입력인 x 원소 값이 0이하면 True 그 외는 False로 유지

Sigmoid 계층 구현

```
class Sigmoid:
    def __init__(self):
        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
```

Affine 계층 구현

그림 5-25 Affine 계층의 역전파 : 변수가 다차원 배열임에 주의. 역전파에서의 변수 형상은 해당 변수명 아래에 표기했다.

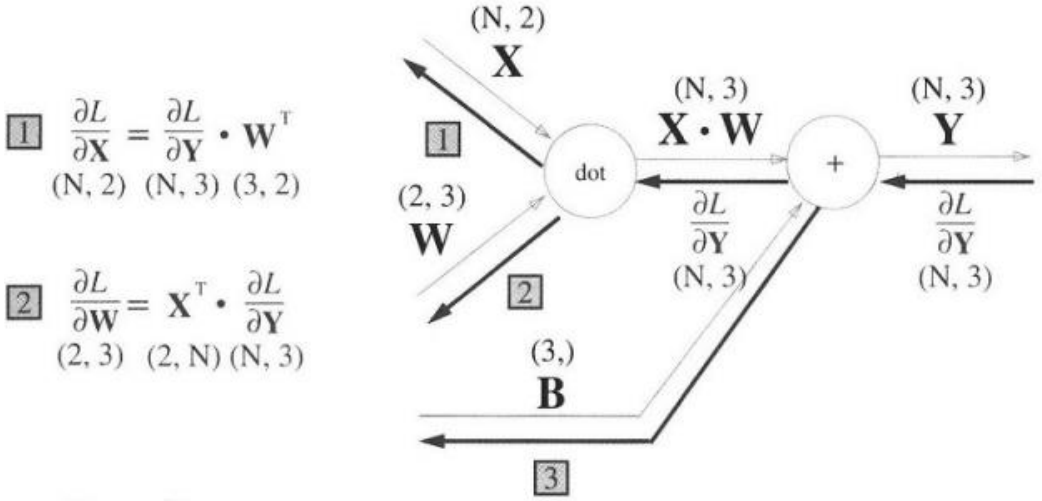


- 순전파의 흐름
  - 뉴런의 가중치 합 계산
    - Y = np.dot(X,W) + B
  - Y를 활성화 함수로 변환 이후 다음 층으로 전파

- 순전파 때 신경망에서 수행하는 행렬의 곱
  - 기하학에서 어파인 변환이라 명명

배치용 Affine 계층 구현

그림 5-27 배치용 Affine 계층의 계산 그래프



1 $\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{\mathrm{T}}$
(N, 2) (N, 3) (3, 2)

2 $\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$
(2, 3) (2, N) (N, 3)

3 $\frac{\partial L}{\partial \mathbf{B}} = \frac{\partial L}{\partial \mathbf{Y}}$ 의 첫 번째 축(0축, 열방향)의 합
(3) (N, 3)

- X 행렬의 (2,) 에서 (N, 2)로 바뀐 것 제외 앞 페이지와 동일
- 배치, 즉 여러 개의 데이터가 들어오는 경우

배치용 Affine 계층 구현

```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

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

        return out

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

        return dx
```

Softmax-with-Loss 계층

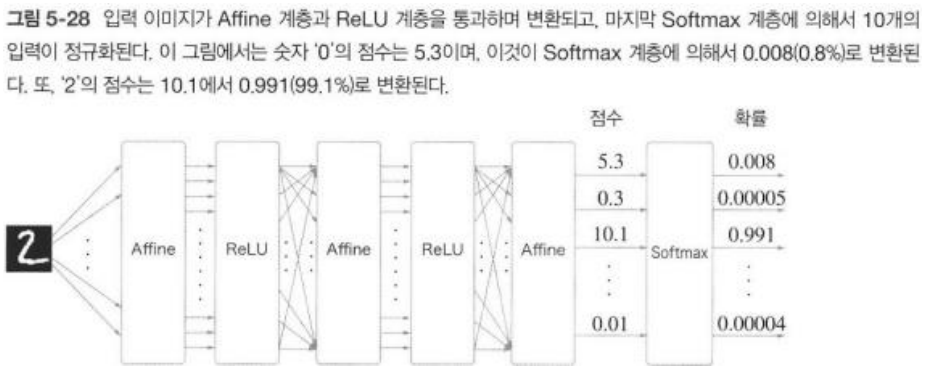


그림 5-28 입력 이미지가 Affine 계층과 ReLU 계층을 통과하며 변환되고, 마지막 Softmax 계층에 의해서 10개의 입력이 정규화된다. 이 그림에서는 숫자 '0'의 점수는 5.3이며, 이것이 Softmax 계층에 의해서 0.008(0.8%)로 변환된다. 또, '2'의 점수는 10.1에서 0.991(99.1%)로 변환된다.

- Softmax 함수가 입력 함수를 정규화
  - 모든 값의 합 = 1
- 손글씨 10가지 -> 입력값 10개

- 역전파에서는 Softmax 계층의 출력과 정답 레이블의 오차 전달
  - 오차가 클수록 학습량 증가, 작을수록 학습량 감소

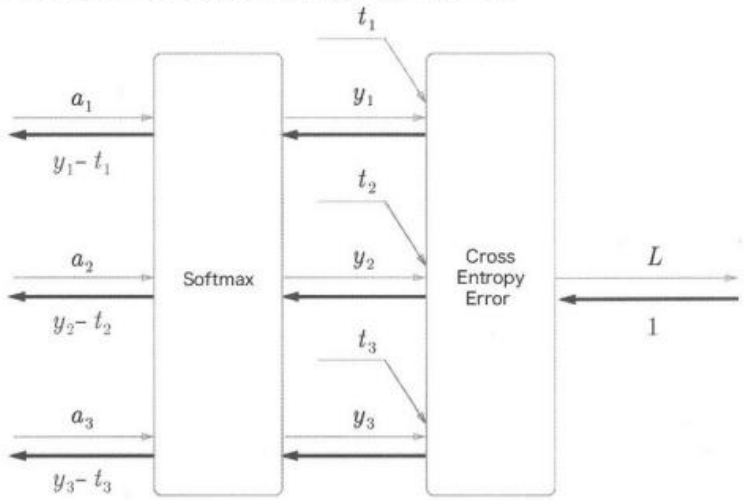


그림 5-30 '간소화한' Softmax-with-Loss 계층의 계산 그래프

Softmax-with-Loss 계층 구현

```
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None # 손실
        self.y = None    # softmax의 출력
        self.t = None    # 정답 레이블(원-핫 벡터)

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)
        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        dx = (self.y - self.t) / batch_size

        return dx
```

신경망 학습의 구조

전제

신경망에는 적응 가능한 가중치와 편향이 있고, 이 가중치와 편향을 훈련 데이터에 적응하도록 조정하는 과정을 '학습'이라 합니다. 신경망 학습은 다음과 같이 4단계로 수행합니다.

1단계 – 미니배치

훈련 데이터 중 일부를 무작위로 가져옵니다. 이렇게 선별한 데이터를 미니배치라 하며, 그 미니배치의 손실 함수 값을 줄이는 것이 목표입니다.

2단계 – 기울기 산출

미니배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기를 구합니다. 기울기는 손실 함수의 값을 가장 작게 하는 방향을 제시합니다.

3단계 – 매개변수 갱신

가중치 매개변수를 기울기 방향으로 아주 조금 갱신합니다.

4단계 – 반복

1~3단계를 반복합니다.

정리

이번 장에서 배운 내용

- 계산 그래프를 이용하면 계산 과정을 시각적으로 파악할 수 있다.
- 계산 그래프의 노드는 국소적 계산으로 구성된다. 국소적 계산을 조합해 전체 계산을 구성한다.
- 계산 그래프의 순전파는 통상의 계산을 수행한다. 한편, 계산 그래프의 역전파로는 각 노드의 미분을 구할 수 있다.
- 신경망의 구성 요소를 계층으로 구현하여 기울기를 효율적으로 계산할 수 있다(오차역전파법).
- 수치 미분과 오차역전파법의 결과를 비교하면 오차역전파법의 구현에 잘못이 없는지 확인할 수 있다(기울기 확인).