Chapter 4 신경망 학습

목표: 학습이란 훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득하는 것.

신경망이 학습할 수 있도록 해주는 지표인 손실함수.

손실 함수의 결과값을 가장 적게 만드는 가중치 매개변수를 찾는 것이 학습의 목표.

즉 손실 함수의 값을 가급적 작게 만드는 기법으로, 함수의 기울기를 활용하는 경사법 소개

4.1 데이터에서 학습한다

4.1.1 데이터 주도학습

- 기계학습은 사람의 개입을 최소화하고 수집한 데이터로부터 패턴을 찾으려 시도
- 이미지에서 특징을 추출하고 그 특징의 패턴을 기계학습 기술로 학습하는 방법 존재
  특징: 입력 데이터(입력 이미지)에서 본질적인 데이터(중요한 데이터)를 정확하게 추출할 수 있도록 설계된 변환기, 이미지의 특징은 보통 벡터로 기술하고, 컴퓨터 비전 분야에서는 SIFT, SURF, HOG 등의 특징을 많이 사용
- 이런 특징을 사용하여 이미지 데이터를 벡터로 변환하고, 변환된 벡터를 가지고 지도 학습 방식의 대표 분류 기법인 SVM, KNN 등으로 학습할 수 있음.
- 모아진 데이터로부터 규칙을 찾아내는 역할을 '기계'가 담당. 다만, 이미지를 벡터로 변환할 때 사용하는 특징은 여전히 '사람'이 설계하는 것!
- 그림과 같이 신경망은 이미지를 '있는 그대로' 학습함. 두 번째 접근 방식(특징과 기계학습 방식)에서는 특징을 사람이 설계했지만, 신경망은 포함된 중요한 특징까지도 '기계'가 스스로 학습할 것.
- 신경망의 이점은 모든 문제를 같은 맥락에서 풀 수 있다는 것

4.1.2 훈련 데이터와 시험 데이터

- 기계학습 문제는 데이터를 train set, test set으로 나눠 학습과 실험을 수행하는 것이 일반적. 우선 훈련 데이터만 사용하여 학습하면서 최적의 매개변수를 찾음. 그런 다음 시험 데이터를 사용하여 앞서 훈련한 모델의 실력을 평가하는 것.
- 범용능력을 제대로 평가하기 위해 시험 데이터를 분리함.
- 범용 능력: 아직 보지 못한 데이터(훈련 데이터에 포함되지 않는 데이터)로도 문제를 올바르게 풀어내는 능력
- 또한 이 범용능력을 획득하는 것이 기계학습의 최종 목표. 오버 피팅을 피하는 것이 기계학습의 중요한 과제

그림 4-2 규칙을 '사람'이 만드는 방식에서 '기계'가 데이터로부터 배우는 방식으로의 패러다임 전환 : 회색 블록은 사람이 개입하지 않음을 뜻한다.

4.2 손실 함수 신경망 성능의 '나쁨'을 나타내는 지표로, 현재의 신경망이 훈련데이터를 얼마나 잘 처리하지 '못 '하느냐를 나타냄

4.2.1 오차제곱합

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2$$

- 가장 많이 쓰이는 손실함수
- y_k는 신경망의 출력(신경망이 추정한 값), t_k는 정답 레이블, k는 데이터의 차원 수
- 손 글씨 숫자 인식(첫 번째 인덱스부터 숫자 0,1,2… 일 때의 값)
- y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
  소프트맥스 함수의 출력-〉 확률로 해석가능
- t = [0, 0, 1, 0 , 0, 0, 0, 0, 0, 0]
  원-핫 인코딩, 정답은 2
- 파이썬으로 구현

```
def sum_squares_error(y, t):
    return 0.5 * np.sum((y-t)**2)
```

- 오차제곱합이 더 작은 것이 정답에 더 가까울 것으로 판단

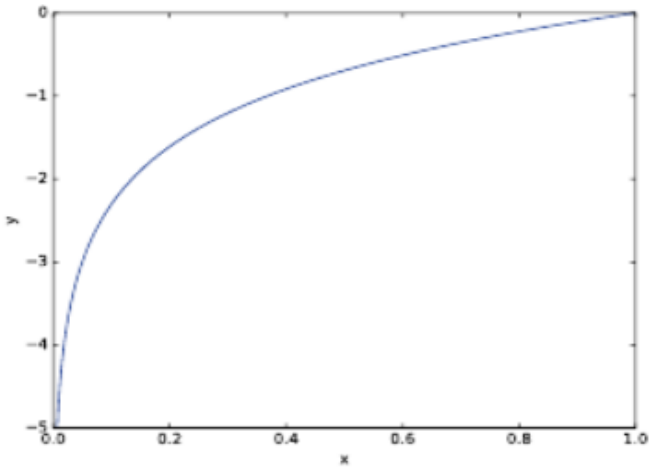
```
>>> # 정답은 '2'
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
>>>
>>> # 예1 : '2'일 확률이 가장 높다고 추정함 ( 0.6 )
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
>>> sum_squares_error(np.array(y), np.array(t))
0.09750000000000031
>>>
>>> # 예2 : '7'일 확률이 가장 높다고 추정함 ( 0.6 )
>>> y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
>>> sum_squares_error(np.array(y), np.array(t))
0.5975000000000003
```

4.2.2 교차 엔트로피 오차

$$E = -\sum_k t_k \log y_k$$

- y_k는 신경망의 출력(신경망이 추정한 값), t_k는 정답 레이블
- t_k는 원-핫 인코딩 결과. 따라서 식은 실질적으로 정답일때의 추정(t_k가 1일 때의 y_k)의 자연로그를 계산하는 식이 됨

그림 4-3 자연로그 $y = \log x$의 그래프



- x가 1일 때 y는 0이 되고 x가 0에 가까워질수록 y의 값은 점점 작아 짐.
- 식도 마찬가지로 정답에 해당하는 출력이 커질수록 0에 다가가다가, 그 출력이 1일 때 0이 됨. 반대로 정답일 때의 출력이 작아질수록 오차는 커짐.

4.2 손실 함수 신경망 성능의 '나쁨'을 나타내는 지표로, 현재의 신경망이 훈련데이터를 얼마나 잘 처리하지 '못 '하느냐를 나타냄

4.2.2 교차 엔트로피 오차

$$E = -\sum_k t_k \log y_k$$

- 파이썬으로 구현

```
def cross_entropy_error(y,t):
    delta = 1e-7
    return -np.sum(t*np.log(y+delta))
```

➢ y와 t는 넘파이 배열

➢ np.log()를 계산할 때 아주 작은 값인 delta를 더함. 이유는 np.log() 함수에 0을 입력하면 마이너스 무한대를 뜻하는 -inf가 되어 더 이상 계산을 진행할 수 없게 되기 때문. 아주 작은 값을 더해 절대 0이 되지 않도록 한 것.

- 교차 엔트로피 오차가 더 작은 것이 정답에 더 가까울 것으로 판단

```
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
>>> cross_entropy_error(np.array(y), np.array(t))
0.51082545709933802
>>>
>>> y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
>>> cross_entropy_error(np.array(y), np.array(t))
2.3025840929945458
```

- 출력이 0.6 VS 0.1

4.2.3 미니배치 학습

- 기계학습 문제는 훈련 데이터에 대한 손실함수의 값을 구하고, 그 값을 최대한 줄여주는 매개변수를 찾아냄. 이렇게 하려면 모든 훈련 데이터를 대상으로 손실 함수 값을 구해야 함.
- N개의 데이터가 존재할 때의 교차 엔트로피 오차

$$E = -\frac{1}{N}\sum_n \sum_k t_{nk} \log y_{nk}$$

- t_{nk}는 n번째 데이터의 k번째 값
- N으로 나눔으로써 '평균 손실 함수'를 구하는 것. 평균을 구해 사용하면 훈련 데이터 개수와 관계없이 언제든 통일된 지표를 얻을 수 있음.
- 그러나 빅데이터 수준이 되면 모든 데이터의 손실 함수의 합을 구하려면 시간이 걸리며 현실적이지 않음.
- 데이터 일부를 추려 전체의 '근사치'로 이용
- 신경망 학습에서도 훈련 데이터로부터 일부만 골라 학습을 수행. 이 일부를 **미니배치**라고 함
- 미니배치 학습: 가령 60000장의 훈련 데이터 중에서 100장을 무작위로 뽑아 그 100장만을 사용하여 학습하는 것

4.2 손실 함수 신경망 성능의 '나쁨'을 나타내는 지표로, 현재의 신경망이 훈련데이터를 얼마나 잘 처리하지 '못 '하느냐를 나타냄

4.2.4 (배치용) 교차 엔트로피 오차 구현하기

- 데이터가 하나인 경우와 데이터가 배치로 묶여 입력될 경우 모두를 처리할 수 있도록 미니배치 같은 배치 데이터를 지원하는 교차 엔트로피 오차 구현

```
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    batch_size = y.shape[0]
    return -np.sum(t * np.log(y + 1e-7)) / batch_size
```

➢ y: 신경망의 출력, t: 정답 레이블(1또는 0의 원-핫 인코딩)

➢ Y가 1차원이라면 (데이터 하나당 교차 엔트로피 오차를 구하는 경우) reshape함수로 데이터의 형상을 바꿔 줌
그리고 배치의 크기로 나눠 정규화 하고 이미지 1장당 평균의 교차 엔트로피 오차를 계산함

- 정답 레이블이 원-핫 인코딩이 아니라 '2'나 '7' 등의 숫자 레이블로 주어졌을 때의 교차 엔트로피 오차의 구현

```
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size
```

- 이 구현에서는 원-핫 인코딩일 때 t가 0인 원소는 교차 엔트로피 오차도 0이므로, 그 계산은 무시해도 좋다는 것이 핵심. 다시 말하면 정답에 해당하는 신경망의 출력만으로 교차 엔트로피 오차를 계산할 수 있음. 그래서 원-핫 인코딩 시 $t * np.log(y)$ 였던 부분을 레이블 표현할 때는 $np.log(y[np.arange(batch\_size), t])$ 로 구현함

➢ $np.log(y[np.arange(batch\_size), t])$:
$np.arange(batch_size)$는 0부터 $batch\_size-1$ 까지 배열을 생성함. 즉, $batch_size$가 5이면 [0,1,2,3,4]라는 넘파이 배열을 생성. t에는 레이블이 [2,7,0,9,4]와 같이 저장되어 있으므로 $y[np.arange(batch_size), t]$는 각 데이터의 정답 레이블에 해당하는 신경망의 출력을 추출함. $y[np.arange(batch_size), t]$는 $[y[0,2], y[1,7], y[2,0], y[3,9], y[4,4]]$인 넘파이 배열을 생성

4.2 손실 함수 신경망 성능의 '나쁨'을 나타내는 지표로, 현재의 신경망이 훈련데이터를 얼마나 잘 처리하지 '못 '하느냐를 나타냄

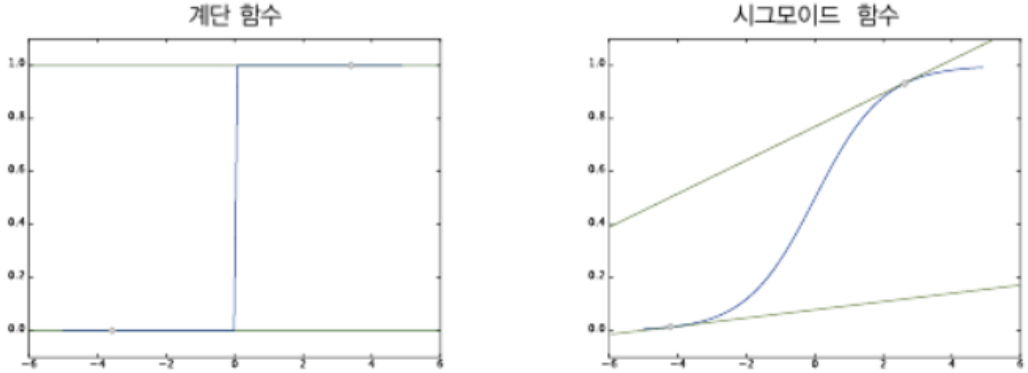
4.2.5 왜 손실 함수를 설정하는가?

- 궁극적인 목적은 높은 정확도를 끌어내는 매개변수 값을 찾는 것. 그렇다면 '정확도'라는 지표를 놔두고 '손실 함수의 값'이라는 우회적인 방법을 택하는 이유는 무엇일까?
- 이 의문은 신경망 학습에서의 '미분'의 역할에 주목한다면 해결됨. 신경망 학습에서는 최적의 매개변수(가중치와 편향)를 탐색할 때 손실함수의 값을 가능한 작게 하는 매개변수 값을 찾음. 이때 매개변수의 미분(기울기)을 계산하고, 그 미분 값을 단서로 매개변수의 값을 서서히 갱신하는 과정을 반복함.
- 가령 가상의 신경망이 있고 그 신경망의 어느 한 가중치 매개변수에 주목한다고 할 때, 그 가중치 매개변수의 손실함수의 미분이란 '가중치 매개변수의 값을 아주 조금 변화시켰을 때, 손실 함수가 어떻게 변하나' 라는 의미.
  만약 이 미분 값이 음수면 그 가중치 매개변수를 양의 방향으로 변화시켜 손실 함수의 값을 줄임.
  반대로 미분 값이 양수면 가중치 매개변수를 음의 방향으로 변화시켜 손실 함수의 값을 줄임.
  그러나 미분 값이 0이면 가중치 매개변수를 어느 쪽으로 움직여도 손실 함수의 값은 줄어들지 않음. 그래서 가중치 매개변수의 갱신은 거기서 멈추게 됨.
- 신경망 학습에 정확도를 지표로 삼아서는 안되는 이유는 미분 값이 대부분의 장소에서 0이 되어 매개변수를 갱신할 수 없기 때문임.

→ *신경망을 학습할 때 정확도를 지표로 삼아서는 안 된다. 정확도를 지표로 하면 매개변수의 미분이 대부분의 장소에서 0이 되기 때문이다.*

- 정확도를 지표로 삼으면 매개변수의 미분이 대부분의 장소에서 0이 되는 이유는 무엇일까?
- 구체적인 예) 한 신경망이 100장의 훈련 데이터 중 32장을 올바로 인식한다고 한다면 정확도는 32%가 됨. 만약 정확도가 지표였다면 가중치 매개변수의 값을 조금 바꾼다고 해도 정확도는 그대로 32%일 것임. 혹, 정확도가 개선된다 하더라도 33%나 34%처럼 불연속적인 띄엄띄엄한 값으로 바뀌어 버림
- 손실함수를 지표로 삼았을 때, 현재의 손실 함수의 값은 0.92543… 같은 수치로 나타냄. 그리고 매개변수의 값이 조금 변하면 그에 반응하여 손실함수의 값도 0.93432… 처럼 연속적으로 변화.
- 활성화 함수로 계단 함수를 사용하면 신경망 학습이 잘 이뤄지지 않음. 대부분의 장소(0이외의 곳)에서 미분이 0임

그림 4-4 계단 함수와 시그모이드 함수 : 계단 함수는 대부분의 장소에서 기울기가 0이지만, 시그모이드 함수의 기울기(접선)는 0이 아니다.



4.3 수치 미분 경사법에서는 기울기(경사) 값을 기준으로 나아갈 방향을 정함

4.3.1 미분

- 한 순간의 변화량 $\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$
- 파이썬으로 구현

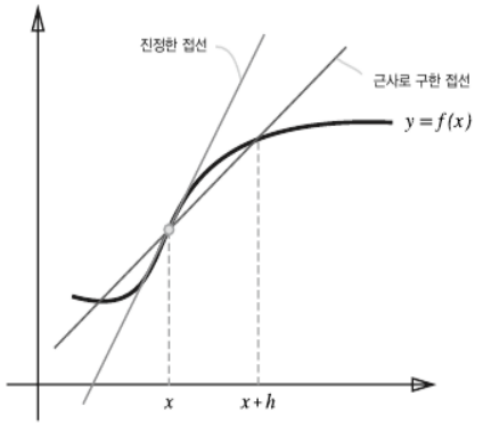
```
#나쁜 구현 예
def numerical_diff(f, x):
    h = 10e-50
    return (f(x + h)-f(x))/h
```

- 개선점1) h에 가급적 작은 값을 대입하고 싶어 10e-50라는 작은 값 이용. 이 값은 0.00…1 형태에서 소수점 아래 0이 49개라는 의미
  -> 이 방식은 반올림 오차 문제를 일으킴. 반올림 오차는 작은 값이 생략되어 최종 계산 결과에 오차가 생기게 함

```
>>> np.float32(1e-50)
0.0
```

- 개선점2) 전방 차분 -> 중심 차분 혹은 중앙 차분

그림 4-5 진정한 미분(진정한 접선)과 수치 미분(근사로 구한 접선)의 값은 다르다.

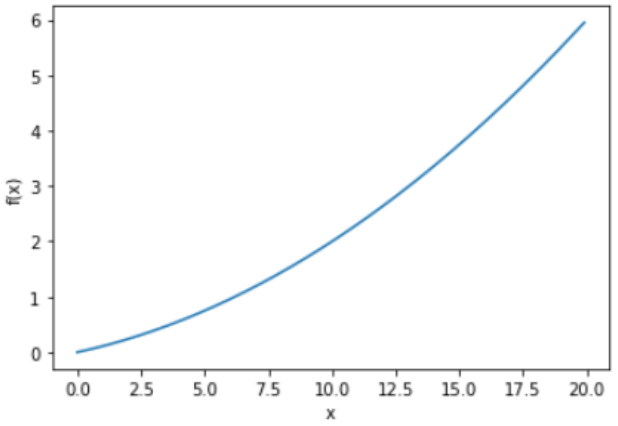



```
def numerical_diff(f,x):
    h = 1e-4 #0.0001
    return (f(x+h) -f(x-h)) / (2*h)
```

4.3.2 수치 미분의 예

- 간단한 함수 미분 $y = 0.01x^2 + 0.1x$

```
#4.3.2
def function_1(x):
    return 0.01*x**2 + 0.1*x
```




```
numerical_diff(function_1, 5)
```

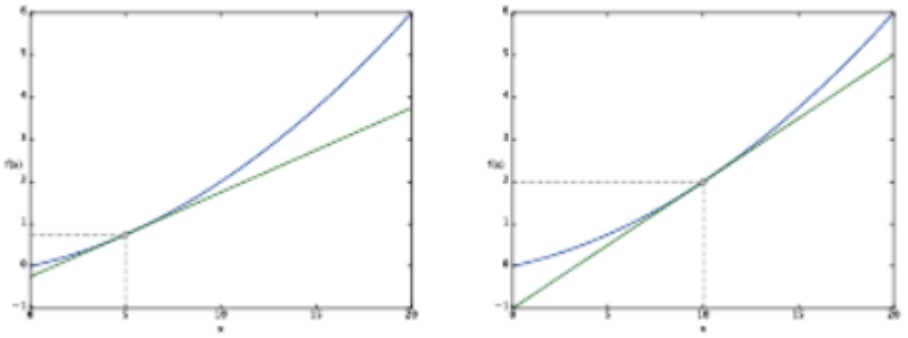
```
0.1999999999990898
```

```
numerical_diff(function_1, 10)
```

```
0.2999999999986347
```

- 5일 때와 10일 때의 미분 값 진정한 미분 값은 2.0, 3.0

그림 4-7 $x = 5, x = 10$에서의 접선 : 직선의 기울기는 수치 미분에서 구한 값을 사용하였다.



4.3 수치 미분 경사법에서는 기울기(경사) 값을 기준으로 나아갈 방향을 정함

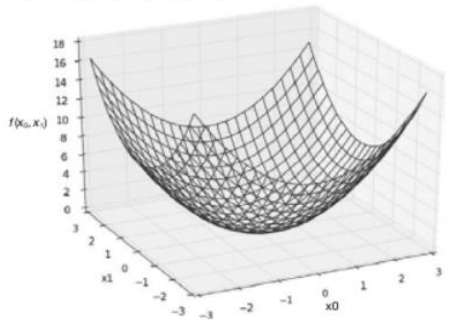
4.3.3 편미분 $f(x_0, x_1) = x_0^2 + x_1^2$

- 변수가 여럿인 함수에 대한 미분 $\frac{\partial f}{\partial x_0}$ 나 $\frac{\partial f}{\partial x_1}$
- 파이썬으로 구현

```
def function_2(x):
    return x[0]**2 + x[1]**2
    # 또는 return np.sum(x**2)
```

- 인수 x는 넘파이 배열

그림 4-8 $f(x_0, x_1) = x_0^2 + x_1^2$의 그래프



문제 1 : $x_0 = 3, x_1 = 4$일 때, x_0에 대한 편미분 $\frac{\partial f}{\partial x_0}$ 를 구하라.

```
>>> def function_tmp1(x0):
...     return x0*x0 + 4.0**2.0
...
>>> numerical_diff(function_tmp1, 3.0)
6.00000000000378
```

문제 2 : $x_0 = 3, x_1 = 4$일 때, x_1에 대한 편미분 $\frac{\partial f}{\partial x_1}$ 를 구하라.

```
>>> def function_tmp2(x1):
...     return 3.0**2.0 + x1*x1
...
>>> numerical_diff(function_tmp2, 4.0)
7.999999999999119
```

- 이 문제들은 변수가 하나인 함수를 정의하고, 그 함수를 미분하는 형태로 구현하여 풀었음

4.4 기울기 모든 변수의 편미분을 벡터로 정리한 것

- 양쪽의 편미분을 묶어서 계산 $\left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}\right)$ $f(x_0, x_1) = x_0^2 + x_1^2$
- 파이썬으로 구현

```
def _numerical_gradient_no_batch(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x) # x와 형상이 같은 배열을 생성

    for idx in range(x.size):
        tmp_val = x[idx]

        # f(x+h) 계산
        x[idx] = float(tmp_val) + h
        fxh1 = f(x)

        # f(x-h) 계산
        x[idx] = tmp_val - h
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val # 값 복원

    return grad
```

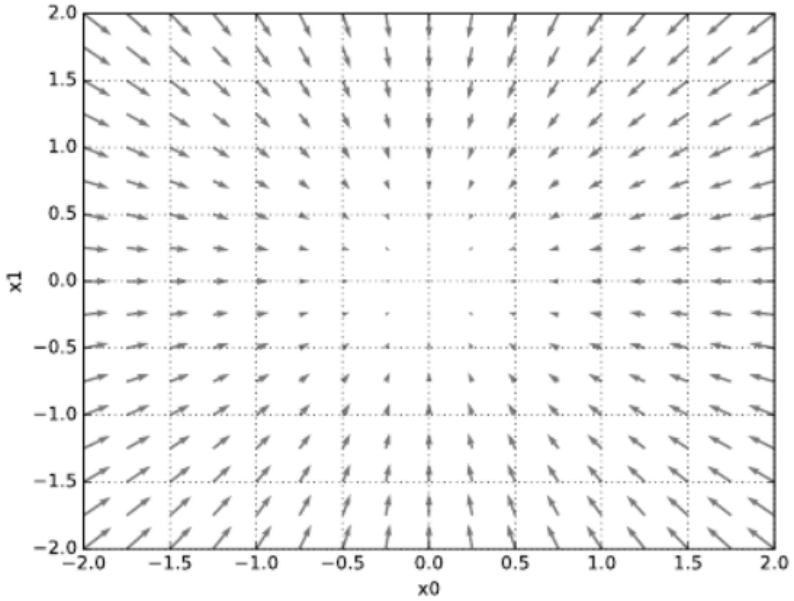
함수의 인수인 f는 함수이고 x는 넘파이 배열이므로 넘파이 배열 x의 각 원소에 대한 수치 미분을 구함

```
>>> numerical_gradient(function_2, np.array([3.0, 4.0]))
array([ 6.,  8.]) *
>>> numerical_gradient(function_2, np.array([0.0, 2.0]))
array([ 0.,  4.])
>>> numerical_gradient(function_2, np.array([3.0, 0.0]))
array([ 6.,  0.])
```

- (3,4), (0,2), (3,0) 세 점 에서의 기울기
  -〉임의의 단위벡터(결과 배열) 방향으로의 점 (x0, x1)에서 함수의 변화률
  -〉이변수 함수의 점 (x0, x1)에서 단위벡터 방향으로 방향도함수

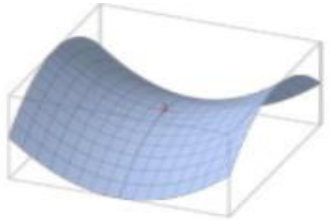
- 기울기의 결과에 마이너스를 붙인 벡터

그림 4-9 $f(x_0, x_1) = x_0^2 + x_1^2$의 기울기



- 기울기는 함수의 '가장 낮은 장소(최솟값)'를 가리킴. 한 점을 가리키고 있으며 '가장 낮은 곳'에서 멀어질수록 화살표의 크기가 커짐.
- 위 그림에서 기울기는 가장 낮은 장소를 가리키지만, 실제는 반드시 그렇다고 할 수 없음. 사실 기울기는 각 지점에서 낮아지는 방향을 가리킴. 더 정확히 말하자면 **기울기가 가리키는 쪽은 각 장소에서 함수의 출력값을 가장 크게 줄이는 방향**임!!

4.4 기울기 모든 변수의 편미분을 벡터로 정리한 것



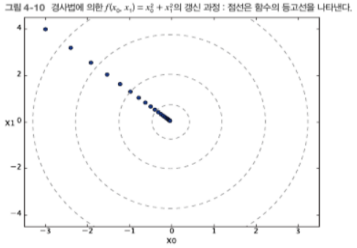
4.4.1 경사법(경사하강법)

- 학습 단계에서 최적의 매개변수를 찾아냄
  -〉 최적이란 손실함수가 최솟값이 될 때의 매개변수 값
  -〉 기울기를 잘 이용해 함수의 최솟값을 찾으려는 것
- 경사법은 현 위치에서 기울어진 방향으로 일정 거리만큼 이동함. 그런 다음 이동한 곳에서도 마찬가지로 기울기를 구하고, 또 그 기울어진 방향으로 나아가기를 반복함. 이렇게 해 함수의 값을 점차 줄이는 것이 **경사법**임. 경사법은 기계학습과 신경망 학습에서 흔히 쓰임

$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

- η는 갱신하는 양, **학습률**을 나타냄
- 한번의 학습으로 얼마만큼 학습해야 할지, 매개변수 값을 얼마나 갱신하느냐를 정하는 것
- 학습률 값은 0.01이나 0.001 등 미리 특정한 값으로 정해두어야 하는데, 일반적으로 이 값이 너무 크거나 작으면 '좋은 장소'를 찾아갈 수 없음. 신경망 학습에서는 보통 이 학습률의 값을 변경하면서 올바르게 학습하고 있는지를 확인하면서 진행 -〉 하이퍼파라미터



- 식 4.7은 1회에 해당하는 갱신이고, 이 단계를 반복함. 즉, 식 4.7처럼 변수의 값을 갱신하는 단계를 여러 번 반복하면서 서서히 함수의 값을 줄이는 것!

- 경사하강법의 구현

```
def gradient_descent(f, init_x, lr = 0.01, step_num = 100):
    x = init_x

    for i in range(step_num):
        grad = numerical_gradient(f, x)
        x -= lr * grad
    return x
```

➢ 인수 f: 최적화하려는 함수, init_x: 초깃값, lr: learning rate를 의미하는 학습률, step_num: 경사법에 따른 반복 횟수

➢ 함수의 기울기는 numerical_gradient(f,x)로 구하고, 그 기울기에 학습률을 곱한 값으로 갱신하는 처리를 step_num번 반복

- 함수의 극솟값을 구할 수 있고 잘하면 최솟값을 구할 수도 있음

```
# 학습률이 너무 큰 예 : lr=10.0
>>> init_x = np.array([-3.0, 4.0])
>>> gradient_descent(function_2, init_x=init_x, lr=10.0, step_num=100)
array([ -2.58983747e+13,  -1.29524862e+12])

# 학습률이 너무 작은 예 : lr=1e-10
>>> init_x = np.array([-3.0, 4.0])
>>> gradient_descent(function_2, init_x=init_x, lr=1e-10, step_num=100)
array([-2.99999994,  3.99999992])
```

- 학습률이 너무 크거나 너무 작으면 너무 큰 값으로 발산해 버리거나, 너무 작으면 거의 갱신되지 않은 채 끝나버린다.
- 매개변수 VS 하이퍼파라미터

4.4 기울기 모든 변수의 편미분을 벡터로 정리한 것

4.4.1 신경망에서의 기울기

- 가중치 매개변수에 대한 손실 함수의 기울기임
- 예를 들어 형상이 2X3, 가중치가 W, 손실함수가 L인 신경망을 생각해 볼 때, 경사는 $\frac{\partial L}{\partial W}$로 나타낼 수 있음

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$$\frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{23}} \end{pmatrix}$$

- $\frac{\partial L}{\partial W}$의 각 원소는 각각의 원소에 관한 편미분. 1행 1번째 원소는 w11를 조금 변경했을 때 손실함수 L이 얼마나 변화하느냐를 나타냄

```
>>> net = simpleNet()
>>> print(net.W) # 가중치 매개변수
[[ 0.47355232  0.9977393   0.84668094]
 [ 0.85557411  0.03563661  0.69422093]]
```

```
>>> x = np.array([0.6, 0.9])
>>> p = net.predict(x)
>>> print(p)
[ 1.05414809 0.63071653 1.1328074]
>>> np.argmax(p) # 최댓값의 인덱스
2
>>>
>>> t = np.array([0, 0, 1]) # 정답 레이블
>>> net.loss(x, t)
0.92806853663411326
```

- 간단한 신경망을 예로 들어 실제로 기울기를 구하는 코드 구현

① simpleNet 클래스

```
class simpleNet:
    def __init__(self):
        self.W = np.random.randn(2,3) # 정규분포로 초기화

    def predict(self, x):
        return np.dot(x, self.W)

    def loss(self, x, t):
        z = self.predict(x)
        y = softmax(z)
        loss = cross_entropy_error(y, t)

        return loss
```

✓ common/functions.py에 정의한 softmax와 cross_entropy_error 메서드 이용

✓ common/gradient.py에 정의한 numerical_gradient 메서드 이용

✓ simpleNet클래스는 형상이 2X3인 가중치 매개변수 하나를 인스턴스 변수로 갖는다. 메서드는 2개인데, 하나는 예측을 수행하는 predict(x)이고, 다른 하나는 손실함수의 값을 구하는 loss(x,t)이다.

✓ 인수 x는 입력 데이터, t는 정답 레이블

4.4 기울기 모든 변수의 편미분을 벡터로 정리한 것

② 기울기 구현

✓ numerical_gradient(f,x)를 써서 구함(여기에서 정의한 f(W) 함수의 인수 W는 더미로 만든 것. numerical_gradient(f,x) 내부에서 f(x)를 실행하는데, 그와의 일관성을 위해 f(W)를 정의한 것)

```
>>> def f(W):
...     return net.loss(x, t)
...
>>> dW = numerical_gradient(f, net.W)
>>> print(dW)
[[ 0.21924763  0.14356247 -0.36281009]
 [ 0.32887144  0.2153437  -0.54421514]]
```

✓ 인수f는 함수, x는 함수 f의 인수. 그래서 여기에서는 net.W를 인수로 받아 손실함수를 계산하는 새로운 함수 f를 정의
그리고 이 새로 정의한 함수를 numerical_gradient(f,x)에 넘김.

✓ dW는 numerical_gradient(f, net.W)의 결과
$\frac{\partial L}{\partial W}$의 $\frac{\partial L}{\partial W11}$은 대략 0.2 =〉 w11을 h만큼 늘리면 손실함수의 값은 0.2h만큼 증가한다는 의미 -〉 손실함수를 줄이려면 음의 방향으로 갱신
$\frac{\partial L}{\partial W}$의 $\frac{\partial L}{\partial W23}$은 대략 -0.5=〉 $\frac{\partial L}{\partial W23}$을 h만큼 늘리면 손실함수의 값은 0.5h만큼 감소한다는 의미 -〉손실함수를 줄이려면 양의 방향으로 갱신

✓ 한번에 갱신되는 양에는 $\frac{\partial L}{\partial W23}$이 w11보다 크게 기여

✓ 신경망의 기울기를 구한 다음에는 경사법에 따라 가중치 매개변수를 갱신하기만 하면 됨.

• numerical_gradient(f,x)가 다차원배열 W를 처리할 수 있도록 구현

```
def numerical_gradient_2d(f, X):
    if X.ndim == 1:
        return _numerical_gradient_1d(f, X)
    else:
        grad = np.zeros_like(X)

        for idx, x in enumerate(X):
            grad[idx] = _numerical_gradient_1d(f, x)

        return grad

def numerical_gradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)

        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)

        x[idx] = tmp_val # 값 복원
        it.iternext()

    return grad
```

nditer: 행렬 원소 접근, 명시적 인덱싱, 슬라이싱 이외에 행렬 모든 원소에 접근할 경우 사용 가능

it.finished: 이터레이터가 finished 위치가 아닐 동안 반복

it.iternext(): 이터레이터를 다음 위치로 넘기기

4.5 학습 알고리즘 구현하기 '손실 함수', '미니배치', '기울기', '경사 하강법'

- 신경망 학습의 절차

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.

- 신경망 학습이 이루어지는 순서, 이는 경사 하강법으로 매개변수를 갱신하는 방법이며, 이때 데이터를 미니배치로 무작위로 선정하기 때문에 **확률적 경사 하강법**이라고 부름. SGD라는 함수로 이 기능을 구현함.

4.5.1 2층 신경망 클래스 구현하기

- 2층 신경망(은닉층이 1개인 네트워크)을 대상으로 MNIST 데이터셋을 사용하여 학습을 수행

```python
class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    def predict(self, x):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']

        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)

        return y

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)

        return cross_entropy_error(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy
```

```python
    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

        return grads

    def gradient(self, x, t):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']
        grads = {}

        batch_num = x.shape[0]

        # forward
        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)

        # backward
        dy = (y - t) / batch_num
        grads['W2'] = np.dot(z1.T, dy)
        grads['b2'] = np.sum(dy, axis=0)

        da1 = np.dot(dy, W2.T)
        dz1 = sigmoid_grad(a1) * da1
        grads['W1'] = np.dot(x.T, dz1)
        grads['b1'] = np.sum(dz1, axis=0)

        return grads
```

4.5 학습 알고리즘 구현하기 ‘손실 함수’, ‘미니배치’, ‘기울기’, ‘경사 하강법’

표 4-1 TwoLayerNet 클래스가 사용하는 변수

변수	설명
params	신경망의 매개변수를 보관하는 딕셔너리 변수(인스턴스 변수) params['W1']은 1번째 층의 가중치, params['b1']은 1번째 층의 편향 params['W2']는 2번째 층의 가중치, params['b2']는 2번째 층의 편향
grads	기울기 보관하는 딕셔너리 변수(numerical_gradient() 메서드의 반환 값) grads['W1']은 1번째 층의 가중치의 기울기, grads['b1']은 1번째 층의 편향의 기울기 grads['W2']는 2번째 층의 가중치의 기울기, grads['b2']는 2번째 층의 편향의 기울기

표 4-2 TwoLayerNet 클래스의 메서드

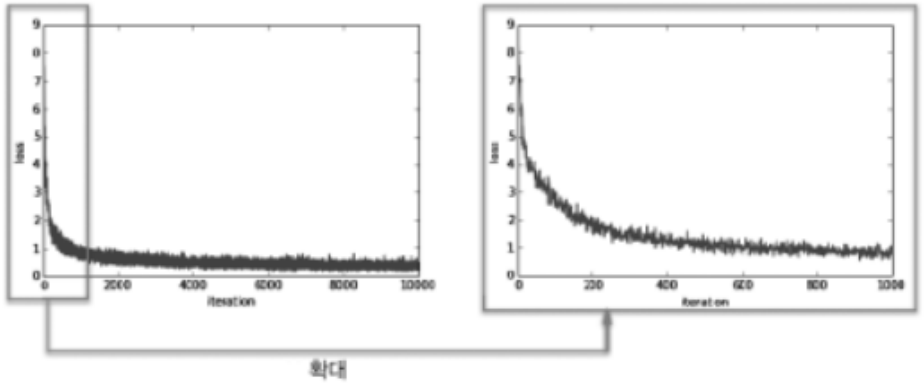
메서드	설명
__init__(self, input_size, hidden_size, output_size)	초기화를 수행한다. 인수는 순서대로 입력층의 뉴런 수, 은닉층의 뉴런 수, 출력층의 뉴런 수
predict(self, x)	예측(추론)을 수행한다. 인수 x는 이미지 데이터
loss(self, x, t)	손실 함수의 값을 구한다. 인수 x는 이미지 데이터, t는 정답 레이블(아래 칸의 세 메서드의 인수들도 마찬가지)
accuracy(self, x, t)	정확도를 구한다.
numerical_gradient(self, x, t)	가중치 매개변수의 기울기를 구한다.
gradient(self, x, t)	가중치 매개변수의 기울기를 구한다. numerical_gradient()의 성능 개선판! 구현은 다음 장에서…

- __init__ 메서드: 초기화 메서드. 가중치 매개변수(어떻게 초기화 하느냐가 신경망 학습의 성공을 좌우하기도 함)도 당장은 정규분포를 따르는 난수로, 편향은 0으로 초기화함
- numerical_gradient(): 수치 미분 방식으로 각 매개변수의 손실함수에 대한 기울기 계산
- gradient(self, x,t): 오차역전파법을 사용하여 기울기를 계산

4.5.2 미니배치 학습 구현하기

- 미니 배치 크기: 100
  매번 60000개의 훈련 데이터에서 임의로 100개의 데이터(이미지 데이터와 정답 레이블 데이터)를 추려 냄
- 그 100개의 미니배치를 대상으로 확률적 경사 하강법을 수행해 매개변수 갱신
- 경사법에 의한 갱신 횟수(반복 횟수)를 10000번으로 설정하고, 갱신 때마다 훈련 데이터에 대한 손실 함수를 계산하고, 그 값을 배열에 추가

그림 4-11 손실 함수 값의 추이 : 왼쪽은 10,000회 반복까지의 추이, 오른쪽은 1,000회 반복까지의 추이



- 학습 횟수가 늘어나면서 손실 함수의 값이 줄어듦. 이는 학습이 잘 되고 있다는 뜻으로, 신경망의 가중치 매개변수가 서서히 데이터에 적응하고 있음을 의미. 바로 신경망이 학습하고 있다는 것. 다시 말해 데이터를 반복해서 학습함으로써 최적 가중치 매개변수로 서서히 다가서고 있음!

4.5 학습 알고리즘 구현하기 '손실 함수', '미니배치', '기울기', '경사 하강법'

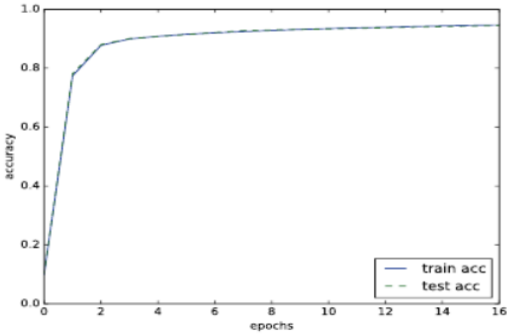
```
# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0]
batch_size = 100   # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)

for i in range(iters_num):
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)
    grad = network.gradient(x_batch, t_batch)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    # 1에폭당 정확도 계산
    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

4.5.3 시험 데이터로 평가하기

- 그림 4-11의 결과에서 학습을 반복함으로써 손실 함수의 값이 서서히 내려가는 것을 확인. 이때의 손실 함수의 값은 정확히는 '훈련 데이터의 미니배치에 대한 손실 함수'의 값임!!
- 신경망 학습에서는 훈련 데이터 외의 데이터를 올바르게 인식하는지 확인. =〉 오버피팅을 일으키지 않는지 확인
- 1에폭별로 훈련 데이터와 시험 데이터에 대한 정확도를 기록함.

➢ 1에폭마다 모든 훈련데이터와 시험 데이터에 대한 정확도를 계산하고, 기록. 정확도를 1에폭마다 계산하는 이유는 for문 안에서 매번 계산하기에는 시간이 오래 걸리고, 그럴 필요가 없기 때문.

그림 4-12 훈련 데이터와 시험 데이터에 대한 정확도 추이



- 훈련데이터에 대한 정확도는 실선으로, 시험 데이터에 대한 정확도를 점선으로
- 에폭이 진행될수록 정확도가 좋아지고 있음.
- 두 정확도에는 차이가 없고 이번 학습에서는 오버피팅이 일어나지 않았음

- 만약 오버피팅이 일어난다면 모습이 어떻게 달라질까? 어느 순간부터 시험 데이터에 대한 정확도가 점차 떨어지기 시작. 이 순간을 포착해 학습을 중단하면 오버피팅을 예방 -〉 조기종료

4.6 정리

이번 장에서 배운 내용

- 기계학습에서 사용하는 데이터셋은 훈련 데이터와 시험 데이터로 나눠 사용한다.
- 훈련 데이터로 학습한 모델의 범용 능력을 시험 데이터로 평가한다.
- 신경망 학습은 손실 함수를 지표로, 손실 함수의 값이 작아지는 방향으로 가중치 매개변수를 갱신한다.
- 가중치 매개변수를 갱신할 때는 가중치 매개변수의 기울기를 이용하고, 기울어진 방향으로 가중치의 값을 갱신하는 작업을 반복한다.
- 아주 작은 값을 주었을 때의 차분으로 미분하는 것을 수치 미분이라고 한다.
- 수치 미분을 이용해 가중치 매개변수의 기울기를 구할 수 있다.
- 수치 미분을 이용한 계산에는 시간이 걸리지만, 그 구현은 간단하다. 한편, 다음 장에서 구현하는 (다소 복잡한) 오차역전파법은 기울기를 고속으로 구할 수 있다.