

CHAPTER 4

신경망 학습

4.1 데이터에서 학습한다!

- 학습이란 훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득하는 것.
- 손실 함수의 결괏값을 가장 작게 만드는 가중치 매개변수를 찾는 것이 학습의 목표.
- 데이터에서 학습한다는 것은 가중치 매개변수의 값을 데이터를 보고 자동으로 결정한다는 뜻.

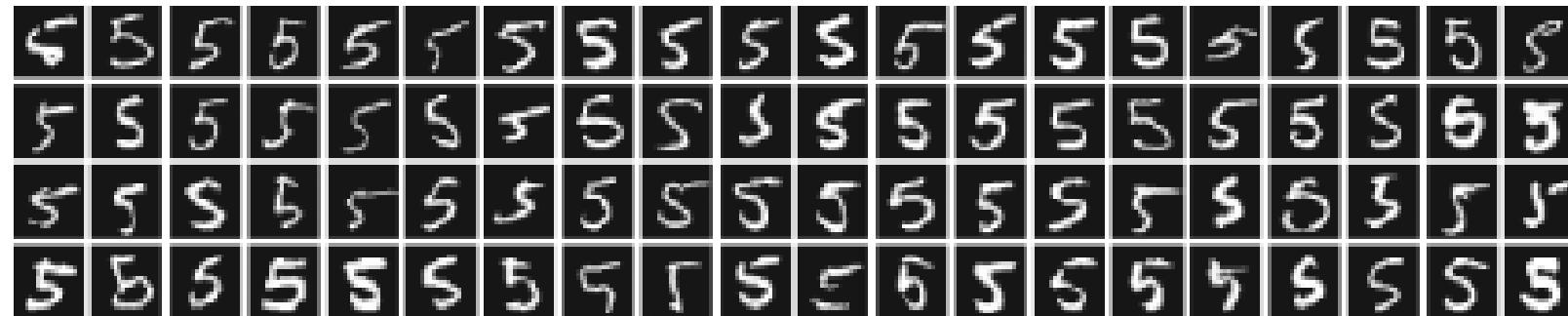
기계학습은 데이터가 생명!!

- 문제해결 방식

- 사람: 사람의 경험과 직관을 단서로 시행착오를 거듭 하며 일을 진행.
- 기계학습: 사람의 개입을 최소화하고 수집한 데이터로부터 패턴을 찾으려 시도

'5'를 분류하기

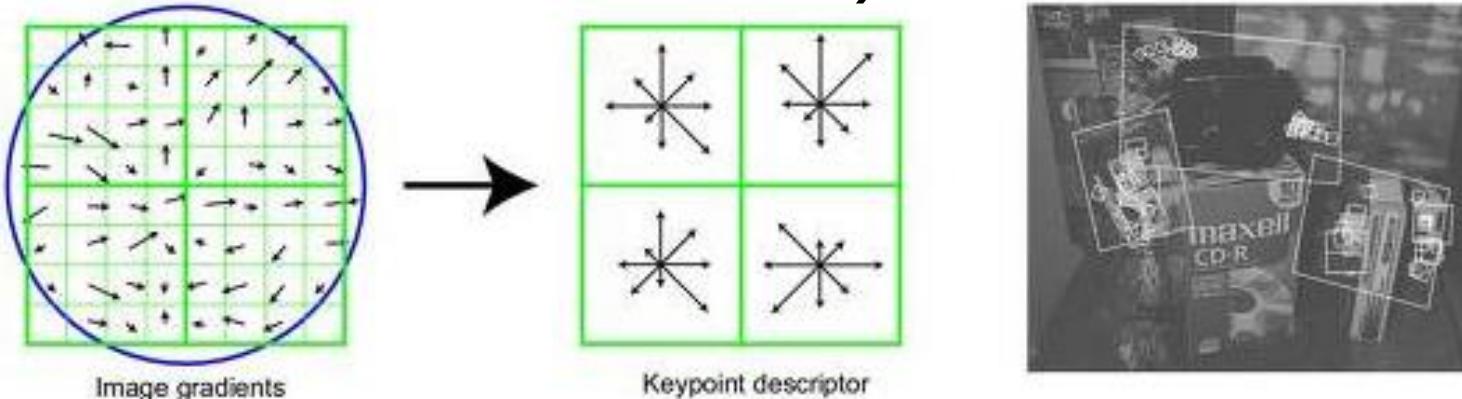
그림 4-1 손글씨 숫자 '5'의 예 : 사람마다 자신만의 패턴이 있다.



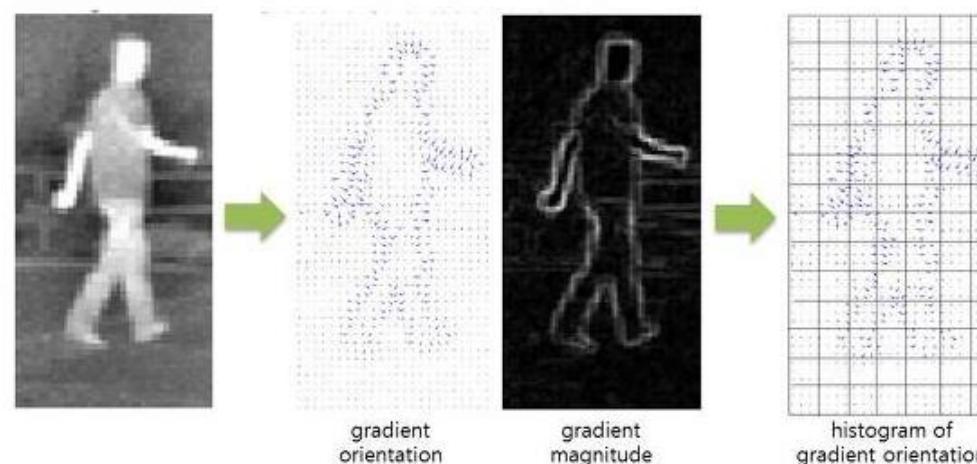
- '5'를 제대로 분류하는 프로그램을 직접 고안해 설계하기란 의외로 어려운 문제
- 그렇다면 이미지의 특징을 추출하고 그 특징의 패턴을 기계학습으로 학습하자.

'5'를 분류하기 - 특징 + 머신러닝

- 컴퓨터 비전 분야에서는 SIFT, SURF, HOG 등의 특징을 사용하고 이를 기반으로 지도 학습 방식의 대표 분류 기법인 SVM, KNN 등으로 학습.
- SIFT(Scale Invariant Feature Transform)

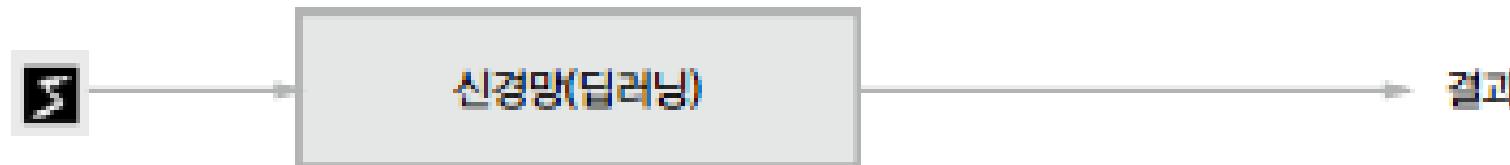
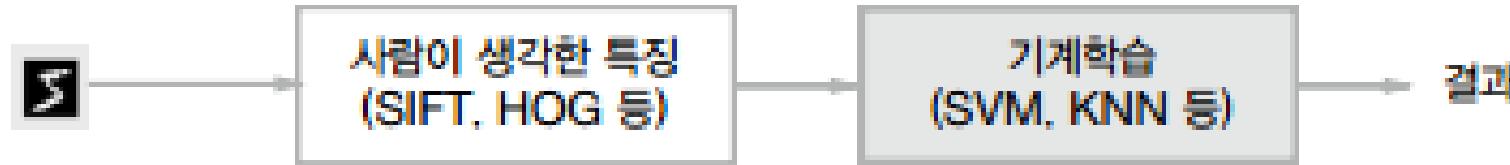


- HOG(Histogram of Oriented Gradient)



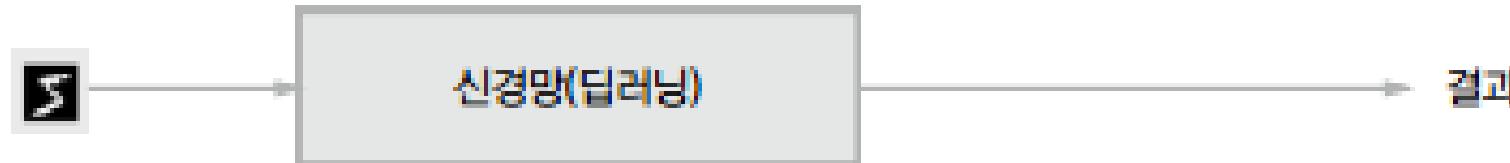
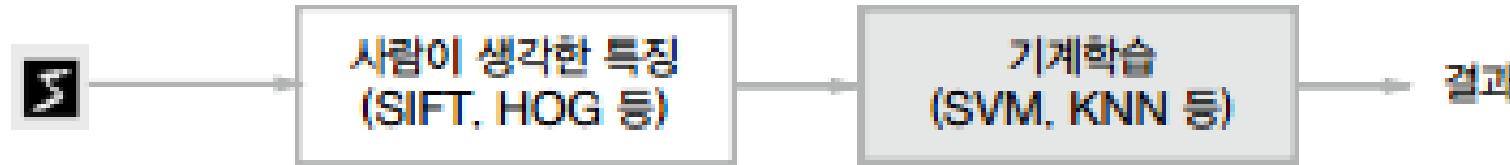
'5'를 분류하기 – 신경망

- 신경망은 이미지를 ‘있는 그대로’ 학습.
- 신경망은 이미지에 포함된 중요한 특징까지도 ‘기계’가 스스로 학습한다.
- 딥러닝을 종단간 기계학습(end-to-end machine learning)이라고도 한다.



'5'를 분류하기 – 신경망

- 신경망은 이미지에 포함된 중요한 특징까지도 '기계'가 스스로 학습한다.
- 결국 신경망은 모든 문제를 주어진 데이터 그대로를 입력 데이터로 활용해 'end-to-end'로 학습하므로 종단간 기계학습 (end-to-end machine learning)이라고도 함.



훈련 데이터와 시험 데이터

- 기계학습 문제는 데이터를 훈련 데이터`training data`와 시험 데이터`test data`로 나눠 학습과 실험을 수행하는 것이 일반적.
- 범용적으로 사용할 수 있는 모델을 원하기 때문에 훈련 데이터와 시험 데이터를 분리하여 모델링한다.
- 과적합(Overfitting)을 피하는 것은 기계학습의 중요한 과제.

4.2 손실 함수

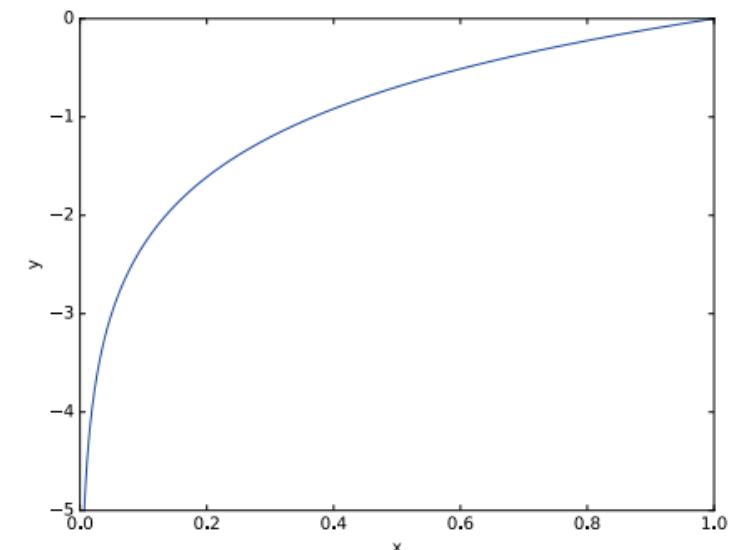
- 신경망 학습에서는 현재의 상태를 '하나의 지표'로 표현.
- 손실 함수 loss function를 기준으로 최적의 매개변수 값을 탐색하고 평균 제곱 오차, 교차 엔트로피 오차를 일반적으로 사용.
 - 평균 제곱 오차

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

- 교차 엔트로피 오차

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

그림 4-3 자연로그 $y = \log x$ 의 그래프



4.2 손실 함수

```
y1 = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
```

```
y2 = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
```

```
t = [ 0, 0, 1, 0, 0, 0, 0, 0, 0] <- 원-핫 인코딩.
```

```
Mean_squared_error(np.array(y1), np.array(t))
```

```
0.09750000000000031
```

```
mean_squared_error(np.array(y2), np.array(t))
```

```
0.5975000000000003
```

```
cross_entropy_error(np.array(y1), np.array(t))
```

```
0.51082545709933802
```

```
cross_entropy_error(np.array(y2), np.array(t))
```

```
2.3025840929945458
```

4.2.3 미니배치 학습

훈련 데이터 모두에 대한 손실 함수의 합을 구하는 방법

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

단. 평균을 구해 사용하면 훈련 데이터 개수와 관계없이 언제든 통일된 지표를 얻을 수 있으나 빅데이터 수준이 되면 손실함수를 계산하는 것도 비용이 큼.

신경망 학습에서 훈련 데이터로부터 일부만 골라 학습을 수행.
이때 이 일부를 미니배치mini-batch라고 함.

4.2.3 미니배치 학습

```
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)) / batch_size #<- 원핫 인코딩
#return -np.sum(np.log(y[np.arange(batch_size), t])) / batch_size <- 정답레이블
```

```
y[np.arange(batch_size), t] -> [[ 0.6]]
print(np.log(y[np.arange(batch_size), t]))
[[-0.51082562]]
print(t * np.log(y+delta))
[[-0. -0. -0.51082546 -0. -0. -0. -0. -0. -0.]]
```

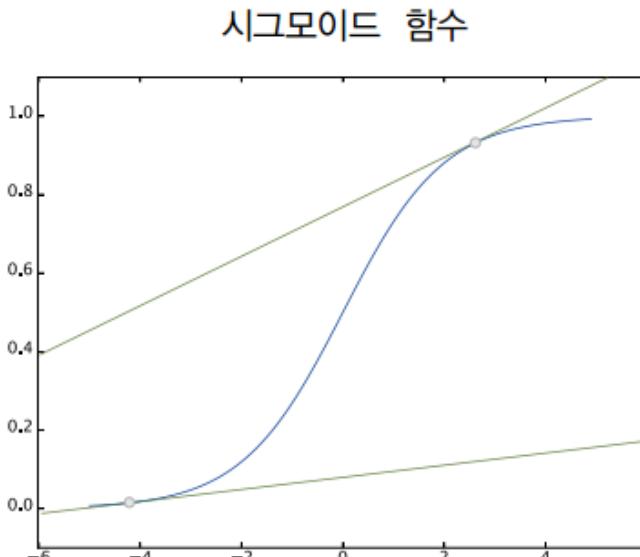
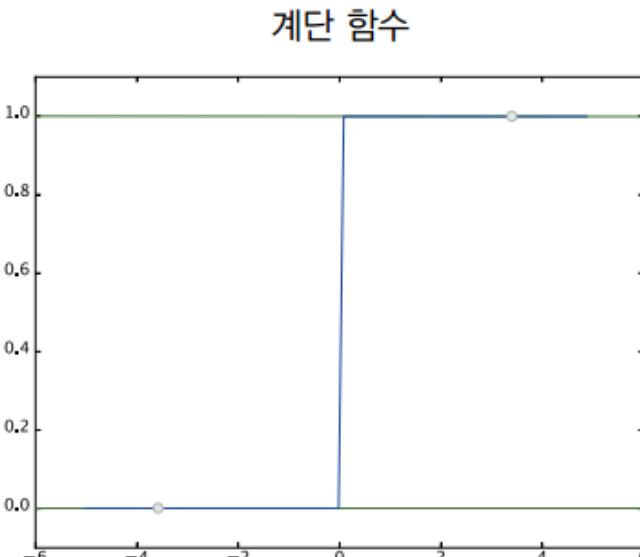
4.2.4 미니배치 학습

```
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)) / batch_size #<- 원핫 인코딩
#return -np.sum(np.log(y[np.arange(batch_size), t])) / batch_size <- 정답레이블
```

```
y[np.arange(batch_size), t] -> [[ 0.6]]
print(np.log(y[np.arange(batch_size), t]))
[[-0.51082562]]
print(t * np.log(y+delta))
[[-0. -0. -0.51082546 -0. -0. -0. -0. -0. -0.]]
```

4.2.5 왜 손실 함수를 설정하는가?

- 신경망에서 가중치 매개변수의 손실 함수의 미분이란 ‘가중치 매개변수의 값을 아주 조금 변화 시켰을 때, 손실 함수가 어떻게 변하나’라는 의미.
 - 미분값의 변화(음수, 양수 등)에 따라 가중치 매개변수를 변화 시켜 손실함수 값을 조정.
- 신경망을 학습할 때 정확도를 지표로 삼아서는 안 된다. 정확도를 지표로 하면 매개 변수의 미분이 대부분의 장소에서 0이 되기 때문.
 - 예) 한 신경망이 100장의 훈련 데이터 중 32장을 올바로 인식했을 때, 가중치 매개변수의 값을 조금 바꾸어서 재측정시 정확도는 그대로 32% 수준. 만약 크게 바꾼다고 해도 그 특성상 32.0123%와 같은 연속적인 변화보다는 33%나 34%처럼 불연속적인 값으로 변화.



4.3 미분

- 10분에서 2km씩 달렸다고 했을 때. 이때의 속도는 $2 / 10 = 0.2$ [km/분]
즉, 1분에 0.2km만큼의 속도로 변화했다고 할 수 있다.
- 10분이라는 시간을 가능한 한 줄여 어느 한순간의 변화량(어느 순간의 속도)를 구하고 싶을 때, 미분을 사용해 '특정 순간'의 변화량을 구할 수 있다.

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

4.3.1 수치 미분

- # 나쁜 구현 예

```
def numerical_diff(f, x):
```

```
    h = 10e-50
```

```
    return (f(x + h) - f(x)) / h <- 미분이란 x 위치의 기울기,
```

0으로 h 를 수렴하는 것은 불가능하므로 결국
 $x + h, x$ 사이의 기울기로 표현될 수 밖에 없다.

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

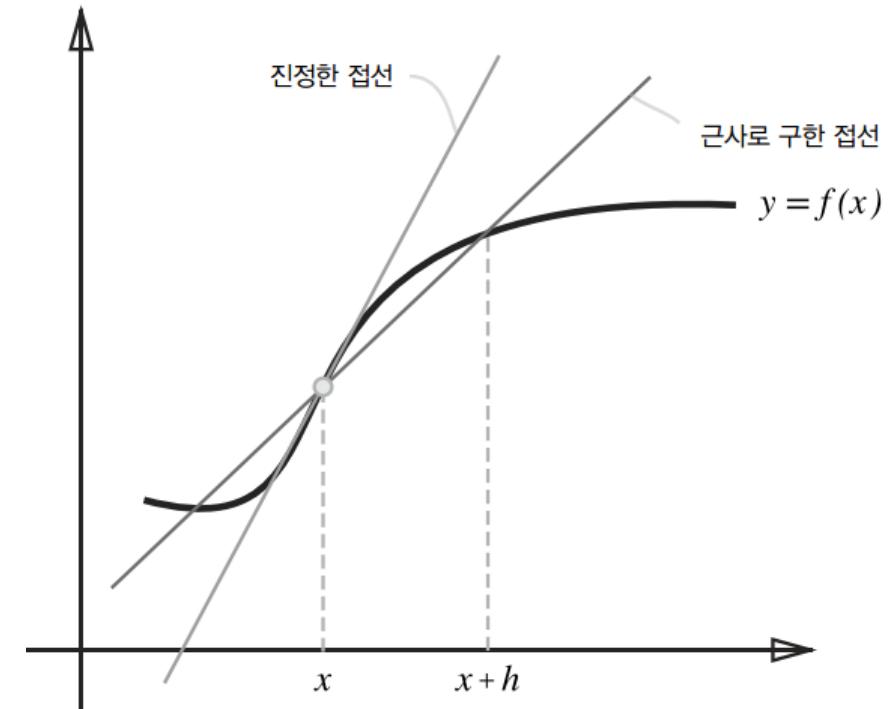
0.0 <- 반올림 오차 rounding error 발생.

개선한 예 (중앙 차분 : $x+h, x-h$ 의 중앙값을 사용)

```
def numerical_diff(f, x):
```

```
    h = 1e-4 # 0.0001
```

```
    return (f(x+h) - f(x-h)) / (2*h)
```

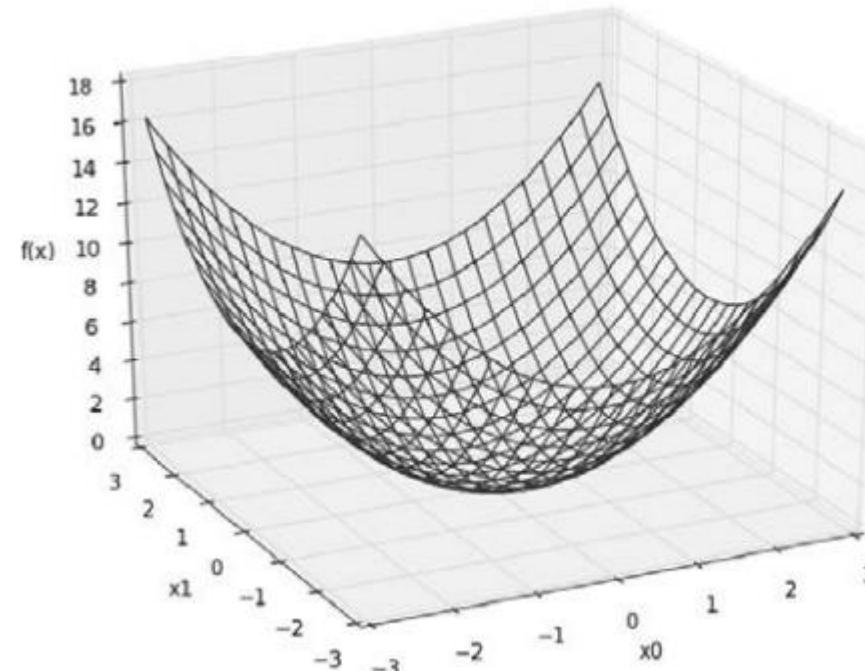


4.3.3 편미분

- 변수가 두개인 함수

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

$f(x_0, x_1) = x_0^2 + x_1^2$ 의 그래프



- 위 함수를 미분한다고 할 때.
- '어느 변수에 대한 미분', 즉 x_0 와 x_1 중 어느 변수에 대한 미분인지 구별 필요.
- 이와 같이 변수가 여럿인 함수에 대한 미분을 '편미분'이라고 한다.

4.3.4 편미분 예

편미분 결과

```
def function_tmp1(x0):
    return x0*x0 + 4.0**2.0
```

```
numerical_diff(function_tmp1, 3.0)
6.00000000000378
```

```
def function_tmp2(x1):
    return 3.0**2.0 + x1*x1
```

```
numerical_diff(function_tmp2, 4.0)
7.99999999999119
```

(3,4) -> 6.0000000000378, 7.99999999999119

4.4 기울기

```
def numerical_gradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x) # x와 형상이 같은 배열을 생성
    print(grad) #-> [ 0. 0.]

    for idx in range(x.size):
        tmp_val = x[idx]
        x[idx] = tmp_val + h
        fxh1 = f(x)
        x[idx] = tmp_val - h
        fxh2 = f(x)
        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val

    return grad

numerical_gradient(function_2, np.array([1.0, 1.5]))
array([ 2., 3.])
```

4.4 기울기

`numerical_gradient(function_2, np.array([1.0, 1.5]))
array([2., 3.])`

$f(x_0, x_1) = x_0^2 + x_1^2$ 의 그라프

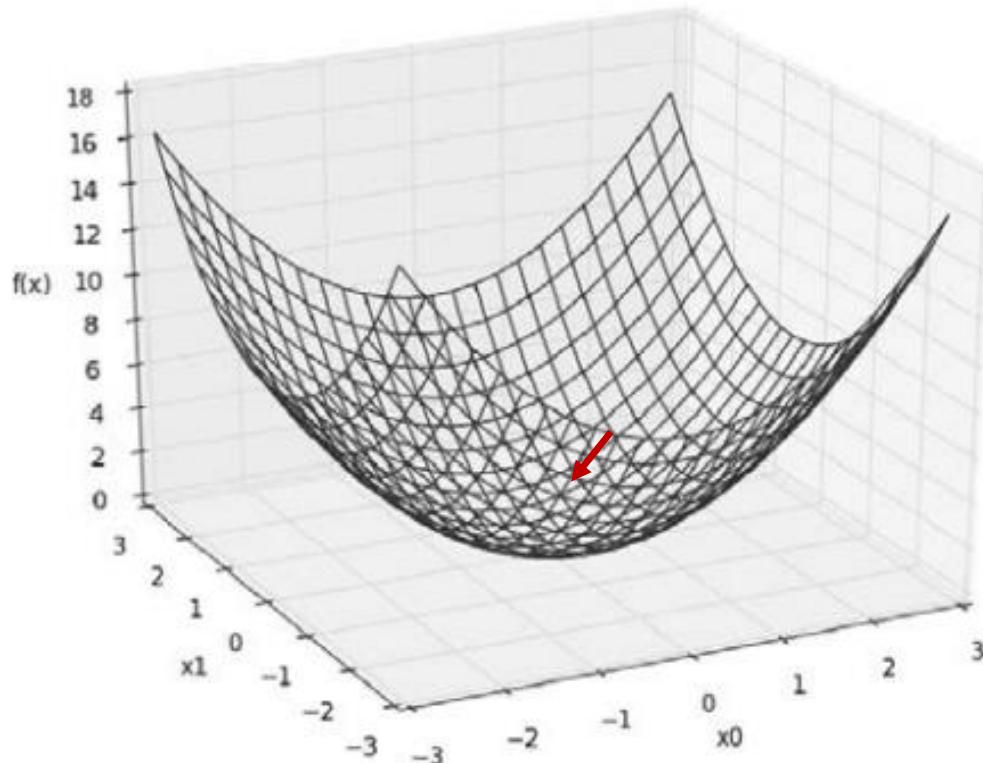
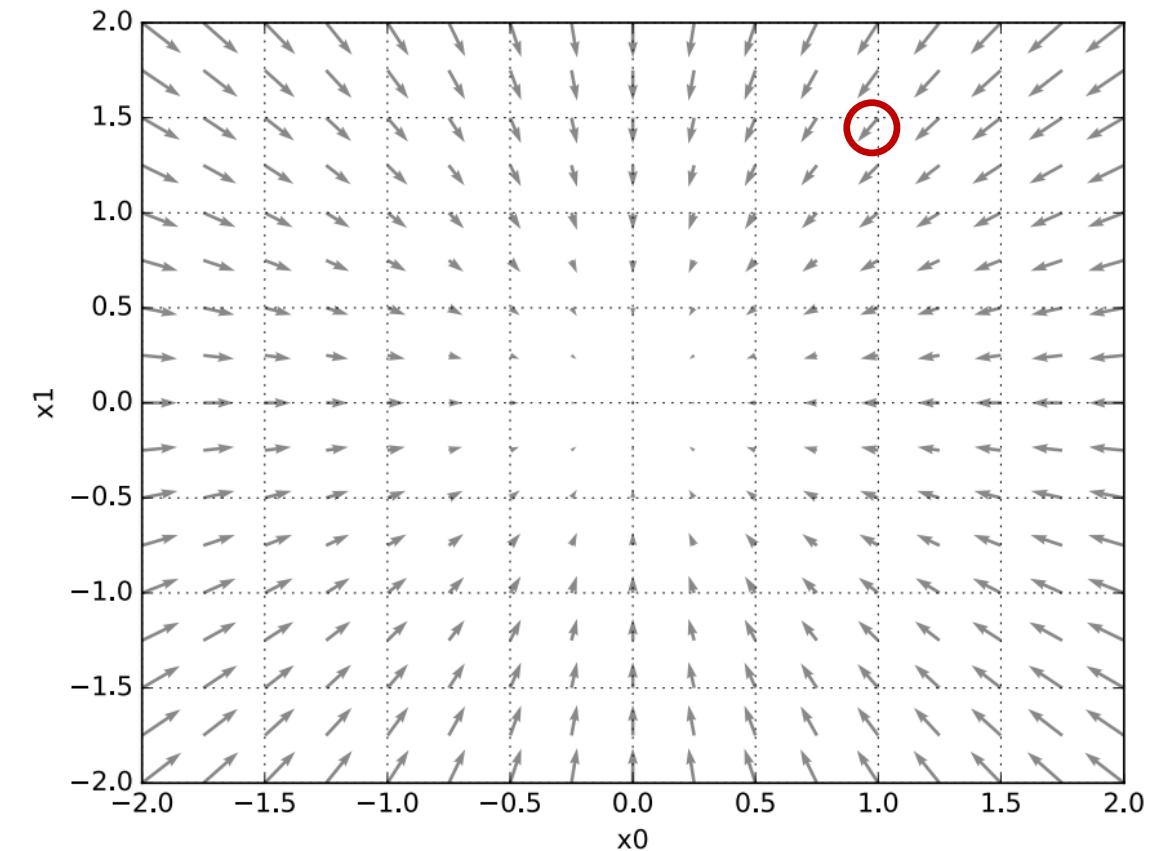


그림 4-9 $f(x_0, x_1) = x_0^2 + x_1^2$ 의 기울기

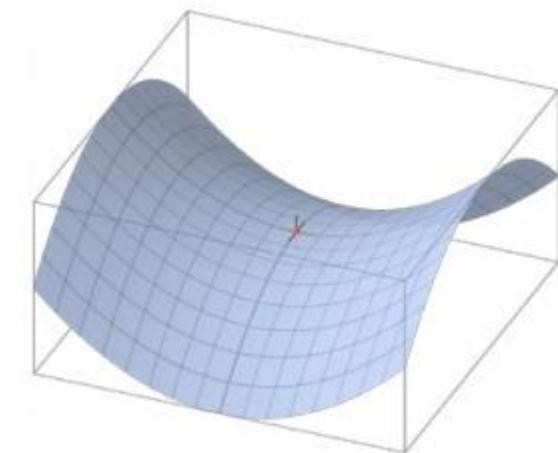


기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 줄이는 방향.

4.4.1 경사하강법

- 기계학습에서 최적(손실함수가 최소값이 되는)의 매개변수를 찾아내는 것이 학습의 목표, 그러나 일반적인 문제의 손실함수는 매우 복잡하고, 매개변수 공간 광대하여 최솟값을 찾기 어려움.
- 이때 사용되는 것이 경사하강법.

WARNING 함수가 극솟값, 최솟값, 또 **안장점**^{saddle point}이 되는 장소에서는 기울기가 0입니다. 극솟값은 국소적인 최솟값 즉 한정된 범위에서의 최솟값인 점입니다. 안장점은 어느 방향에서 보면 극댓값이고 다른 방향에서 보면 극솟값이 되는 점입니다.* 경사법은 기울기가 0인 장소를 찾지만 그것이 반드시 최솟값이라고는 할 수 없습니다(극솟값이나 안장점일 가능성이 있습니다). 또, 복잡하고 찌그러진 모양의 함수라면 (대부분) 평평한 곳으로 파고들면서 **고원**^{plateau, 플래토}이라 하는, 학습이 진행되지 않는 정체기에 빠질 수 있습니다.



4.4.1 경사하강법

- 인수 f 는 최적화하려는 함수, $init_x$ 는 초기값, lr 은 learning rate를 의미하는 학습률, $step_num$ 은 경사법에 따른 반복 횟수

```
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
```

```
>>> def function_2(x):  
...     return x[0]**2 + x[1]**2  
...  
>>> init_x = np.array([-3.0, 4.0])  
>>> gradient_descent(function_2, init_x=init_x, lr=0.1, step_num=100)  
array([-6.11110793e-10, 8.14814391e-10])
```

4.4.1 경사하강법

- 인수 f는 최적화하려는 함수, init_x는 초기값, lr은 learning rate를 의미하는 학습률, step_num은 경사법에 따른 반복 횟수

```
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
```

```
>>> def function_2(x):  
...     return x[0]**2 + x[1]**2  
...  
>>> init_x = np.array([-3.0, 4.0])  
>>> gradient_descent(function_2, init_x=init_x, lr=0.1, step_num=100)  
array([-6.1110793e-10, 8.14814391e-10])
```

결과 출력.

[-2.4 3.2]
[-1.92 2.56]
[-1.536 2.048]
[-1.2288 1.6384]
[-0.98304 1.31072]
[-0.786432 1.048576]

.....

[-1.49196971e-09 1.98929295e-09]
[-1.19357577e-09 1.59143436e-09]
[-9.54860614e-10 1.27314749e-09]
[-7.63888491e-10 1.01851799e-09]
[-6.11110793e-10 8.14814391e-10]

4.4.1 경사하강법

- 학습률을 적절히 설정하는 것이 중요!!!.

- 크면 발산하고 작으면 갱신하지 않고 끝나게 된다.

```
# 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])
```

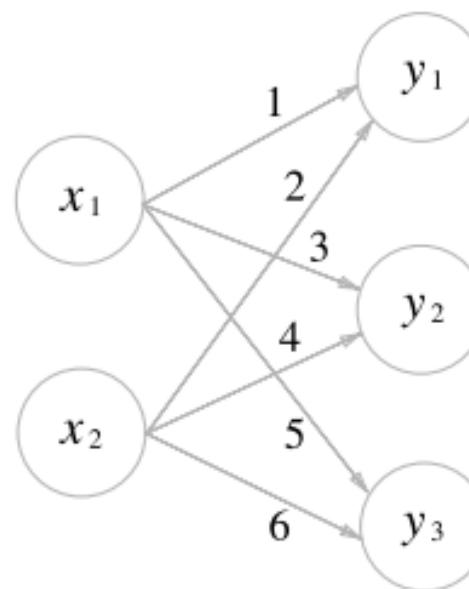
4.4.2 신경망에서의 기울기

- 신경망 학습에서도 기울기(가중치 매개변수에 관한 손실 함수의 기울기)를 구해야 된다.
- 예) 형상이 2×3 , 가중치가 W, 손실 함수가 L인 신경망

$$W = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

$$\frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{31}} \\ \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{32}} \end{pmatrix}$$

그림 3-14 행렬의 곱으로 신경망의 계산을 수행한다.



$$\begin{matrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{matrix} \quad X \quad \begin{matrix} 2 \\ 2 \end{matrix} \quad \begin{matrix} 2 \times 3 \\ \text{일치} \end{matrix} \quad = \quad Y \quad \begin{matrix} 3 \\ 3 \end{matrix}$$

4.4.2 신경망에서의 기울기

```
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  
  
x = np.array([0.6, 0.9])  
t = np.array([0, 0, 1])  
  
net = simpleNet()  
  
f = lambda w: net.loss(x, t)  
dW = numerical_gradient(f, net.W)  
print(dW)
```

```
self.W-----  
[[ 0.03382862 -1.62688388  1.37106193]  
 [ 1.39870468 -1.60908402 -1.30546409]]  
z-----  
[ 1.27919139 -2.42430594 -0.35228052]  
y-----  
[ 0.81948499  0.02018982  0.16032519]  
loss-----  
1.83055111281  
z-----  
[ 1.27907139 -2.42430594 -0.35228052]  
y-----  
[ 0.81946724  0.02019181  0.16034095]  
loss-----  
1.83045277567  
z-----  
[ 1.27913139 -2.42424594 -0.35228052]  
y-----  
[ 0.81947512  0.020192   0.16033288]  
loss-----  
1.83050315546  
z-----  
[ 1.27913139 -2.42430594 -0.35237052]  
y-----  
[ 0.81948794  0.02019111  0.16032095]  
loss-----  
1.83057751454  
print dw-----  
[[ 0.49168567  0.01211449 -0.50380016]  
 [ 0.7375285   0.01817174 -0.75570024]]
```

4.5 학습 알고리즘 구현하기

- 신경망 학습은 아래와 같이 4단계로 구성.

1단계 - 미니배치

훈련 데이터 중 일부를 무작위로 가져오고, 이렇게 선별한 데이터를 미니배치라 함.

이때 미니배치의 손실 함수 값을 줄이는 것을 목표로 한다.

2단계 - 기울기 산출

미니배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기 계산.
기울기는 손실 함수의 값을 가장 작게 하는 방향을 제시.

3단계 - 매개변수 갱신

가중치 매개변수를 기울기 방향으로 아주 조금 갱신.

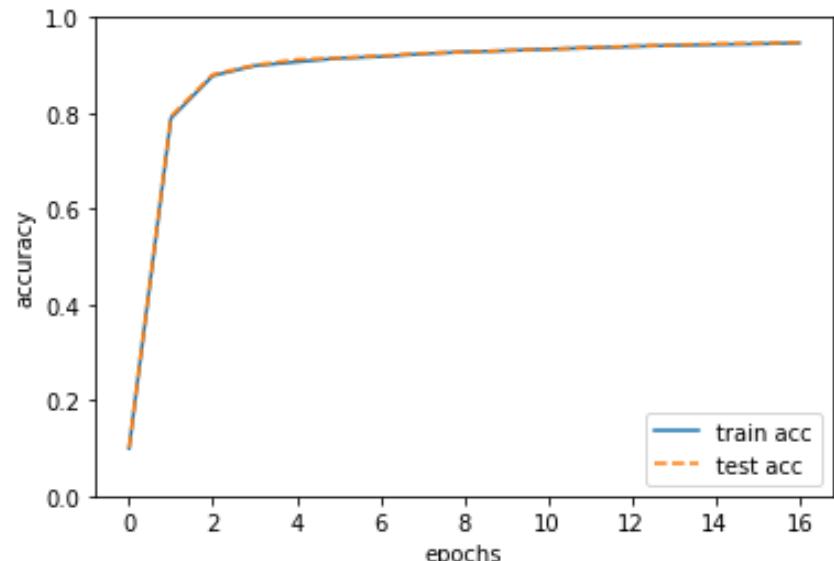
4단계 - 반복

1~3단계를 반복한다.

여기서, 데이터를 미니배치로 무작위로 선정하기 때문에 확률적 경사 하강법 (stochastic gradient descent:SGD)이라 한다.

4.5 학습 알고리즘 구현하기

```
train acc, test acc | 0.09915, 0.1009
train acc, test acc | 0.789016666667, 0.7932
train acc, test acc | 0.877933333333, 0.8805
train acc, test acc | 0.898866666667, 0.9001
train acc, test acc | 0.906783333333, 0.9118
train acc, test acc | 0.9145, 0.915
train acc, test acc | 0.917833333333, 0.9194
train acc, test acc | 0.923483333333, 0.9249
train acc, test acc | 0.927733333333, 0.9274
train acc, test acc | 0.930366666667, 0.9307
train acc, test acc | 0.93355, 0.9332
train acc, test acc | 0.936133333333, 0.9368
train acc, test acc | 0.9391, 0.9396
train acc, test acc | 0.941383333333, 0.9422
train acc, test acc | 0.943033333333, 0.9448
train acc, test acc | 0.944383333333, 0.9455
train acc, test acc | 0.946533333333, 0.9477
```



4.6 정리

이번 장에서 배운 것

- 기계학습에서 사용하는 데이터셋은 훈련 데이터와 시험 데이터로 나눠 사용한다.
- 훈련 데이터에서 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 미분을 구하는 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.