Deep Learning from Scratch

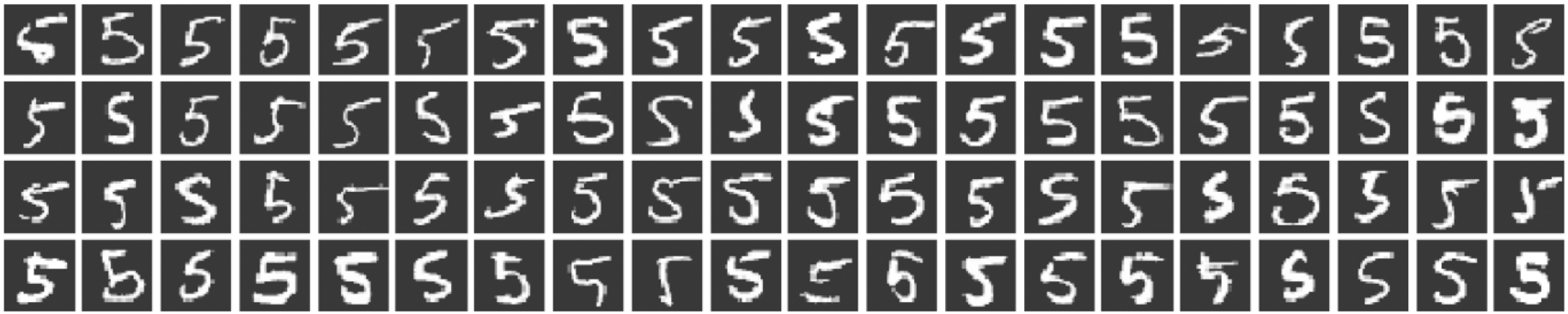
조용군

iceman4u@naver.com

신경망 학습

데이터로부터 학습

- ✓ **아래 손글씨 숫자 이미지로부터 '5'를 인식하는 프로그램을 구현하려면?**
  - 떠오르는 알고리즘?



데이터로부터 학습

✓ **3가지 방법론**

✓ **규칙을 '사람'이 만드는 방식 ➔ '기계'가 데이터로부터 규칙(특징)을 발견해 내는 방식으로 패러다임 전환**

데이터로부터 학습

- ✓ **신경망의 이점**
  - 모든 문제를 같은 맥락에서 풀 수 있다
    - '5'를 인식하는 문제
    - '개'를 인식하는 문제
    - '사람 얼굴'을 인식하는 문제
    - 위 각각에 대해 특징(feature)를 별도로 지정할 필요 없음

- ✓ **종단간 기계학습(end-to-end machine learning)**
  - 데이터(입력)에서 목표한 결과(출력)를 사람의 개입 없이 얻는다

훈련 데이터와 시험 데이터

- ✓ **훈련 데이터(training data)와 시험 데이터(test data)로 나누는 이유?**
  - 범용적으로 사용할 수 있는 모델 필요
    - 범용 능력을 제대로 평가하기 위해
  - 범용 능력
    - 생소한 데이터로도 문제를 올바르게 해결하는 능력
    - 손글씨 숫자 인식
      - '특정인'의 '특정 글자'가 아닌 '일반인'의 '일반 글자'를 인식하는 능력
  - 오버피팅(과적합, overfitting)
    - 특정 데이터 셋에 지나치게 최적화된 상태

손실 함수(비용 함수)

- ✓ **손실 함수**
  - 신경망에서 최적의 매개변수 값을 탐색하는 '하나의 지표'
  - 신경망 성능의 '나쁨'을 나타내는 지표
    - 성능의 '나쁨'을 최소화 = 성능의 '좋음'을 최대화
  - 평균 제곱 오차
  - 교차 엔트로피 오차

손실 함수

✓ 평균 제곱 오차(Mean Squared Error, MSE)

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

- y_k: 신경망의 출력(신경망이 추정한 값)
- t_k: 정답 레이블
- k: 데이터의 차원 수

▪ 오차에 대한 2차 방정식

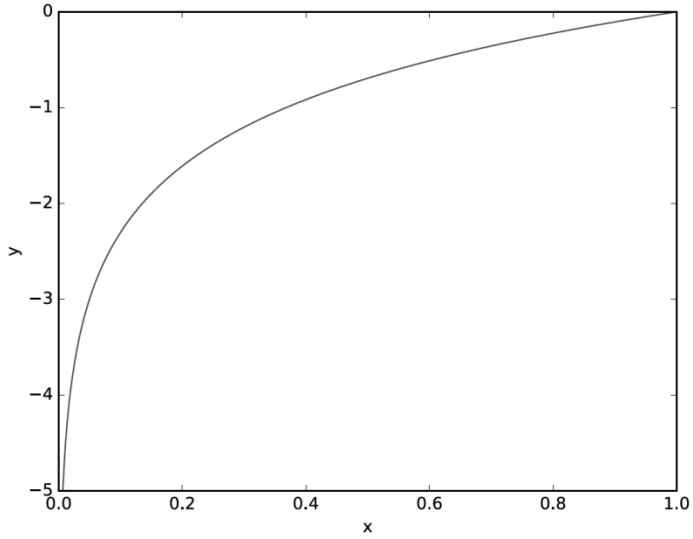
```
# 손글씨 숫자 인식 예에서
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

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

손실 함수

✓ 교차 엔트로피 오차(Cross Entropy Error, CEE)

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

- y_k: 신경망의 출력(신경망이 추정한 값)
- t_k: 정답 레이블
- k: 데이터의 차원 수



- 2개의 확률 분포 사이에 정의되는 척도
  - 정보 엔트로피와 닮은 꼴
- t_k가 원-핫 인코딩이라면 정답일 때의 추정값의 자연로그를 계산하는 식

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

미니 배치 학습

- ✓ **훈련 데이터 모두에 대한 손실 함수 CEE의 합**

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk}\log y_{nk}$$

- ✓ **평균을 구한다는 것은 훈련 데이터 개수와 관계없이 언제든 통일된 지표를 다룬다는 것**

- ✓ **빅데이터 시대**
  - 모든 데이터에 대해 손실함수 값을 구한다는 건 아주 시간이 많이 걸림

- ✓ **미니 배치(mini-batch) 학습**
  - 훈련 데이터 전체가 아닌 일부만 골라서 학습
    - np.random.choice()
  - 전체 데이터에서 무작위로 추출한 표본 ➔ 전체 데이터를 잘 대표

(배치용)교차 엔트로피 오차

✓ 정답 레이블이 원-핫 인코딩인 경우

```
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
```

(배치용)교차 엔트로피 오차

- ✓ 정답 레이블이 '2', '7' 과 같은 숫자 레이블로 주어진 경우

```
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]))\
        / batch_size
```

(배치용)교차 엔트로피 오차

- ✓ **정답 레이블이 원-핫 인코딩이든 숫자 레이블이든 무관**
  - 원-핫 인코딩을 숫자 레이블로 변환

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

    # 훈련 데이터가 원-핫 벡터라면 정답 레이블의 인덱스로 반환
    if t.size == y.size:
        t = t.argmax(axis=1)

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

손실 함수를 설정하는 이유

- ✓ **손실 함수 vs. 정확도**

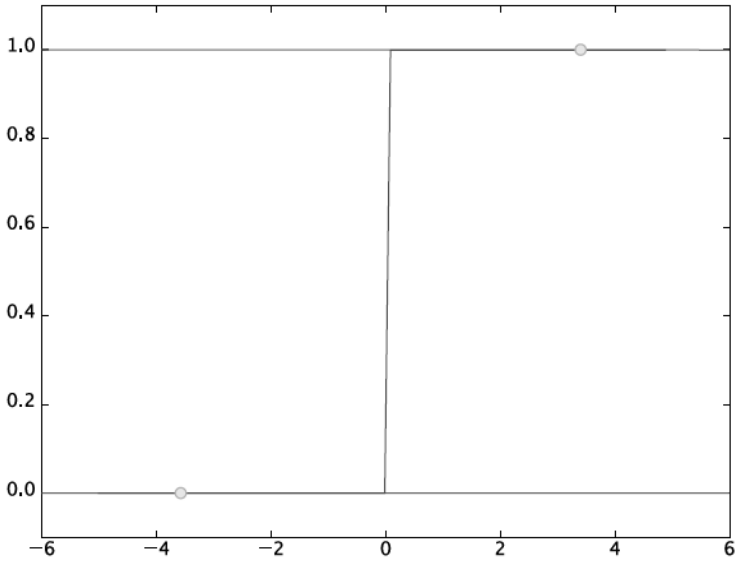
- ✓ **가중치 매개변수에 대한 손실 함수의 미분**
  - 가중치 매개변수의 값을 아주 조금 변화시킬 때, 손실 함수가 어떻게 변하나
  - 손실 함수 값 줄이기
    - 미분 값이 음수 ➔ 해당 가중치 매개변수를 양의 방향으로 변화
    - 미분 값이 양수 ➔ 해당 가중치 매개변수를 음의 방향으로 변화
    - 미분 값이 0 ➔ 해당 가중치 매개변수 갱신 중단

- ✓ **신경망 학습 시 정확도를 지표로 삼아선 안 된다**
  - 정확도에 대한 매개변수의 미분이 대부분의 장소에서 0이 되기 때문
  - 가중치를 조금 변화시켜도 정확도는 거의 그대로이므로
  - 계단 함수를 활성화 함수로 사용하지 않는 이유와도 일맥상통

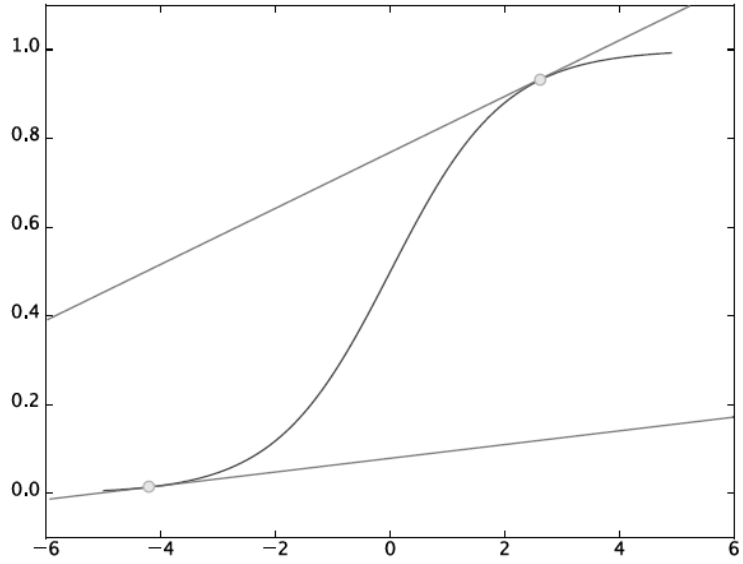
계단 함수와 시그모이드 함수

- ✓ **계단 함수를 활성화 함수로 사용하지 않는 이유**
  - 계단 함수: 대부분의 장소에서 기울기가 0
  - 시그모이드 함수: 어느 장소라도 기울기가 0이 아님

계단 함수



시그모이드 함수



수치 미분

✓ **미분**

- '평균 변화량'이 아닌 '순간 변화량'

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

- x의 '작은 변화'가 함수 $f(x)$를 얼마나 변화시키느냐

- 파이썬으로 수치 미분 구현

```
def numerical_diff(f, x):
    h = 1e-50
    return (f(x+h) - f(x)) / h
```

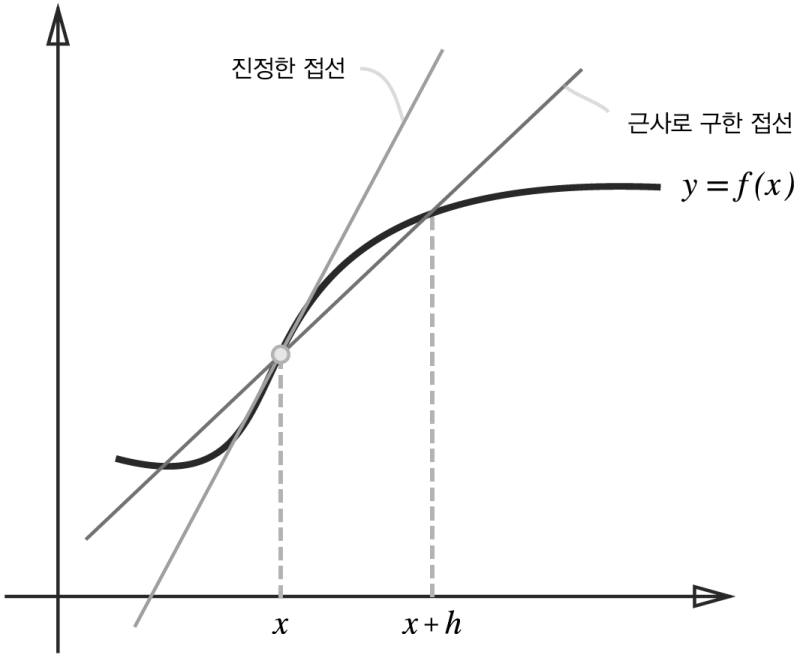
- 두 가지 문제점
  - 반올림 오차(rounding error)
  - 함수의 차분과 기울기 간의 간극

수치 미분

- ✓ **반올림 오차**
  - 원인: 너무 작은 값
  - 해결책: h의 값으로 1e-4 사용

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

- ✓ **차분과 기울기의 간극**
  - 원인: 전방 차분
  - 해결책: 중앙 차분



수치 미분

✓ 개선된 파이썬 수치 미분 코드

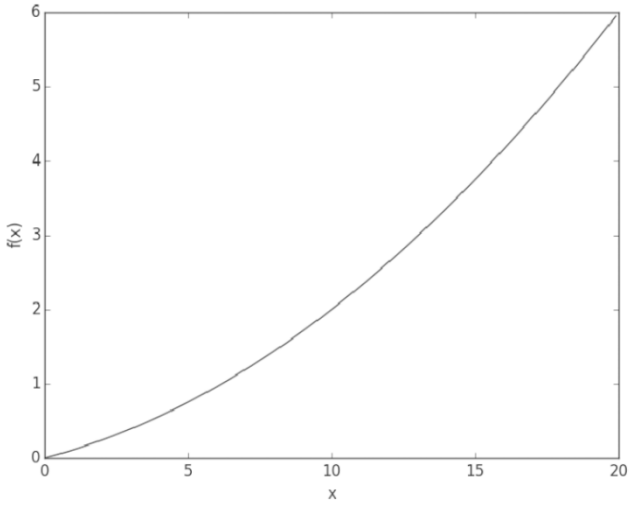
```
def numerical_diff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)
```

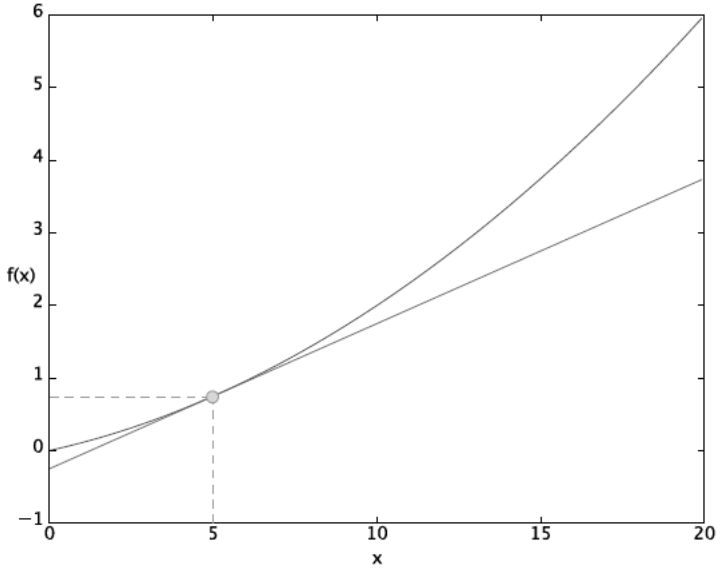
수치 미분

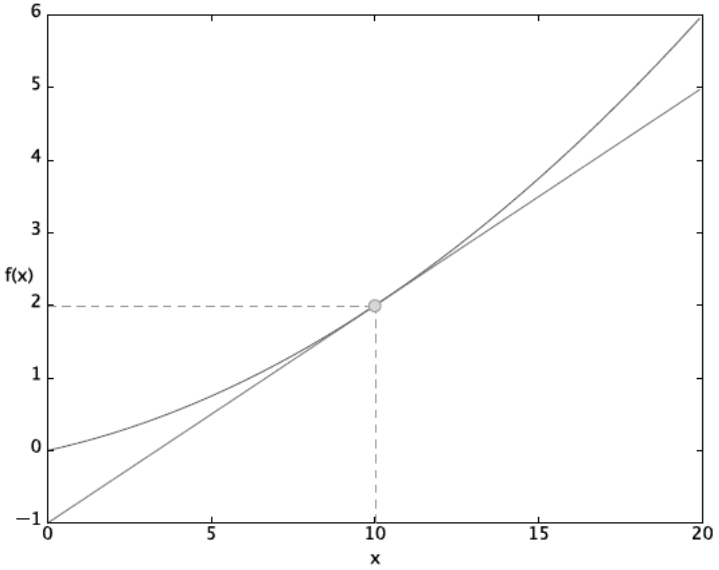
✓ **수치 미분의 예**

- gradient_1d.py 참고

$$y = 0.01x^2 + 0.1x$$





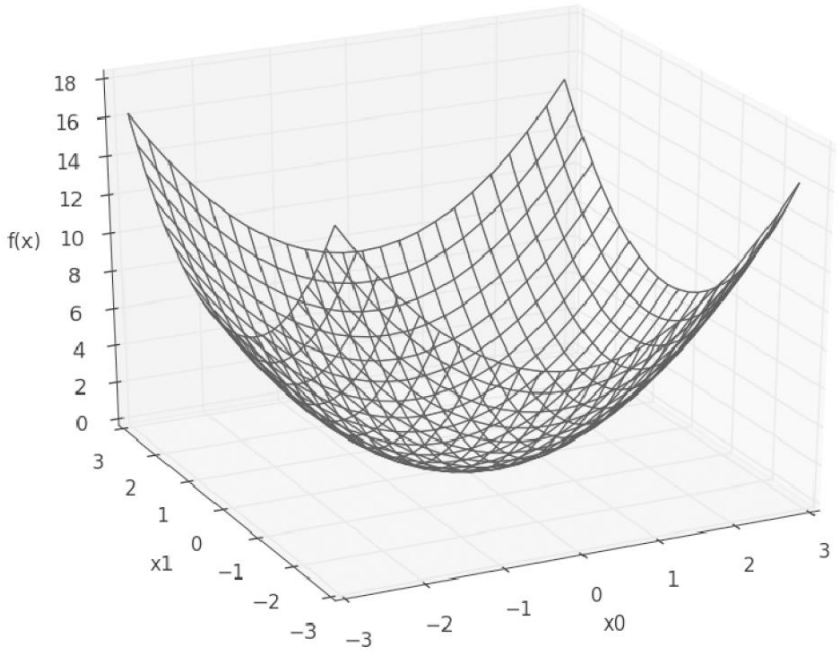


편미분

✓ **변수가 2개 이상**

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

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



기울기

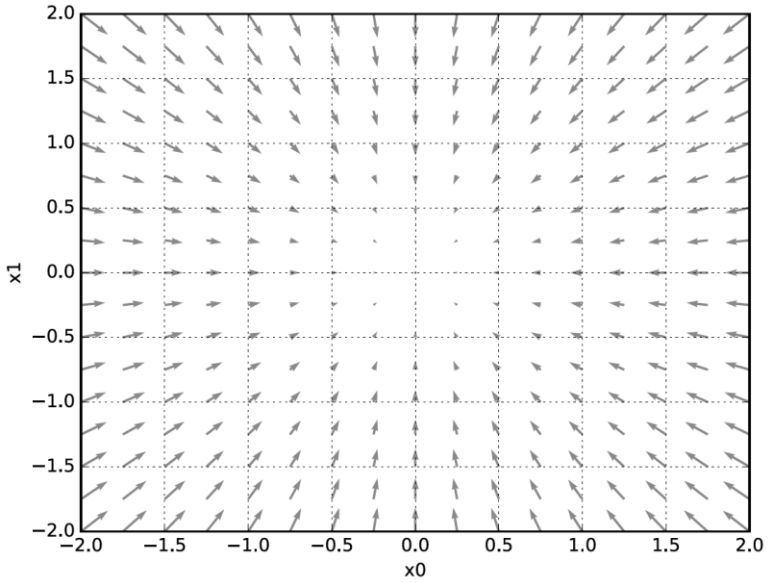
✓ 기울기

- 모든 변수의 편미분을 벡터로 정리한 것

$$\left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}\right)$$

```
def numerical_gradient(f, x):
    ...
    return grad
```

- !기울기가 가리키는 쪽은
  각 장소에서 함수의 출력 값을
  가장 크게 줄이는 방향!



경사법(경사 하강법)

- ✓ **기울기를 이용, 함수의 최솟값을 찾으려는 방법**
  - 기울기가 0인 지점
    - 최솟값, 극솟값, 안장점(saddle point)

- ✓ **경사 하강법(Gradient Descent)**
  - 학습률

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

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

경사법(경사 하강법)

✓ **파이썬 코드**

```
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
```

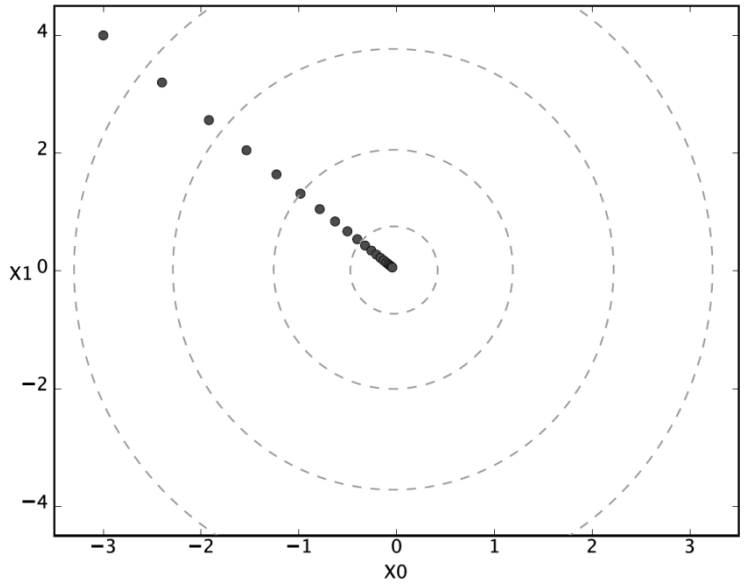
경사 하강법 예제

✓ 경사법으로 $f(x_0, x_1) = {x_0}^2 + {x_1}^2$의 최솟값을 구하라

```
>>> 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)
```

✓ 학습률이 너무 큰 예

✓ 학습률이 너무 작은 예



하이퍼파라미터

- ✓ **Hyper parameter, 초매개변수**
  - 학습률과 같은 파라미터
  - 가중치, 바이어스와는 다른 성질
  - 사람이 직접 설정해야 하는 파라미터
  - 여러 후보 값 중에서 시험을 통해 가장 적합한 값을 찾는 과정 필요

신경망에서의 기울기

- ✓ 가중치 매개변수에 대한 손실 함수의 기울기

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

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

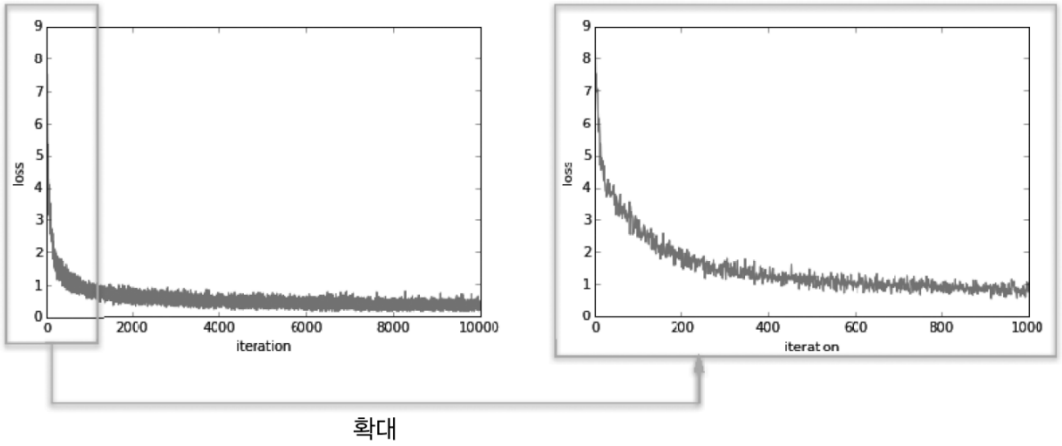
- ✓ W와 $\frac{\partial L}{\partial W}$ 의 형상이 같다
- ✓ Ch04/gradient_simplenet.py의 simpleNet 클래스
- ✓ 손실 함수의 기울기에 대한 의미를 이해하자.

학습 알고리즘 구현하기

- ✓ **신경망 학습 절차: SGD 사용**
  - 전제
    - 신경망에는 적응 가능한 가중치와 편향(바이어스)이 있고, 이 가중치와 바이어스를 훈련 데이터에 적응하도록 조정하는 과정을 '학습'이라 한다. 신경망 학습은 다음과 같이 4단계로 수행한다.
  - 1단계 - 미니 배치
    - 훈련 데이터 중 일부를 **무작위**로 가져온다.
    - 이렇게 선별된 데이터를 미니 배치라 하며, 이 미니 배치의 손실 함수 값을 줄이는 것이 목표다.
  - 2단계 - 기울기 산출
    - 미니 배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기를 구한다.
    - 기울기는 손실 함수의 값을 가장 작게 하는 방향을 제시한다.
  - 3단계 - 매개변수 갱신
    - 가중치 매개변수를 기울기 방향으로 아주 조금 갱신한다.
  - 4단계
    - 1~3 단계 반복

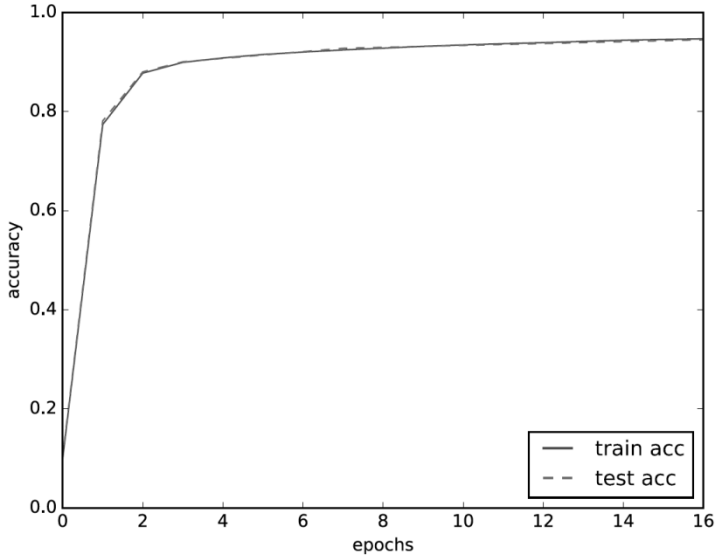
학습 알고리즘 구현하기

- ✓ **2층 신경망에 대해 MNIST 데이터셋을 학습하도록 구현**
  - 2층 신경망 클래스 구현하기
    - ch04/two_layer_net.py 코드 참조
  - 미니 배치 학습 구현하기
    - ch04/train_neuralnet.py 코드 참조
    - 매번 60,000개의 훈련 데이터에서 임의로 100개의 데이터를 추려냄
    - 100개의 데이터에 대해 SGD를 수행하여 매개변수 갱신



학습 알고리즘 구현하기

- ✓ **2층 신경망에 대해 MNIST 데이터셋을 학습하도록 구현**
  - 시험 데이터로 평가하기
    - 훈련 데이터의 손실 함수 값이 작아진다. ➔ 학습이 잘 되고 있다
    - 오버피팅 여부 확인
    - 에폭(epoch)별로 훈련 데이터와 시험 데이터에 대한 정확도 기록
      - 1 epoch: 학습 데이터를 전부 소진했을 때의 학습 데이터 수



정리

- ✓ 기계학습에서 사용하는 데이터셋은 훈련 데이터와 시험 데이터로 나눠 사용
- ✓ 훈련 데이터로 학습한 모델의 범용 능력을 시험 데이터로 평가
- ✓ 신경망 학습은 손실 함수를 지표로, 손실 함수의 값이 작아지는 방향으로 가중치 매개변수를 갱신
- ✓ 가중치 매개변수를 갱신할 때는 가중치 매개변수의 기울기를 이용, 기울어진 방향으로 가중치의 값을 갱신하는 작업을 반복
- ✓ 아주 작은 값을 주었을 때의 차분으로 미분하는 것을 수치 미분이라 함
- ✓ 수치 미분을 이용해 가중치 매개변수의 기울기를 구할 수 있다.
- ✓ 수치 미분을 이용한 계산에는 시간이 걸리지만, 그 구현은 간단하다. (다소 복잡한) 오차역전파법은 기울기를 고속으로 구할 수 있다.