Neural Network Learning

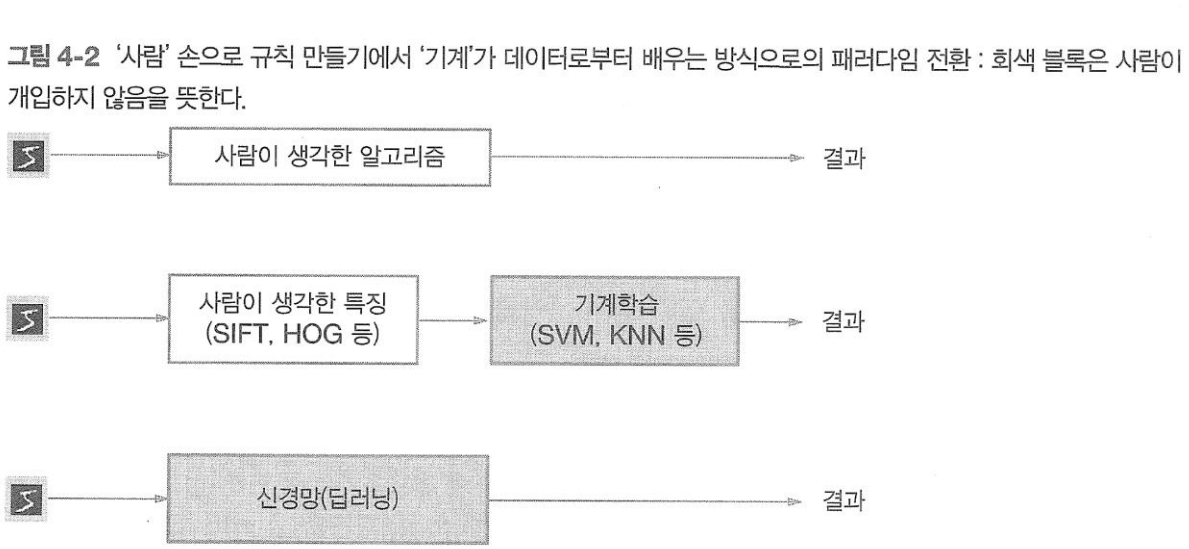
Computational Linguistics @ Seoul National University

DL from Scratch

By Hyopil Shin

Learning

- Training data로부터 가중치(weight) 매개변수(parameter)의 최적값을 자동으로 획득하는 것
- 데이터 주도 학습
- Training Data
- Test Data
  - 범용능력을 평가하기 위해
- Overfitting
  - 한 데이터셋에삼 지나치게 최적화된 상태



Revisit to Cost Function

- 모두를 위한 머신러닝 Lecture03- how to minimize cost

What *cost(W)* looks like?

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

- W=1, cost(W)=0

$$\frac{1}{3}((1*1-1)^2 + (1*2-2)^2 + (1*3-3)^2)$$

- W=0, cost(W)=4.67

$$\frac{1}{3}((0*1-1)^2 + (0*2-2)^2 + (0*3-3)^2)$$

- W=2, cost(W)=?

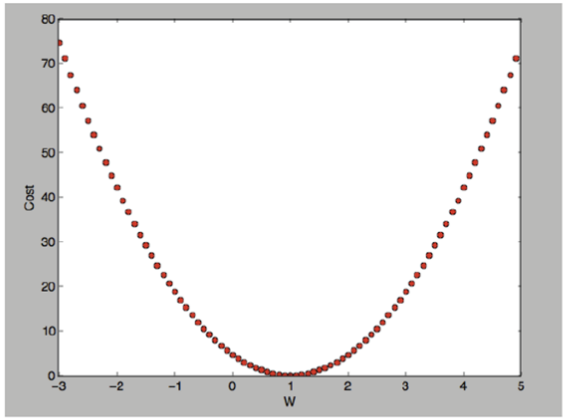
- W=1, cost(W)=0
- W=0, cost(W)=4.67
- W=2, cost(W)=4.67

Revisit to Cost Function

- How to minimize cost?

What *cost(W)* looks like?

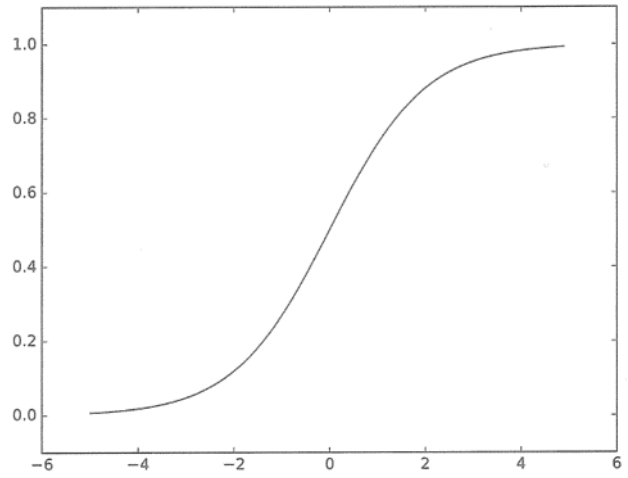
$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$



Revisit to Logistic Regression

- Binary classification
- Logistic Hypothesis : z = Wx + b, g(z) -> 0 or 1

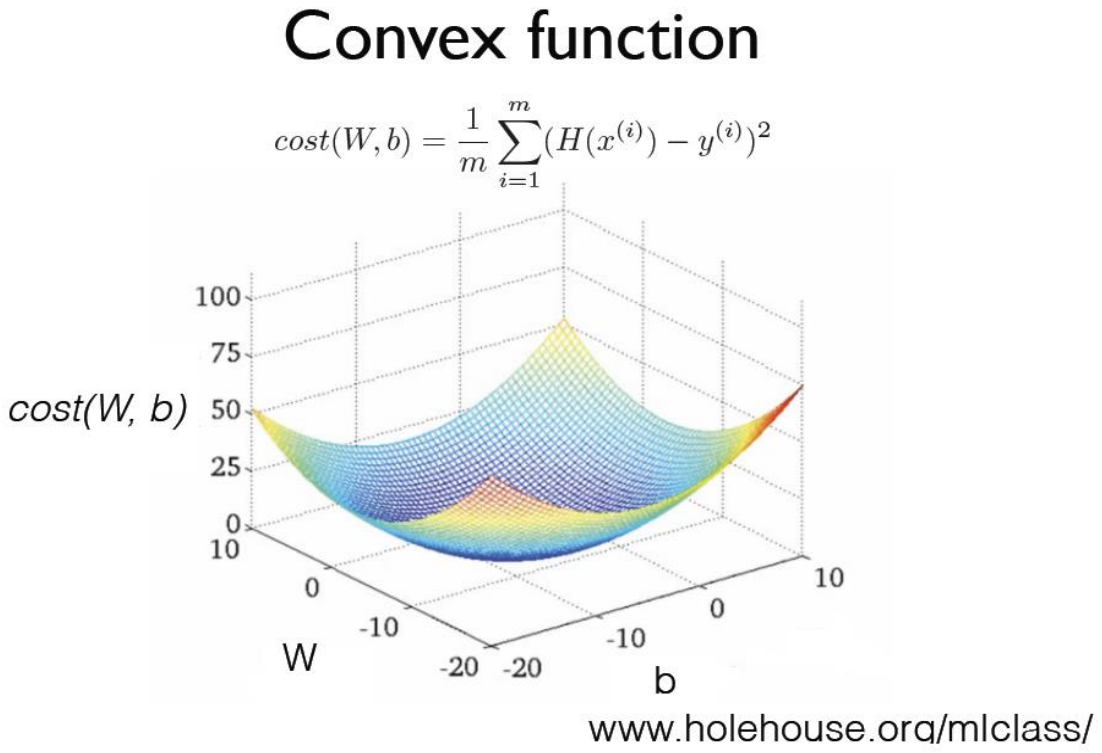
그림 3-7 시그모이드 함수의 그래프*



$$g(z) = \frac{1}{(1+e^{-z})}$$

$$H(X) = \frac{1}{1+e^{-(W^T X)}}$$

Revisit to Logistic Regression



www.holehouse.org/mlclass/

손실함수(Loss Function)

- 신경망 학습에서는 현재의 상태를 '하나의 지표'로 표현
  - 이 지표를 가장 좋게 만들어주는 가중치 매개변수의 값을 탐색
- 현재의 신경망이 훈련데이터를 얼마나 잘 처리하지 못하느냐
- 평균제곱오차 (Mean Squared Error, MSE)

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

```
>>> 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)
```

```
>>> # 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]
>>> mean_squared_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]
>>> mean_squared_error(np.array(y), np.array(t))
0.5975000000000003
```

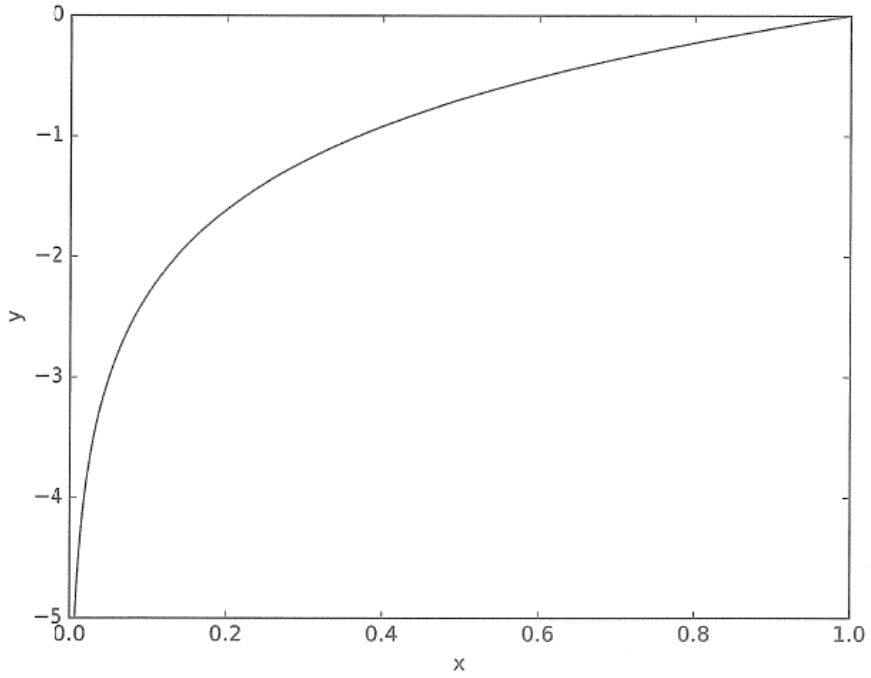
Cross Entropy Error

- Cross entropy
  - T_k가 1일 때의 y_k의 자연로그 계산

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

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

그림 4-3 자연로그 $y = \log x$의 그래프




```
>>> 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
```

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

Mini-batch Learning

- 많은 데이터에서 손실 함수를 모두 계산하는 것은 시간이 오래 걸림
- 훈련 데이터에서 일정수를 무작위로 뽑아 학습 -> mini-batch

```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=True, one_hot_label=True)

print(x_train.shape) # (60000, 784)
print(t_train.shape) # (60000, 10)
```

```
train_size = x_train.shape[0]
batch_size = 10
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
t_batch = t_train[batch_mask]
```

```
>>> np.random.choice(60000, 10)
array([ 8013, 14666, 58210, 23832, 52091, 10153,  8107, 19410, 27260, 21411])
```

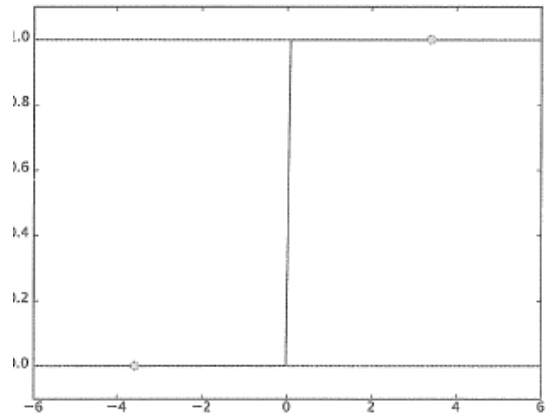
Cross Entropy Error with Mini Batch

```
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
```

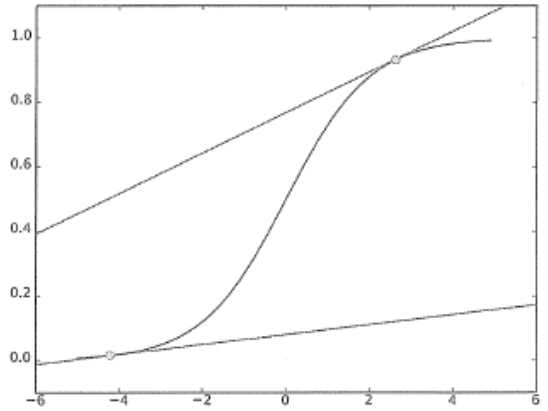
Why Loss Function?

- 높은 Accuarcy를 끌어내는 매개변수 값을 찾는 것
- 최적의 매개변수(가중치와 편향) -> 손실값을 가능한 한 작게 하는 매개변수를 찾는 것
  - 미분을 사용
  - 가중치 매개 변수의 값을 아주 조금 변화시켰을 때 손실 함수가 어떻게 변화하는지
  - 미분이 음수이면 가중치 매개변수를 양의 방향으로 변화시켜 손실 함수의 값을 줄임, 양수면 반대로 음의 방향으로 변화시켜 손실함수의 값을 줄임
  - 미분 값이 0이면 가중치 매개변수를 어느 쪽으로 움직여도 손실 함수의 값은 달라지지 않음. -> 가중치 매개변수의 갱신 중단
- 정확도를 지표로 삼지 않는 것은 미분값이 대부분이 0이 되어 매개변수를 갱신할 수 없기 때문
- 정확도는 매개변수의 미묘한 변화에는 거의 반응을 보이지 않고 반응이 있더라도 그 값이 불연속적으로 갑자기 변화
  - 계단함수보다 Sigmoid 함수를 사용하는 이유
  - 계단함수의 미분은 대부분의 장소에서 0. 따라서 매개변수의 작은 변화가 주는 파장을 계단함수가 말살하여 손실함수에는 아무런 변화가 나타나지 않기 때문
  - Sigmoid 함수의 미분은 어느 장소라도 0이 되지 않음

계단 함수



시그모이드 함수



Differentiation

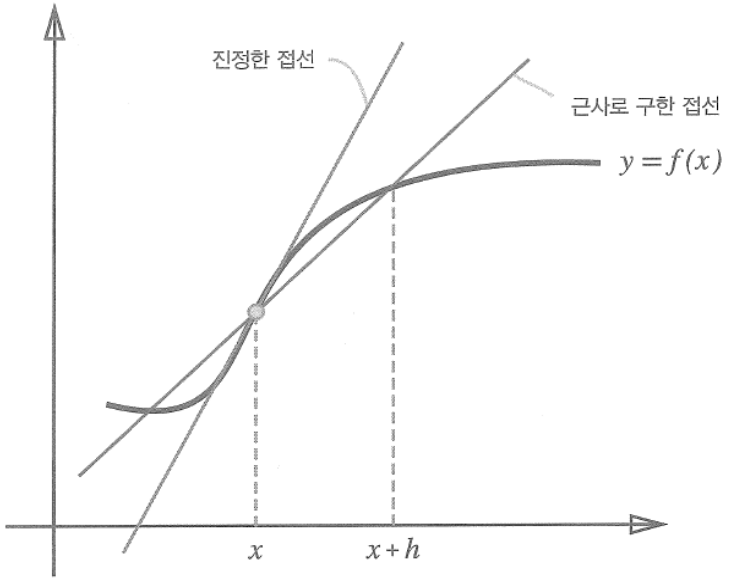
- '특정 순간'의 변화량
  - f(x)의 x에 대한 미분
  - x의 작은 변화가 함수 f(x)를 얼마나 변화시키느냐를 의미
- 수치미분(numerical differentiation)
  - 작은 차분으로 미분을 구하는 방법
    - 반올림 오차(rounding error): 작은값이 생략되어 최종 결과에 오차
    - 차분 오차 : 중심차분, 전방차분

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

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

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

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

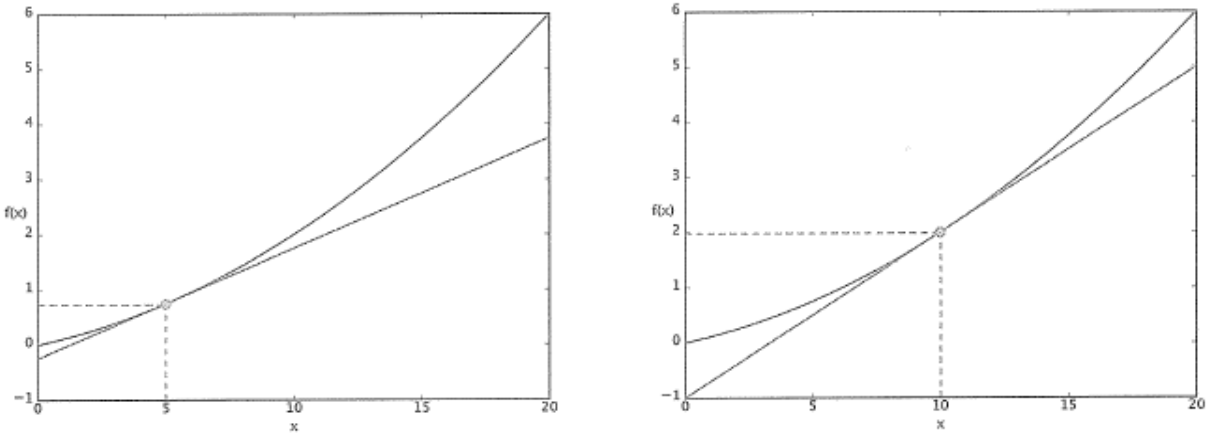


Differentiation

- Numerical differentiation
  - gradient_1d.py

- 해석적 미분(Analytic differentiation)
  - 오차를 포함하지 않는 진정한 미분
- $Y = 0{,}01x^2 + 0.1x$

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



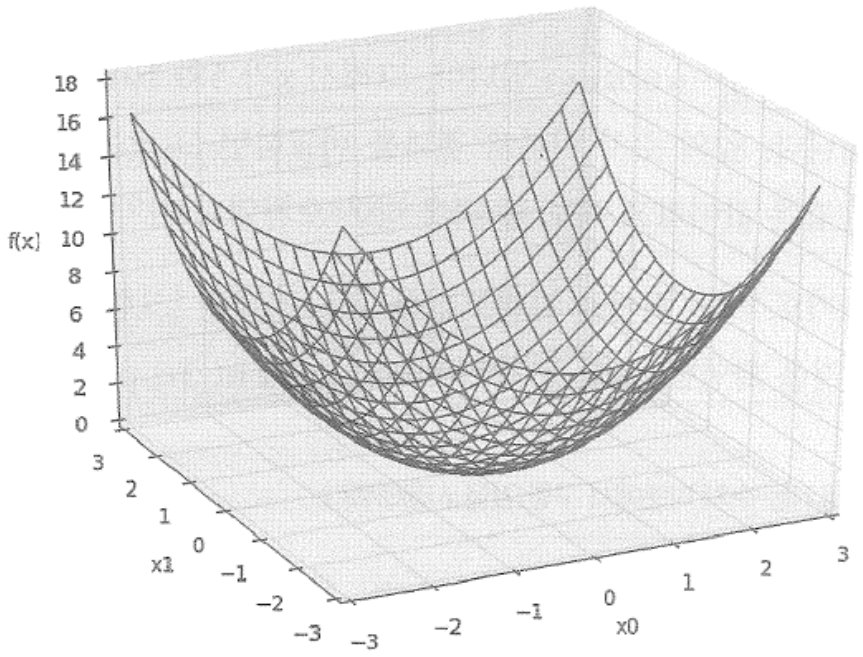
Partial Derivative

- 편미분 - 변수가 여럿인 함수에 대한 미분
  - 여러 변수 중 목표 변수 하나에 초점을 맞추고 다른 변수의 값은 고정
  - $$f(x_0, x_1) = x_0^2 + x_1^2$$

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

$\frac{\partial f}{\partial x_0}$ 와 $\frac{\partial f}{\partial x_1}$

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



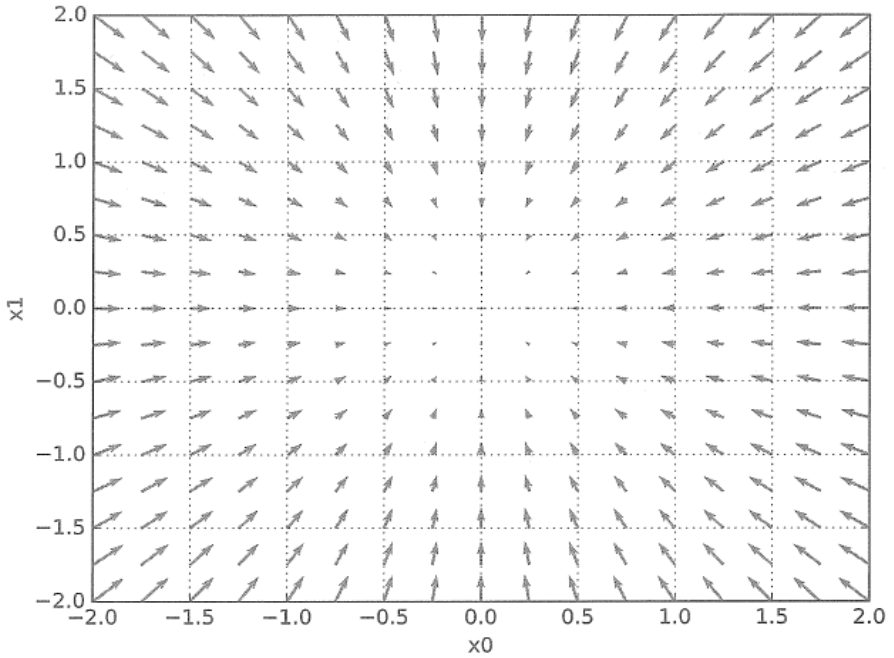

```
>>> def function_tmp1(x0):
...     return x0*x0 + 4.0**2.0
...
>>> numerical_diff(function_tmp1, 3.0)
6.00000000000378
```

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

Gradient

- 모든 변수(x_0, x_1)의 편미분을 동시에 계산($\left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}\right)$)
- 기울기(gradient)-모든 변수의 편미분을 벡터로 정리한 것
  - gradient_2d.py
  - 기울기는 함수의 '가장 낮은 장소(최소값)'을 가리킴
  - 가장 낮은 곳에서 멀어질수록 화살표 크기가 커짐
  - 기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 줄이는 방향

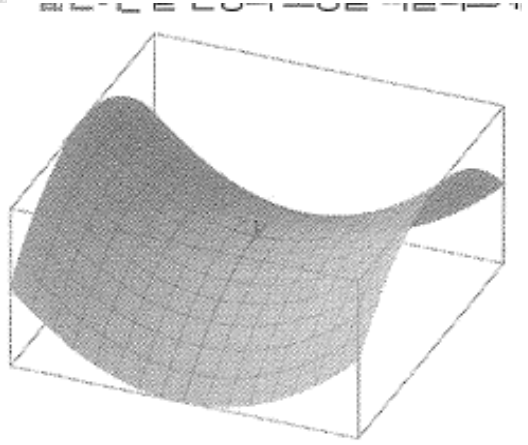
그림 4-9 $f(x_0, x_1) = x_0^2 + x_1^2$의 기울기



Gradient Descent Method

- 학습단계에서 최적의 매개변수 찾기 -> loss function이 최소가 될 때의 매개변수 값

WARNING_ 함수가 극솟값, 최솟값, 또 **안장점**saddle point이 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, 플래토이라 하는, 학습이 진행되지 않는 정체기에 빠질 수 있습니다.

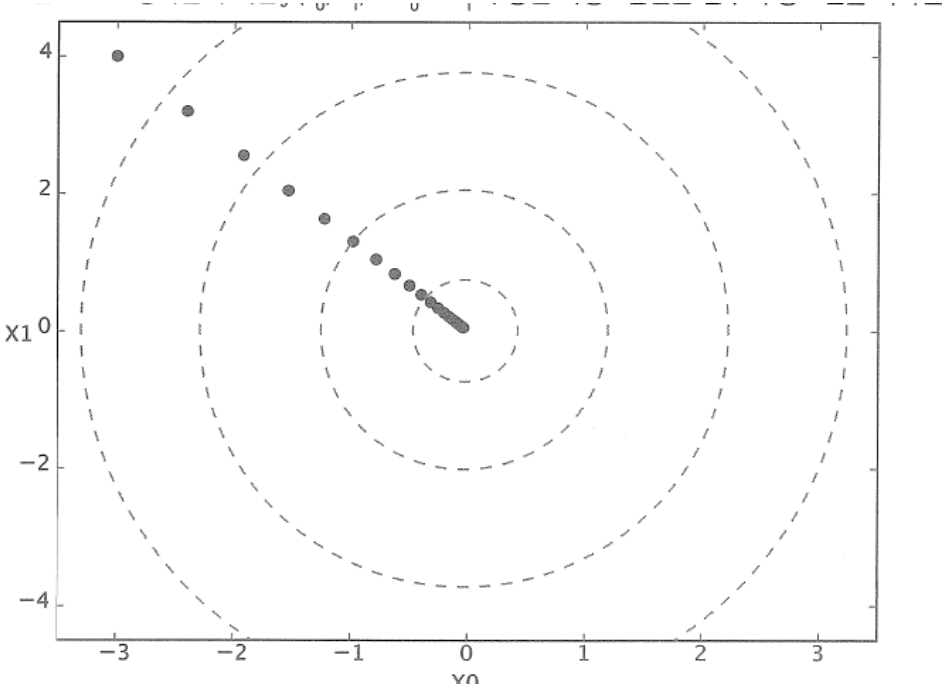


Gradient Descent Method

- 경사법
  - 현 위치에서 기울어진 방향으로 일정 거리만큼 이동. 다시 이동한 곳에서도 마찬가지로 기울기를 구하고, 또 그 기울어진 방향으로 나아가는 일을 반복
  - 경사하강법(Gradient Descent Method)
- 학습률(learning rate)
  - 특정의 값으로 미리 정함 (hyper parameter)-0.01, 0.001
  - 이 값이 너무 크거나 작으면 최적화 되기 어려움
  - 이 학습률을 변경하면서 올바르게 학습하고 있는지 확인하면서 진행
  - 학습률이 너무 크면 큰 값으로 발산해 버리고, 너무 작으면 거의 갱신되지 않음 -> 학습률을 적절히 설정하는 것이 중요
- 04-gradient_method.py

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

$$x_1 = x1 - \eta \frac{\partial f}{\partial x_1}$$



Gradient Descent Method

- 신경망 학습에서의 기울기 -> 가중치 매개변수에 관한 손실 함수의 기울기
  - 예) 형상이 2x3, 가중치 W, 손실 함수 L인 신경망
  - 이 경우 경사는 $\frac{\partial L}{\partial \mathbf{W}}$

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

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

  - $\frac{\partial L}{\partial \mathbf{W}}$ 는 각 원소 각각의 대한 편미분
  - $\frac{\partial L}{\partial w_{11}}$ 은 w_{11}을 조금 변경했을 때 손실함수 L이 얼마나 변화하느냐를 나타냄
  - $\frac{\partial L}{\partial \mathbf{W}}$ 의 형상은 W와 같음
  - 04-gradient_simplenet .py

Implementing Learning Algorithm

- 신경망 학습절차
  - 전제: 신경망에는 적용 가능한 가중치와 편향이 있고, 이 가중치와 편향을 훈련 데이터에 적응하도록 조정하는 과정이 학습
  - 1단계: 미니배치
    - 훈련 데이터 중 일부를 무작위로 가져옵니다. 이를 미니배치라고 하고, 미니배치의 손실 함수 값을 줄이는 것을 목표로 함
  - 2단계: 기울기 산출
    - 미니배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기를 구함. 기울기는 손실함수의 값을 가장 작게 하는 방향을 제시
  - 3단계: 매개변수 갱신
    - 가중치 매개변수를 기울기 방향으로 아주 조금 갱신
  - 4단계: 반복
    - 1~3단계를 반복
- 확률적 경사하강법(Stochastic Gradient Descent, SGD): 확률적으로 무작위로 골라낸 데이터에 대해 수행하는 경사 하강법
- 04-two_layer_net.py

Implementing Learning Algorithm

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

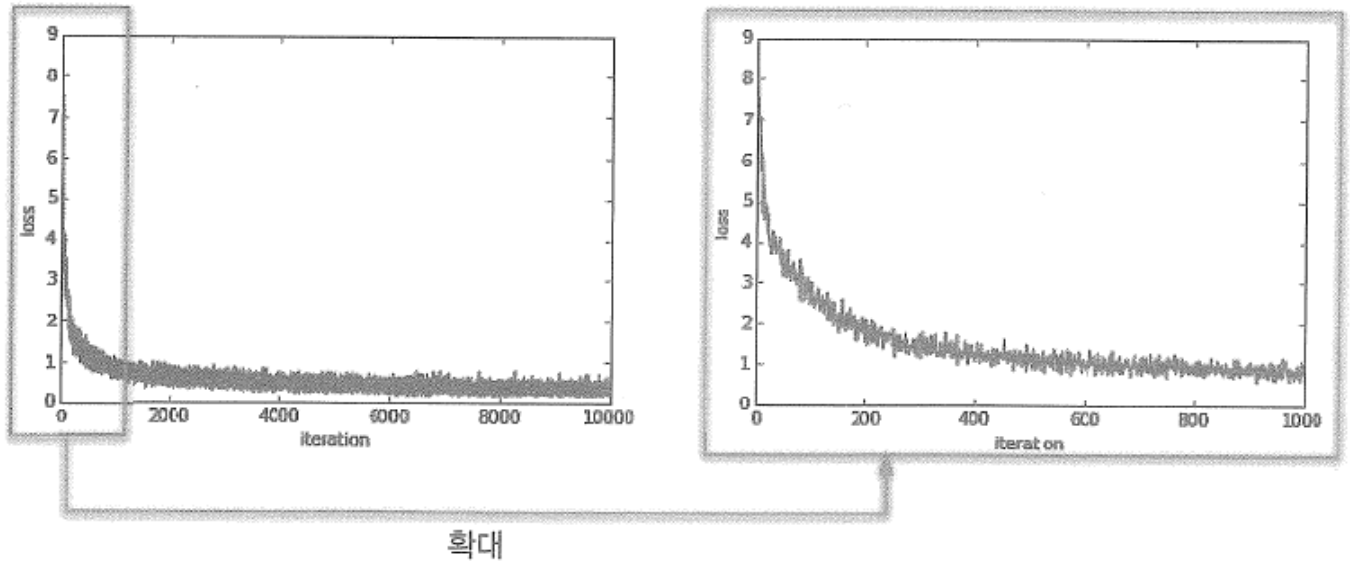
표 4-2 TwoLayerNet 클래스의 메서드

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...

Mini-batch training

- 04-train_neuralnet.py
- 그림 4-11: 학습횟수가 늘어가면서 손실함수의 값이 줄어듬.
  - 학습이 잘되고 있으며, 신경망의 가중치 매개변수가 서서히 데이터에 적응하고 있음을 의미

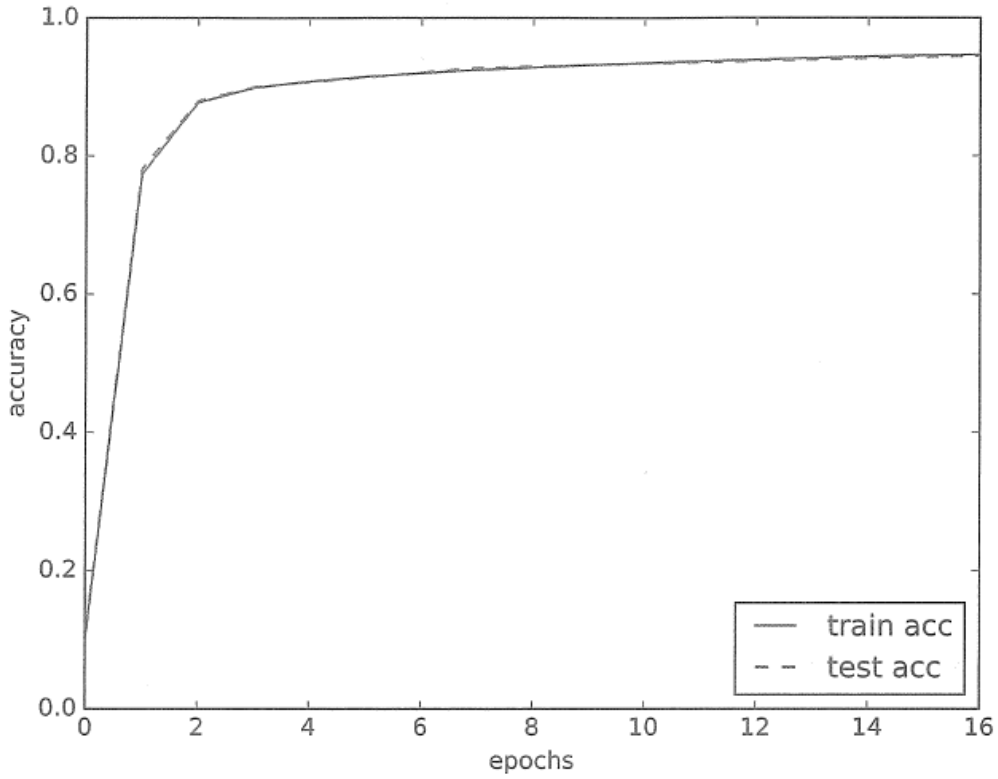
그림 4-11 손실 함수의 추이 : 왼쪽은 10,000회 반복까지의 추이, 오른쪽은 1,000회 반복까지의 추이



시험 데이터로 평가하기

- 학습이 잘 이루어졌는지 평가
- 그림 4-12
  - 학습이 진행될 수록 훈련데이터와 시험데이터를 사용하고 평가한 정확도가 증가
  - 두 선이 거의 겹쳐 있기 때문에 두 데이터 사이의 정확도에는 차이가 없다 -> overfitting이 일어나지 않음

그림 4-12 훈련 데이터와 시험 데이터에 대한 정확도 추이



Round Up

- 기계학습에서 사용하는 데이터셋은 훈련 데이터와 시험 데이터로 나눠 사용
- 훈련 데이터에서 학습한 모델의 범용 능력을 시험 데이터로 평가
- 신경망 학습은 손실 함수를 지표로, 손실 함수의 값이 작아지는 방향으로 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
- 수치미분을 이용한 계산에는 시간이 걸리지만 구현은 간단
  - Backpropagation- 기울기를 빠르게 구할 수 있게 함