##0302

2021년 3월 1일 월요일 오후 4:52

■ 4장. 신경망

□4.1 데이터에서 학습

4.1.1 데이터 주도 학습

raw_data	→	사람이 생각한 알고리즘 (한땀한땀 구현)	→	→	→	결과
raw_data	→	사람이 생각한 특징 (SIFT, HOG 등)	→	기계학습 (SVM, KNN 등)	→	결과
raw_data	→	**신경망 (딥러닝)** **(Black box)**	→	→	→	결과

raw_data : 손글씨 데이터 (예시)

딥러닝 : 종단간 기계학습(end-to-end machine learning)

신경망(딥러닝) 은 이미지(raw-data)를 있는 그대로 학습한다.

4.1.2 훈련 데이터와 시험 데이터

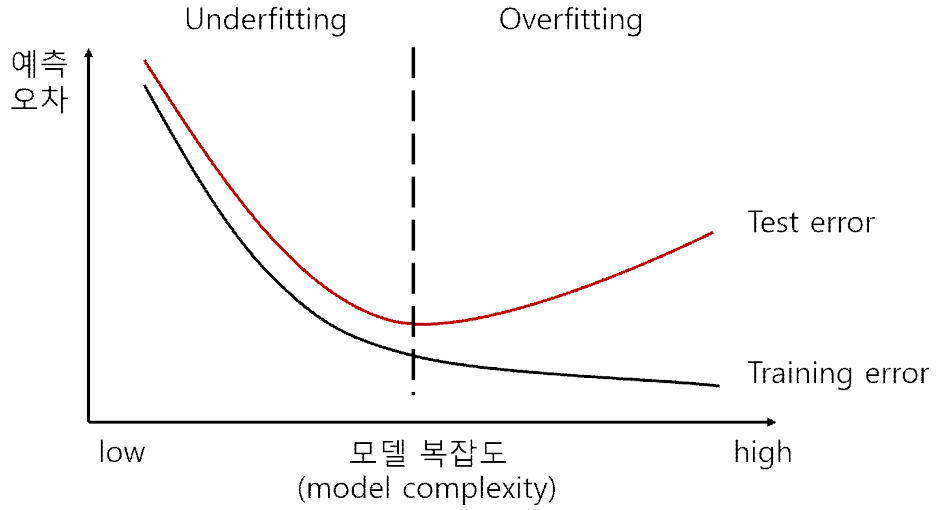
오버피팅과 언더피팅 url : https://nittaku.tistory.com/289

overfitting : high bias
underfitting : high varance

bias : 실제값에서 멀어진 척도 ---> 모델의 예측값이 전반적으로 좋지 않음

variance : 예측된 값들의 분산(서로 떨어진 정도) : trainset ~ testset 의 accuraccy 의 차이

[1] 다항회귀의 모델 예시

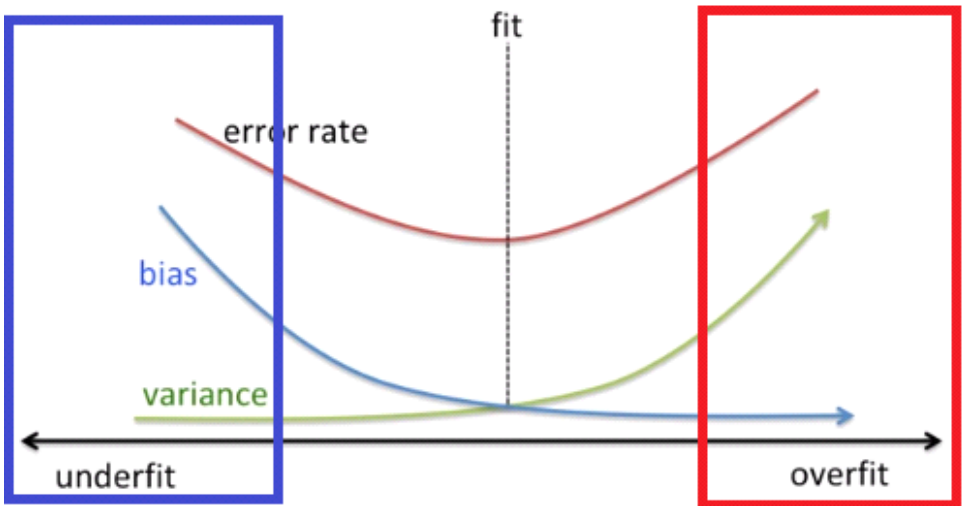


다항회귀에서 모델의 복잡도 증가 예시

Y = b0 + b1x
Y = b0 + b1x + b2x^2
Y = b0 + b1x + b2x^2 + b3x^3
Y = b0 + b1x + b2x^2 + b3x^3 + b4x^4

Y = b0 + b1x + b2x^2 + b3x^3 + b4x^4 + ...

[2] bias 와 variance, error rate(=accuraccy)



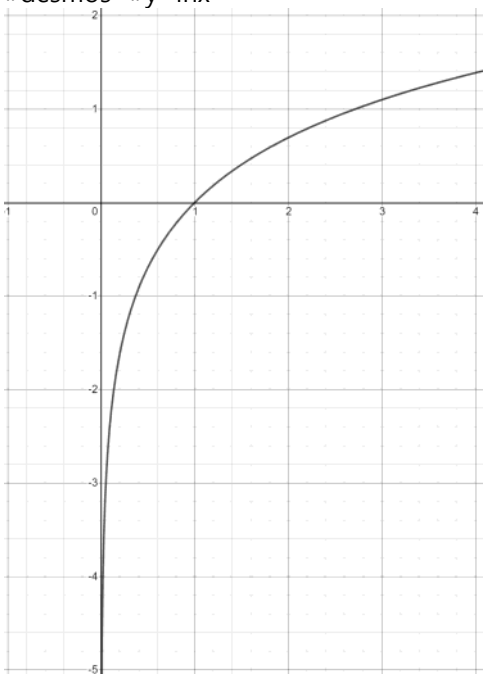
□4.2 손실함수

4.2.1 오차제곱합(SSE : Sum of Squares for Error)

E	**=**	**$1/2 \sum_k (y_k - t_k)^2$**
code	----->	def sum_squares_error(y, t): return 0.5 *np.sum((y-t)**2)

4.2.2 교차 엔트로피 오차(cross_entropy)

#desmos #y=lnx



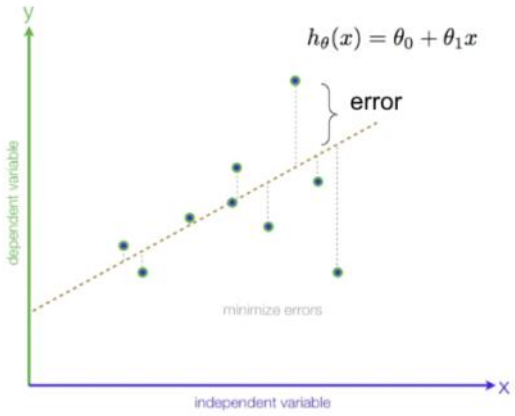
E	**=**	**$-\sum_k t_k \log y_k$**
code	----->	def cross_entropy(y, t): delta = 1e-7 return -np.sum(t *np.log(y + delta)) #np.log(0) ---> - infinite

손실함수 : loss function
손실함수는 비용함수(cost function) 으로도 불리고, 다양한 함수로 표현 가능하다
MSE, MAE, 등의 형태가 대표적
MSE : Mean of Squared Errors

[3] 선형회귀에서의 cost function(=loss function) 예시

What is Cost Function?

The primary set-up for learning neural networks is to define a cost function (also known as a loss function) that measures how well the network predicts outputs on the test set. The goal is to then find a set of weights and biases that minimizes the cost. One common function that is often used is the mean squared error, which measures the difference between the actual value of *y* and the estimated value of *y* (the prediction). The equation of the below regression line is hθ(*x*) = θ + θ1*x*, which has only two parameters: weight (θ1)and bias (θ0).



Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Goal:

$$\underset{\theta_0, \theta_1}{minimize} J(\theta_0, \theta_1)$$

뒤의 절에서 등장하는 gradient descent 는 위 loss function 혹은 cost function 을 최소화하는 방법이다.
이 때 미분의 개념이 사용된다. (다차원 ---> 편미분)

4.2.3 미니배치 학습(mini-batch)

ex) 60,000 장의 훈련 데이터 중 100장을 무작위로 뽑아 100장만 학습하는 형태

Q. 그렇다면 왜 미니배치 학습을 사용할까?
A. 계산비용(calculation cost)을 절약하기 위해서
A2. 미니배치의 예시 중 텔레비전 시청률도 모든 시청자를 전수조사하는 것이 아닌 표본을 선택하여 계산

4.2.4 (배치용) 교차 엔트로피 오차 구현하기

p.118 의 코드 참고 : 위에서 정의된 손실함수에 미니배치를 적용(여러개를 동시에 구하는 코드)

코드실습은 책의 설명을 천천히 따라서 읽으면 되고, 아래에서는 numpy 의 mathod 들에 대해 간단한 실습

1) np.arnage()

```
import numpy as np
np.arange(5)  #array([0, 1, 2, 3, 4])
np.arange(10)  #array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print( np.arange(5), 7 )  #[0 1 2 3 4] 7
```

2) np.random.randn, .shape, .ndim

```
import numpy as np
y = np.random.randn(5, 2)  #Gaussian 정규분포를 따르는 난수를 생성, 5 by 2 ndarray 로 생성함
print(y.shape)  #(5,2)  #ndarray 의 형태 : 5 by 2 행렬을 의미
print(y.ndim)  #2  #ndarray 의 차원을 표기

y2 = np.random.randn(7)  #1차원 행렬 return
print(y2.shape)  #(7,)
print(y2.ndim)  #1
```

4.2.5 손실 함수 정의의 의의

손실함수는 '미분' 과 밀접한 관련이 있으며,

최적의 매개변수(가중치와 편향)를 탐색시 손실함수의 값을 가능한 작게 하는 매개변수 값을 찾는다.

이 때 매개변수의 미분(기울기)를 계산하고 이 값을 단서로 매개변수의 값을 서서히 갱신하는 과정을 반복한다.

미분의 성질에 대해 자세히 다루겠지만 미분 값이 0이면 가중치 매개변수를 움직여도 손실함수의 값은 변경되지 않는다.

즉, 해당 지점에서 가중치 매개변수의 갱신이 멈추는 것이다. ---> gradient descent(경사하강법) 과 연관

□4.5 학습 알고리즘 구현하기

신경망 학습의 절차 p.136

1) 미니배치 ---> 확률적 경사 하강법(mini batch ---> 표본을 뽑는 것이기 때문)

2) 기울기 산출

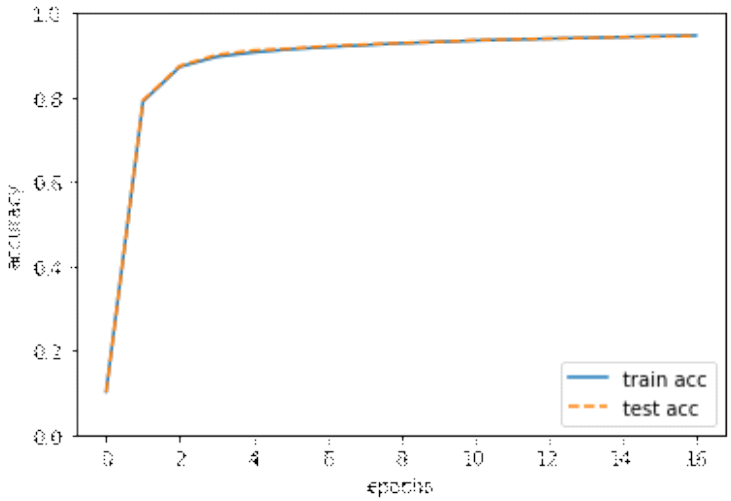
3) 매개변수 갱신

4) 1~3 반복(기울기가 최저값에 도달했다고 판단하여 갱신 중지할때까지)

에폭(단위)

1 epoch = 학습에서 훈련 데이터를 모두 소진했을 때의 횟수

code[2] 실행하면 도출되는 코드



epochs 가 진행될수록(학습이 진행될수록) 훈련데이터와 테스트 데이터의 정확도가 증가함을 알 수 있다.
또한 이번 학습에서는 두 테스트 간의 varince 가 관찰되지 않아 overfitting 이 일어나지 않음을 알 수 있다.

□4.6 정리

수치 미분을 이용한 계산에는 시간(calculation cost)이 걸리지만 구현이 간단하다.
5장의 '오차역전파법' 에서는 기울기를 고속으로 구현할 수 있다. (단, 구현은 복잡하다)

code[1]

two_layer_net.py

```
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
from common.functions import *
from common.gradient import numerical_gradient


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

    # 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
```

code[2]

rain_neuralnet.py

```
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

# 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))

# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

```
Downloading train-images-idx3-ubyte.gz ...
Done
Downloading train-labels-idx1-ubyte.gz ...
Done
Downloading t10k-images-idx3-ubyte.gz ...
Done
Downloading t10k-labels-idx1-ubyte.gz ...
Done
Converting train-images-idx3-ubyte.gz to NumPy Array ...
Done
Converting train-labels-idx1-ubyte.gz to NumPy Array ...
Done
Converting t10k-images-idx3-ubyte.gz to NumPy Array ...
Done
Converting t10k-labels-idx1-ubyte.gz to NumPy Array ...
Done
Creating pickle file ...
Done!
train acc, test acc | 0.10218333333333333, 0.101
train acc, test acc | 0.7904833333333333, 0.7932
train acc, test acc | 0.87255, 0.8749
train acc, test acc | 0.8965, 0.901
train acc, test acc | 0.90695, 0.9118
train acc, test acc | 0.9142333333333333, 0.916
train acc, test acc | 0.9194833333333333, 0.922
train acc, test acc | 0.9240333333333334, 0.9255
train acc, test acc | 0.92845, 0.9292
train acc, test acc | 0.9317166666666666, 0.9318
train acc, test acc | 0.935, 0.9359
train acc, test acc | 0.9371833333333334, 0.9377
train acc, test acc | 0.93935, 0.9395
train acc, test acc | 0.9411333333333334, 0.941
train acc, test acc | 0.94325, 0.9431
train acc, test acc | 0.94535, 0.9442
train acc, test acc | 0.9467166666666667, 0.946
```

+ 그래프 (위에 있음)

#####endLine=====