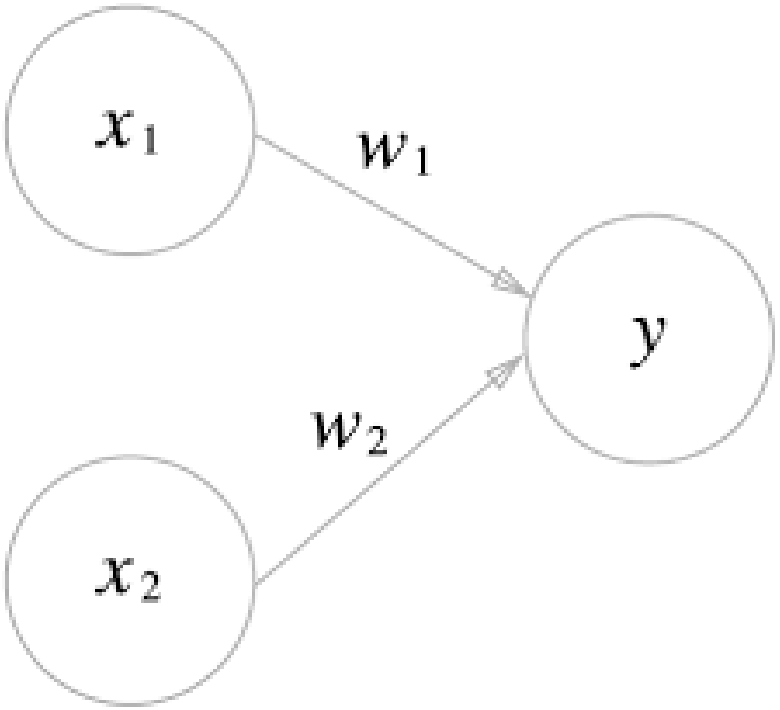
밑바닥부터 시작하는 딥러닝

20213093 정현우

CONTENTS

1. 퍼셉트론

퍼셉트론 소개, 한계, 개선

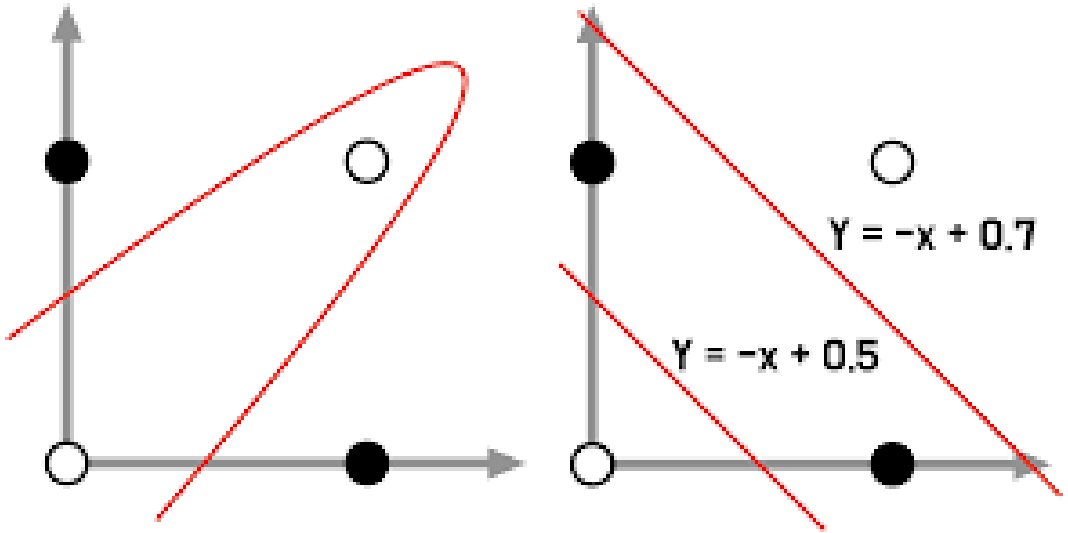



```python
def AND(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    tmp = x1 * w1 + x2 * w2
    if tmp <= theta:
        return 0
    elif tmp > theta:
        return 1
```

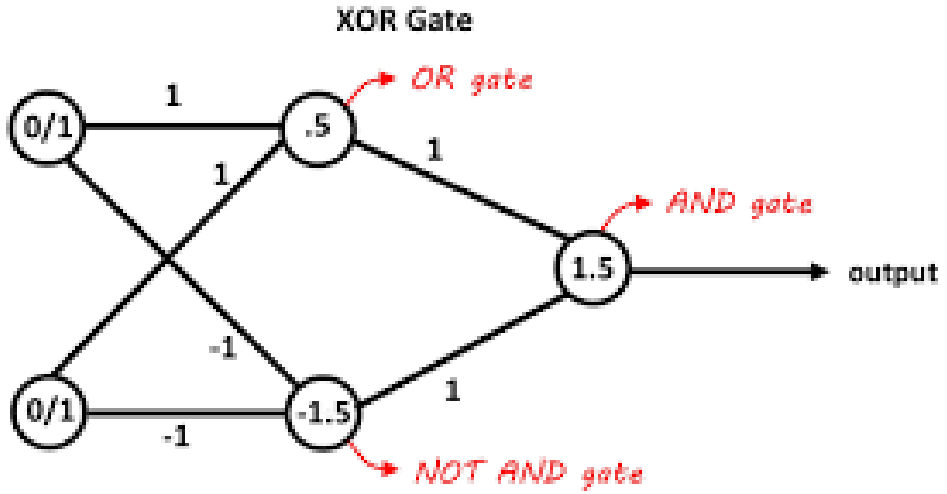
```python
def OR(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.3
    tmp = x1 * w1 + x2 * w2
    if tmp <= theta:
        return 0
    elif tmp > theta:
        return 1
```

1. 퍼셉트론

퍼셉트론 소개, 한계, 개선



퍼셉트론의 한계와 개선



1. 퍼셉트론

퍼셉트론 소개, 한계, 개선

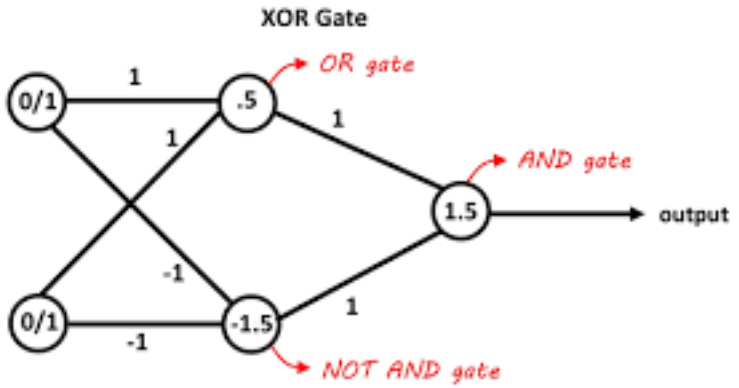
```python
def NAND(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    tmp = x1 * w1 + x2 * w2
    if tmp > theta:
        return 0
    elif tmp <= theta:
        return 1
```

XOR 구현

앞에서 본대로 OR와 NADN의 출력을 AND게이트의 입력으로 넣어주면 XOR 게이트가 된다.

```python
def XOR(x1, x2):
    tmp1 = OR(x1, x2)
    tmp2 = NAND(x1, x2)
    out = AND(tmp1, tmp2)
    return out

print(XOR(0, 0))
print(XOR(0, 1))
print(XOR(1, 0))
print(XOR(1, 1))
```

```
0
1
1
0
```



2. 신경망

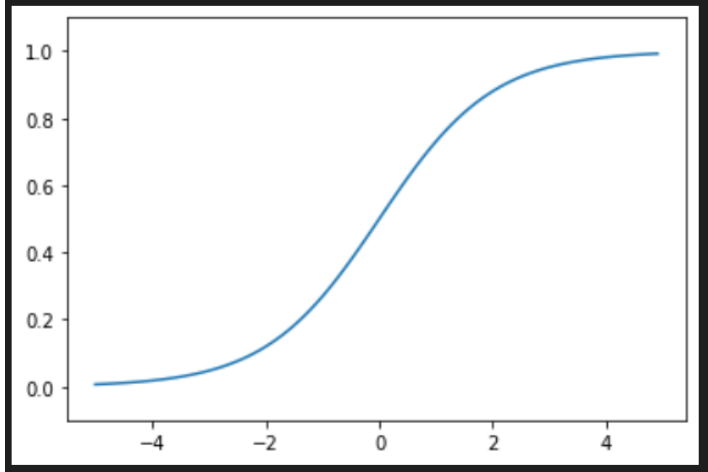
활성화 함수, 출력층, 구현

```
def sigmoid(x):
    return 1 / ( 1 + np.exp(-x))
```

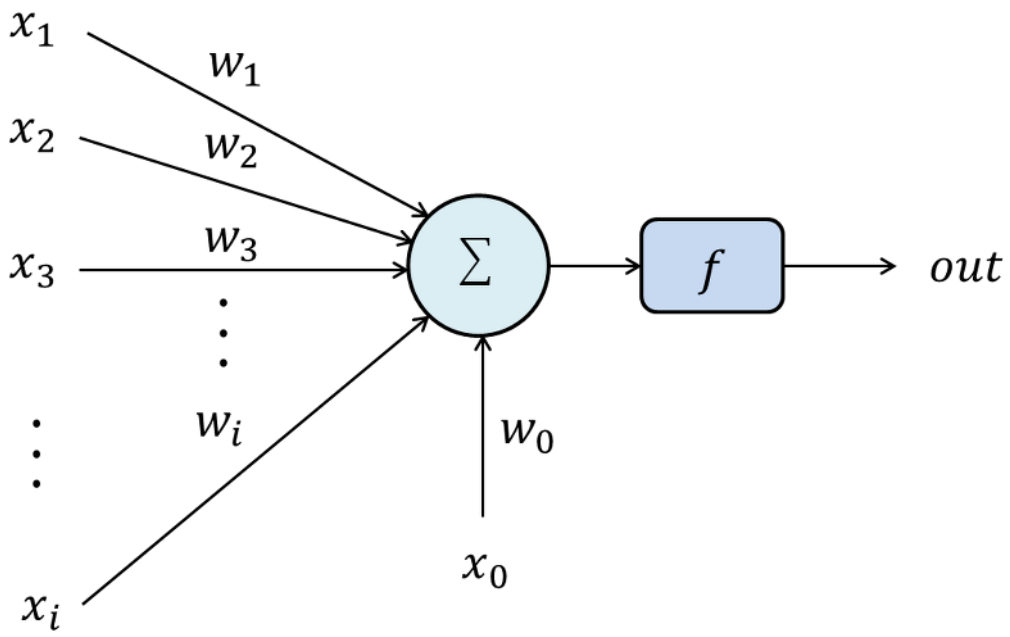
✓ 0.2s

```
x = np.arange(-5.0, 5.0, 0.1)
y = sigmoid(x)
plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.show()
```

✓ 0.1s



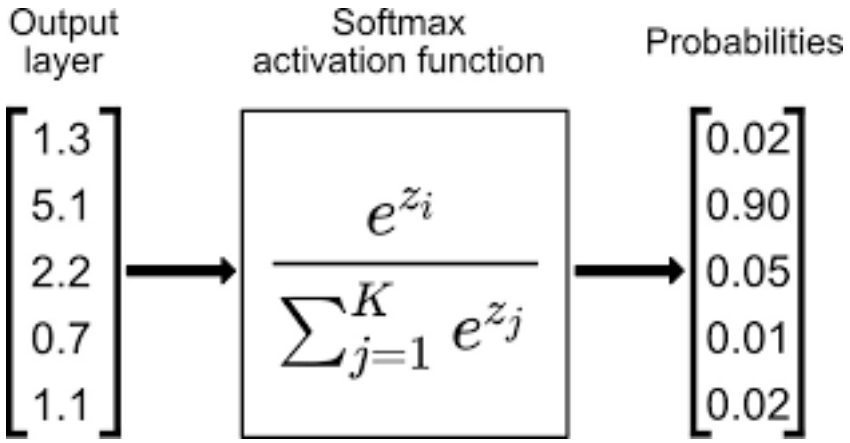
활성화 함수



내적 후 활성 함수

2. 신경망

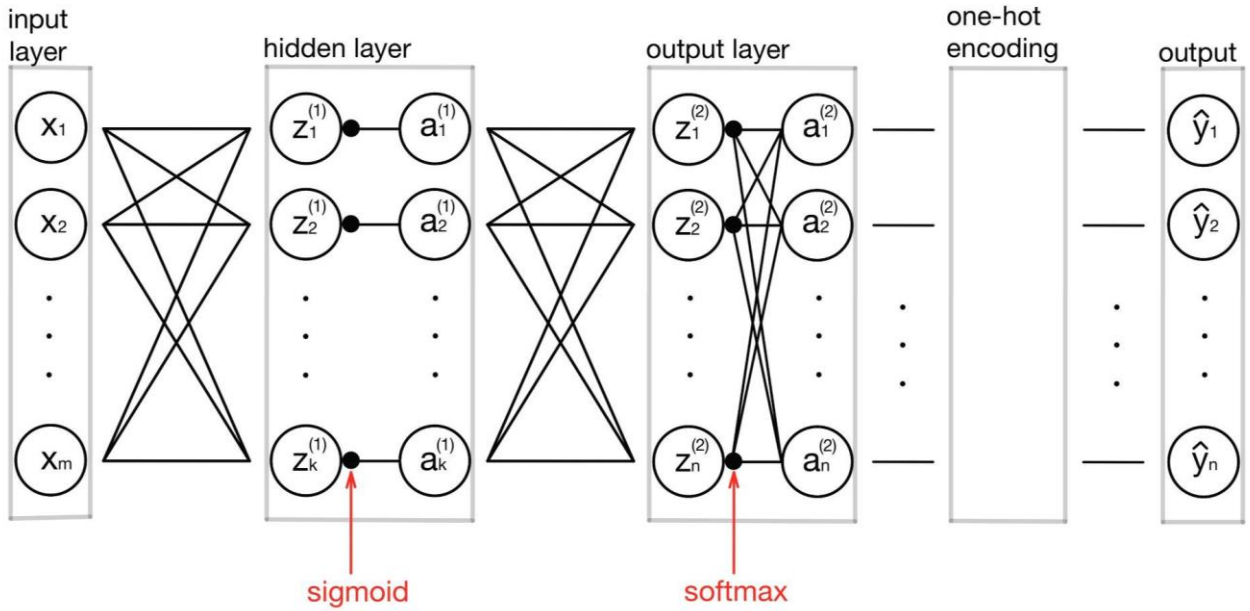
활성화 함수, 출력층, 구현




```python
def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a - c) # 오버플로를 방지
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a

    return y
```

✓ 0.2s

출력층



2. 신경망

활성화 함수, 출력층, 구현

표준 정규화와 비교할 때 Softmax에는 Nice 속성이 하나 있습니다.

분포는 균일 한 분포로 신경망의 낮은 자극 (흐린 이미지 생각)과 0과 1에 가까운 확률로 높은 자극 (예 : 큰 숫자, 선명한 이미지 생각)에 반응합니다.

표준 정규화는 비율이 동일한 한 신경 쓰지 않습니다.

소프트 맥스가 10 배 더 큰 입력을 가졌을 때 어떤 일이 발생하는지보십시오.

```
>>> softmax([1,2])              # blurry image of a ferret
[0.26894142,      0.73105858])  #     it is a cat perhaps !?
>>> softmax([10,20])            # crisp image of a cat
[0.0000453978687, 0.999954602]) #     it is definitely a CAT !
```

그런 다음 표준 정규화와 비교하십시오.

```
>>> std_norm([1,2])                     # blurry image of a ferret
[0.3333333333333333, 0.6666666666666666] #     it is a cat perhaps !?
>>> std_norm([10,20])                   # crisp image of a cat
[0.3333333333333333, 0.6666666666666666] #     it is a cat perhaps !?
```

129 Piotr Czapla 2017. 7. 19.

https://www.wake-up-neo.com/ko/math/%ED%91%9C%EC%A4%80-%EC%A0%95%EA%B7%9C%ED%99%94%EC%99%80-%EB%8B%AC%EB%A6%AC-softmax%EB%A5%BC-%EC%82%AC%EC%9A%A9%ED%95%98%EB%8A%94-%EC%9D%B4%EC%9C%A0%EB%8A%94-%EB%AC%B4%EC%97%87%EC%9E%85%EB%8B%88%EA%B9%8C/1072889787/

왜 수학적인 확률이 아닌 softmax를 쓰나?

$$\text{확률} = \frac{\text{어떤 일이 일어날 경우의 수}}{\text{모든 경우의 수}}$$

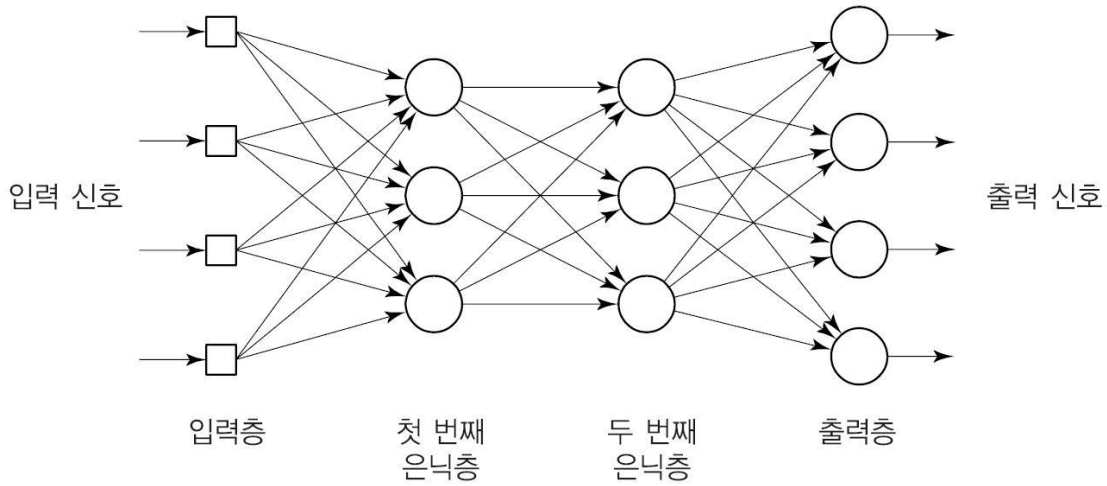
뭐가 다른지는 알겠는데 왜 쓰는지는 아직 모르겠다.

2. 신경망

활성화 함수, 출력층, 구현

```python
def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, W3) + b3
    y = softmax(a3)

    return y

x, t = get_data()
network = init_network()
#print(network)
accuracy_cnt = 0
for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y)
    if p == t[i]:
        accuracy_cnt += 1

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

구현



[그림 6-8] 두 개의 은닉층이 있는 다층 신경망

3. 신경망 학습

손실함수, 기울기, 학습 알고리즘

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

✓ 0.2s

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

✓ 0.3s

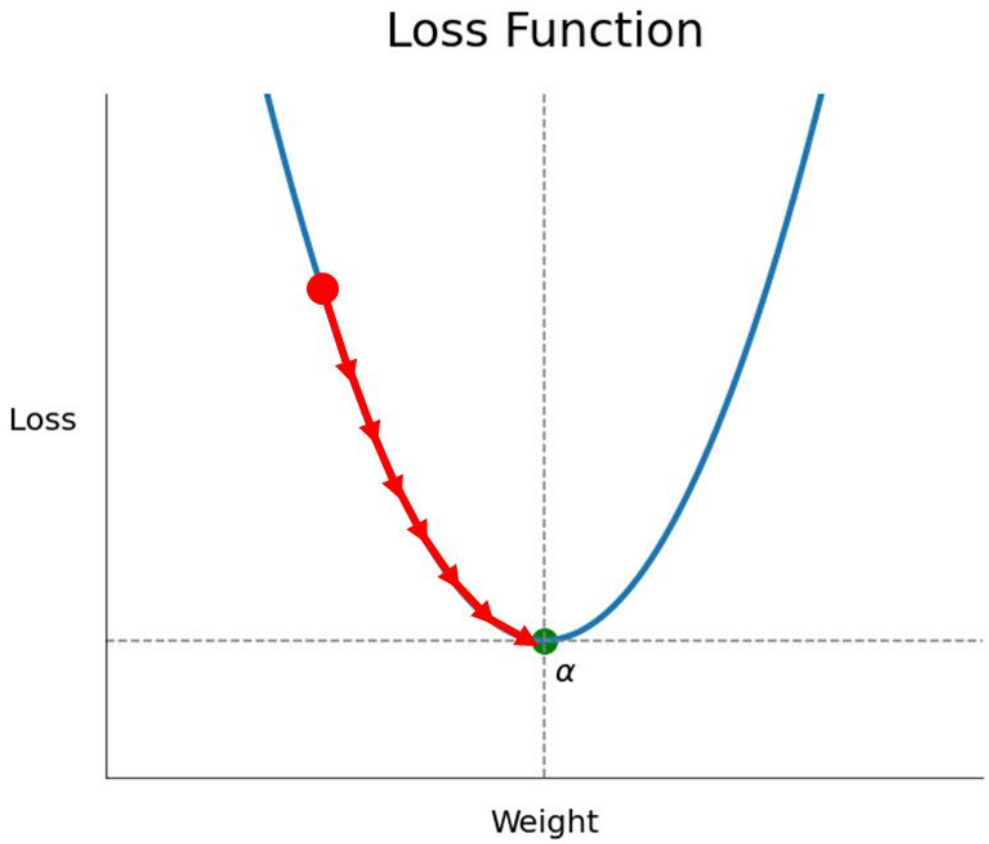
손실함수

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

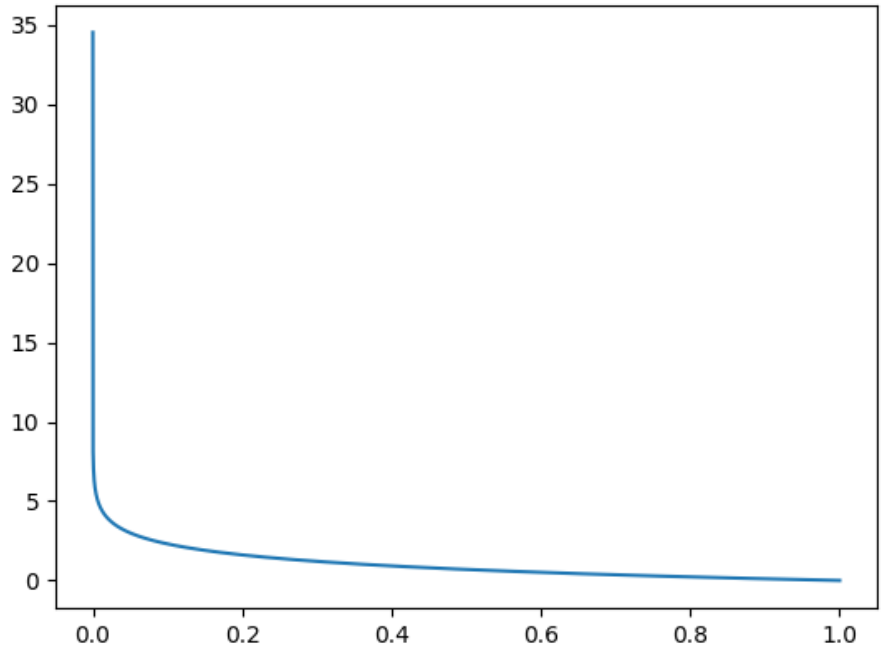
$$E = -\sum_k t_k \log y_k$$

3. 신경망 학습

손실함수, 기울기, 학습 알고리즘



손실함수 왜 사용하는가?



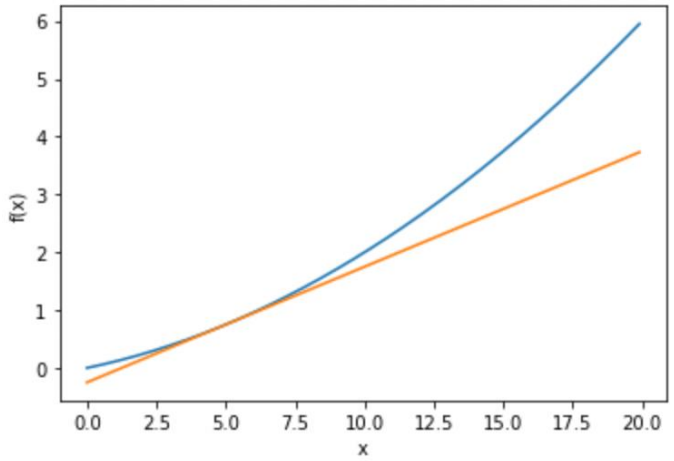
3. 신경망 학습

손실함수, 기울기, 학습 알고리즘

```python
def numerical_gradient(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
```

기울기

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

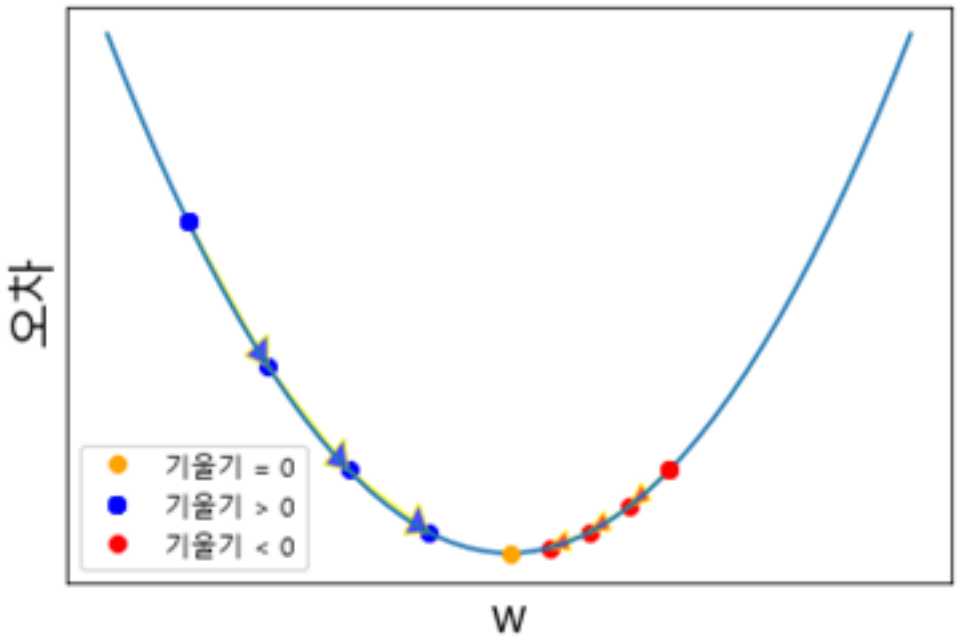


3. 신경망 학습

손실함수, 기울기, 학습 알고리즘

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

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

    return grad
```

기울기



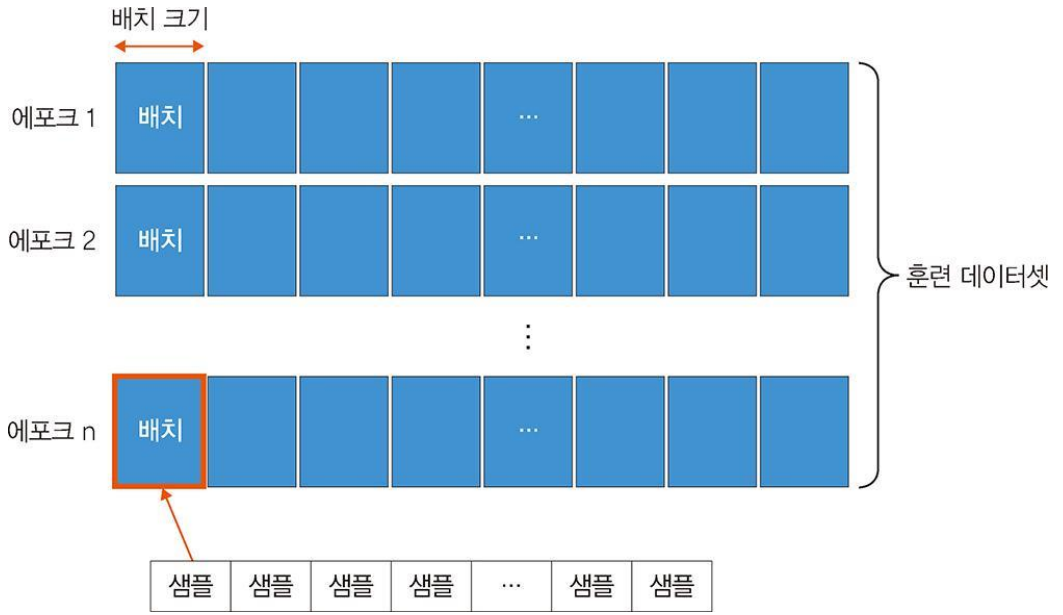
3. 신경망 학습

손실함수, 기울기, 학습 알고리즘

```
# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
# shape : (60000, 784) (60000, 10) (10000, 784) (10000, 10)
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

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

훈련 데이터 : 60000
미니 배치 크기 : 100
반복 횟수 : 10000
에포크 당 반복 : 600
에포크 : 10000 / 600
쓰는 데이터 : 10000 * 100 (?)

배치(묶음)와 에포크

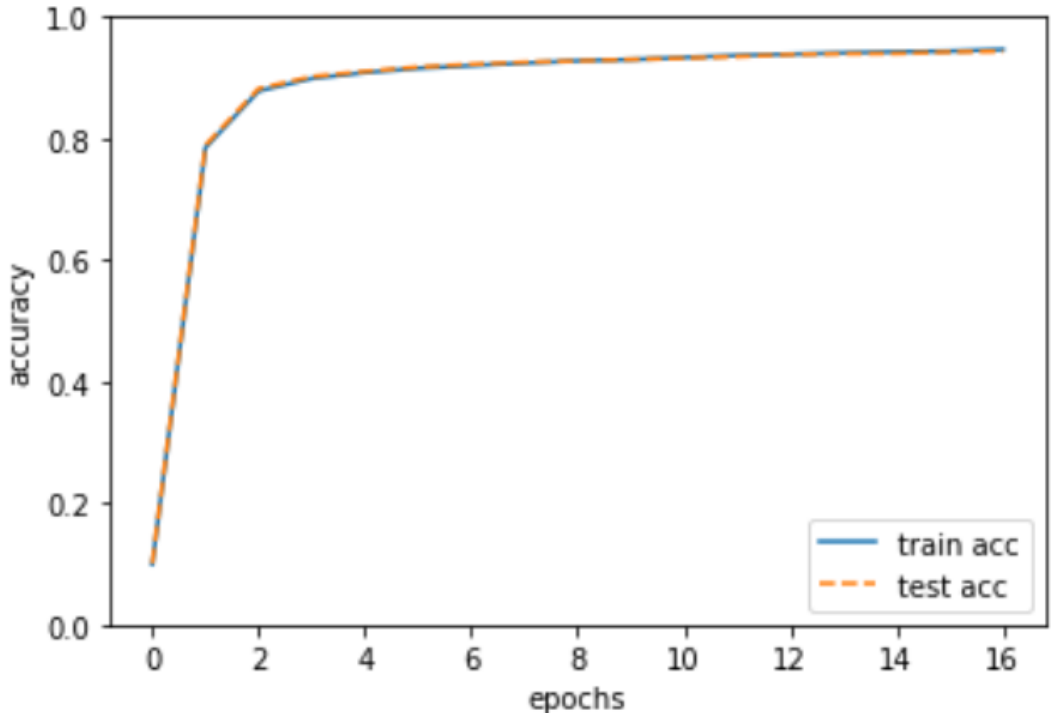


3. 신경망 학습

손실함수, 기울기, 학습 알고리즘

```python
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))
```

학습 알고리즘



3. 신경망 학습

손실함수, 기울기, 학습 알고리즘

단점

너무 오래 걸린다.

많은 양의 데이터를 다루기에 적합한 방법은 아니다.

3. 신경망 학습

손실함수, 기울기, 학습 알고리즘

개선 방법

오차 역전파를 통해 빠른 계산을 해보자.

Thank you