딥러닝 3장 신경망

2021년 3월 1일 월요일 오후 11:28

3.4 3층 신경망 구현하기

3층 신경망에서 수행되는 입력부터 출력까지의 처리를 구현

- 넘파이의 다차원 배열을 사용

1. 표기법

P.84 그림 3-16 참고

2. 각 층의 신호 전달 구현

P.85 그림 3-17 참고

a1(1)의 수식

$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

행렬의 곱을 이용하여 가중치 부분 간소화

$$\mathbf{A}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}$$

a. 입력층에서 1층으로 신호 전달

```
###### 3층 신경망 구현 ######
import numpy as np

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

def identity_function(x):
    return x

'''
1. 입력층에서 1층으로의 신호 전달
'''
X = np.array([1, 0.5])
W1 = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
B1 = np.array([0.1, 0.2, 0.3])

# 배열의 형상 확인
print (W1.shape) # (2, 3)
print (X.shape) # (2,)
print (B1.shape) # (3,)

# 은닉층에서의 가중치 합(a1) 출력
A1 = np.dot(X, W1) + B1

print (A1) # [0.3 0.7 1.1]
```

배열의 형상 확인은 X와 W의 대응하는 차원의 원소 수가 같은지 확인하기 위해 출력

b. 활성화 함수를 이용하여 입력 신호의 총합을 출력 신호로 변환

```
# 활성화 함수 h()로 변환된 신호(z1) 출력
Z1 = sigmoid(A1)

print (Z1) # [0.57444252 0.66818777 0.75026011]
```

c. 1층에서 2층으로 신호 전달(위의 내용 반복)

```
'''
2. 1층에서 2층으로의 신호 전달
'''
W2 = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
B2 = np.array([0.1, 0.2])

# 1층에서의 가중치 합(a2) 출력
A2 = np.dot(Z1, W2) + B2

print (A2) # [0.51615984 1.21402696]

# 활성화 함수 h()로 변환된 신호(z2) 출력
Z2 = sigmoid(A2)

print (Z2) # [0.62624937 0.7710107 ]
```

d. 2층에서 출력층으로 신호 전달

```
'''
3. 2층에서 출력층으로의 신호 전달
'''
W3 = np.array([[0.1, 0.3], [0.2, 0.4]])
B3 = np.array([0.1, 0.2])

# 1층에서의 가중치 합(a2) 출력
A3 = np.dot(Z2, W3) + B3

print (A3) # [0.31682708 0.69627909]

# 활성화 함수 σ()로 변환된 신호(y) 출력
Y = identity_function(A3)

print (Y) # [0.31682708 0.69627909]
```

출력층의 구현은 활성화 함수를 항등 함수인 identity_function()를 이용하여 신호를 출력한다. identity_function() 함수는 생략 가능하다. (위의 흐름을 통일 시키기 위해 구현)

3.5 출력층 설계하기

출력층의 활성화 함수는 풀고자 하는 문제의 성질에 맞게 정한다.

ex)
회귀 : 항등 함수
2클래스 분류 : 시그모이드 함수
다중 클래스 분류 : 소프트맥스 함수

항등 함수
- 입력 그대로 출력하는 함수(입력과 출력이 항상 같다)

소프트맥스 함수
- 입력받은 값을 출력으로 0~1사이의 값으로 정규화하여 출력하는 함수
- 출력의 총합은 1이다.
- 소프트맥스 함수를 적용해도 각 원소의 대소 관계는 변하지 않는다.
- 추론 단계에서는 출력층의 소프트맥스 함수를 생략하지만, 신경망 학습시킬 때는 출력층에서 소프트맥스를 사용한다.

1. 소프트맥스 함수 구현

P.92 코드 참고

2. 소프트맥스 함수 구현 시 주의점

소프트맥스는 지수함수를 사용하는데, 지수함수는 큰 결과값이 출력되고, 컴퓨터가 다룰 수 있는 수의 범위를 넘는 오버플로 문제가 발생한다.

오버플로 문제를 해결 하기 위한 해결방안은 다음과 같다.

1) C(임의의 정수)를 분자와 분모에 곱한다.
2) C를 함수 exp() 안으로 옮겨 logC 로 만든다.
3) logC를 C' 로 치환한다.

오버플로 문제 해결 예

```
###### 오버플로 문제 해결 방법 예제 ######
a = np.array([1010, 1000, 990])

print (np.exp(a) / np.sum(exp_a)) # [inf inf inf]

c = np.max(a) # 1010

print (a - c) # [  0 -10 -20]

print (np.exp(a - c) / np.sum(np.exp(a - c)))
# [9.99954600e-01 4.53978686e-05 2.06106005e-09]
```

소프트맥스 함수 생성 코드

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

    return y

a = np.array([0.3, 2.9, 4])
y = softmax(a)

print (y) # [0.01821127 0.24519181 0.73659691]

print (np.sum(y)) # 1.0
```

3.6 손글씨 숫자 인식

1. MNIST 데이터셋

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

# (훈련 이미지, 훈련 레이블), (시험 이미지, 시험 레이블) 형식으로 반환
(x_train, t_train), (x_test, t_test) = \
    load_mnist(flatten=True, normalize=False)

print(x_train.shape) # (60000, 784)
print(t_train.shape) # (60000,)
print(x_test.shape) # (10000, 784)
print(t_test.shape) # (10000,)
```

load_mnist함수 인수

1) normalize : 입력 이미지의 픽셀 값을 0~1 사이의 값으로 정규화 할지 정한다. False로 설정하면 원래 값 그대로 0~255 사이의 값을 유지한다.
2) flatten : 입력 이미지를 1차원 배열로 만들지 정한다. False로 설정하면 입력 이미지를 1x28x28의 3차원 배열로, True로 설정하면 784개의 원소로 이뤄진 1차원 배열로 저장한다.
3) one_hot_label : 레이블을 원-핫 인코딩 형태로 저장할지를 정한다. False로 설정하면 숫자 형태의 레이블을 저장하고, True로 설정하면 정답을 뜻하는 원소만 1이고 나머지는 0인 배열로 저장한다.

2. 신경망의 추론 처리

P.92~93 코드 참고

def init_network() 함수

- 가중치와 편향 매개변수가 딕셔너리 변수로 저장

정확도를 확인하는 for문

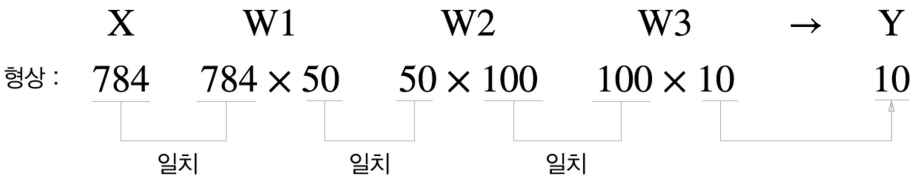
- x에 저장된 이미지 데이터를 한장씩 꺼내 predict() 함수로 분류

3. 배치 처리

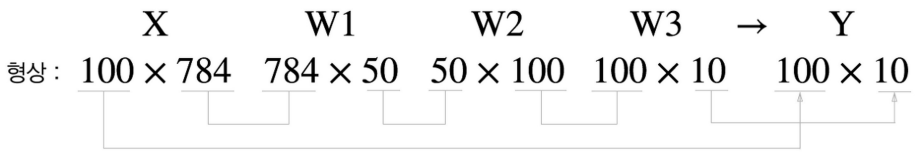
```
x, _ = get_data()
network = init_network()
W1, W2, W3 = network['W1'], network['W2'], network['W3']

print (x.shape) # (10000, 784)
print (x[0].shape) # (784,)
print (W1.shape) # (784, 50)
print (W2.shape) # (50, 100)
print (W3.shape) # (100, 10)
```

배열의 형상을 확인하면 X와 W의 대응하는 차원의 원소 수가 같다는 것을 확인 할 수 있다.



원소 784개로 구성된 1차원배열이 입력되어 마지막에는 원소가 10개인 1차원 배열이 출력된다.



100x784의 입력 데이터를 받아 100x10의 데이터가 출력된다.

위에서 처럼 하나로 묶은 입력 데이터를 배치 라고 한다.

배치의 장점

- 배치 처리는 컴퓨터로 계한할 때 이미지 1장당 처리 시간을 대폭 줄여주는 큰 이점을 준다.