연구지도 및 연구윤리 7주차 보고서

(Chapter3. 신경망 구현2(mnist))

경북대학교 전자공학부
2016113566 김남영

1. Softmax 함수 구현(출력층 설계)

```python
import numpy as np
import matplotlib as plt

# 지수가 너무 커서 overflow발생
a = np.array([1010, 1000, 990])
y= np.exp(a) / np.sum(np.exp(a))
print(y)

# 입력 신호 중 최댓값을 이용하여 보정
c = np.max(a)
a = a - c

y = np.exp(a) / np.sum(np.exp(a))
print(y) # 보정해도 결과는 변하지 않음

# 함수로 정리
def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a-c)
    sum_exp_a = np.sum(exp_a)

    return exp_a / sum_exp_a

print(softmax(a))

a = np.array([0.3, 2.9, 4.0])
y = softmax(a)
print(y)
print(np.sum(y)) #softmax 함수의 출력 총합은 1.0
```

```
ipdb> !debugfile('D:/coding/spyder/ch3/softmax_2.py', wdir='D:/coding/spyder/ch3')
Reloaded modules: dataset.mnist
d:\coding\spyder\ch3\softmax_2.py:6: RuntimeWarning: overflow encountered in exp
  y= np.exp(a) / np.sum(np.exp(a))
d:\coding\spyder\ch3\softmax_2.py:6: RuntimeWarning: invalid value encountered in true_divide
  y= np.exp(a) / np.sum(np.exp(a))
[nan nan nan]
[9.99954600e-01 4.53978686e-05 2.06106005e-09]
[9.99954600e-01 4.53978686e-05 2.06106005e-09]
[0.01821127 0.24519181 0.73659691]
1.0
```

1-1. 입력값 중 가장 큰 값으로 입력 신호를 보정하여도 소프트맥스 함수의 결과 값은 변하지 않는다.
1-2. 모든 출력값의 합은 1이므로 소프트맥스 함수의 출력을 확률로 해석할 수 있다.
1-3. 소프트맥스 함수를 거치더라도 입출력의 대소관계는 변하지 않으므로 추론 단계에서는 생략이 가능하다.

2. batch

```python
import numpy as np

x = np.array([[0.1, 0.8, 0.1], [0.3, 0.1, 0.6], [0.2, 0.5, 0.3], [0.8, 0.1, 0.1]])
y = np.argmax(x, axis = 1) # 열을 기준(axis=1이므로)으로 가장 큰 확률의 인덱스 반환

print(y)
```

```
ipdb> !debugfile('D:/coding/spyder/ch3/batch_test.py', wdir='D:/coding/spyder/ch3')
Reloaded modules: dataset.mnist
[1 2 1 0]
```

2-1. argmax 메소드는 인자로 들어온 값 중 가장 큰 값의 인덱스를 반환한다.
2-2. axis = 1 은 각 행을 기준으로 함을 의미한다.(각 행에서 가장 큰 값의 인덱스 반환)
2-3. axis = 0 은 각 열을 기준으로 함을 의미한다.

3. mnist 구현

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

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

def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a-c)
    sum_exp_a = np.sum(exp_a)
    return exp_a / sum_exp_a

def img_show(img):
    pil_img = Image.fromarray(np.uint8(img)) #데이터를 넘파이배열로 만든 뒤 fromarray를 통해 이미지로 변환
    pil_img.show()

def get_data(): # 입력 데이터 설정
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(normalize=True, flatten=True, one_hot_label=False)
    return x_test, t_test

def init_network(): # 가중치, 편향
    with open("sample_weight.pkl", 'rb') as f:
        network = pickle.load(f) #이미 학습이 되어있는 3층 신경망 sample 가중치 파일을 사용

    return network

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()
batch_size = 100
accuracy_cnt = 0

for i in range(0,len(x),batch_size): #0부터 len(x)-1까지 batch_size의 간격으로 증가
    x_batch = x[i:i+batch_size] #리스트 슬라이싱 (x[0]부터 x[99]까지)
    y_batch = predict(network, x_batch)
    p  = np.argmax(y_batch, axis = 1) # 열 기준(axis=1) 가장 높은 확률 인덱스 반환
    accuracy_cnt += np.sum(p == t[i:i+batch_size]) # true갯수 합만큼 cnt

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

```
ipdb> !debugfile('D:/coding/spyder/ch3/mnist_number_2.py', wdir='D:/coding/
spyder/ch3')
Accuracy:0.9352
```

3-1. mnist.py의 경로 설정이 중요하다.(같은 디렉토리라면 부모디렉토리 경로 설정 필요없음)
3-2. load_mnist를 통해 입력 데이터를 받을 수 있다. 3장의 경우, 학습은 하지않고 추론만 하기 때문에 테스트 데이터만 return 받으면 된다.
3-3. init_network() 구성 시에도 이미 학습된 3층 신경망(sample_weight)을 이용하여 구성한다.
3-4. 출력층의 softmax는 생략이 가능하므로 a3을 return하더라도 결과 값은 같다.
3-5. batch는 한번에 여러 개의 데이터를 처리하는 것을 말하고, 데이터를 읽는 횟수가 줄기 때문에 전체적인 속도가 대폭 줄어든다.(컴퓨터에서는 분할된 작은 배열을 여러 번 계산하는 것보다 큰 배열을 한꺼번에 계산 하는 것이 빠르다.)
3-6. 정확도는 93.52%로 매우 높게 나왔고, 신경망 층을 늘린다면 더 높은 정확도를 가질 것이다.
3-7. 이미 학습된 가중치를 사용하였기 때문에 임의의 값을 이용하여 층 수를 늘린다면 오히려 정확도가 감소할 것이다.

4. 5층 신경망 구현

```
def init_network(): # 가중치, 편향
    with open("sample_weight.pkl", 'rb') as f:
        network = pickle.load(f) #이미 학습이 되어있는 3층 신경망 sample 가중치 파일을 사용
    network['W4'] = np.array([[0.1, 0.5],[0.7, 0.2],[0.3, 0.8],[0.2, 0.1],[0.6, 0.7],
                              [0.2, 0.5],[0.1, 0.6],[0.3, 0.5],[0.3, 0.4],[0.7, 0.8]])
    network['b4'] = np.array([0.3, 0.4])
    network['W5'] = np.array([[0.1, 0.2, 0.3, 0.4, 0.5, 0.4, 0.3, 0.2, 0.1, 0.7],
                              [0.3, 0.5, 0.1, 0.7, 0.6, 0.1, 0.4, 0.5, 0.9, 0.4]])
    network['b5'] = np.array([0.3, 0.4, 0.2, 0.6, 0.7, 0.4, 0.5, 0.1, 0.4, 0.1])
    return network

def predict(network, x):
    W1, W2, W3, W4, W5 = network['W1'], network['W2'], network['W3'], network['W4'], network['W5']
    b1, b2, b3, b4, b5 = network['b1'], network['b2'], network['b3'], network['b4'], network['b5']

    a1 = np.dot(x,W1) +b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1,W2) +b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2,W3) +b3
    z3 = sigmoid(a3)
    a4 = np.dot(z3,W4) +b4
    z4 = sigmoid(a4)
    a5 = np.dot(z4,W5) +b5
    y = softmax(a5)

    return y
```

```
ipdb> !debugfile('D:/coding/spyder/ch3/mnist_number_4.py', wdir='D:/coding/
spyder/ch3')
Accuracy:0.0982
```

4-1. 5층 신경망으로 구현하기 위해 W4, W5, b4, b5를 임의의 값을 넣어 network에 추가하였다.
4-2. 정확도가 약 10%정도로, 학습된 3층 신경망에 비해 확연히 떨어졌다.
4-3. 처음에는 784개의 행이 있는 1층부터 수작업으로 network를 구성해야한다고 생각했는데, 기존의 3층 신경망에 4층, 5층 network만 추가해주면 될 일 이었다.

5. 고찰

5-1. 처음에 mnist 파일의 경로 설정 문제로 시간을 많이 썼다. 하지만 그만큼 다음 과제를 할 때에는 더 신중해 질 것이라 생각한다.
5-2. 오늘은 이미 학습된 가중치 파일을 이용해서 추론만 했는데, 앞으로는 학습과 추론 모두 혼자 구현해보고 싶다.
5-3. 이전에 딥러닝 강의를 들었을 때에는 이해되지 않던 개념들이 점점 이해되기 시작해서 뿌듯하다.
5-4. 막연히 층수를 깊게 하면 더 정확해진다고 생각은 했지만, 정말로 그러한지 궁금하다. 가중치 학습까지 배우게 된다면 같은 예제로 5층 신경망을 구현해 보아야겠다.
5-5. 가중치를 학습할 때 어떤 기준으로 피드백을 할지 궁금하다.