인공신경망(MLP) 구현하기

소프트웨어공학

2017156034

전상민

210513~210517

—

인공지능(01)

—

배유석 교수님 下

1. 코드 상세 분석

```
import numpy as np
import matplotlib.pyplot as plt
import time
from dataset.mnist import load_mnist

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

def sigmoid_grad(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

- sigmoid 함수 와 그 미분함수 정의

```

#%% made functions
```

… (설명 순서를 위해 후로 이동)

```

#%% parameters
#parameters
input_size=784
hidden_size=50
output_size=10
```

- 파라미터 설정.
- 입력층 노드수, 은닉층 노드수, 출력층 노드수.

```

# hyperparameter
iters_num = 30000
train_size = 60000
```

```
batch_size = 100
learning_rate = 0.1
```

- 하이퍼파라미터 설정.
- 반복 횟수, 입력 크기, 미니배치 크기, 학습률

```

train_acc_list = []
test_acc_list = []

epoch = 0

#%% parsing
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
```

- 모델 불러오기

```

pr = {}
pr['W1'] = 0.01 * np.random.randn(input_size, hidden_size)
pr['b1'] = np.zeros(hidden_size)
pr['W2'] = 0.01 * np.random.randn(hidden_size, output_size)
pr['b2'] = np.zeros(output_size)
```

- 가중치 초기값 선언

```

#%% training
```

- 학습

```
# start timer
start_time = time.time()
```

- 시간 측정 시작

```

for i in range(iters_num):
```

- 학습횟수 동안

```
    # mini-batch
    batch_mask = np.random.choice(train_size, batch_size)
    x = x_train[batch_mask]
    t = t_train[batch_mask]
```

- 미니배치 크기: 100

```
    # gradient
    W1, W2 = pr['W1'], pr['W2']
    b1, b2 = pr['b1'], pr['b2']

    # forwards using sigmoid
```

- sigmoid 활성화 함수를 이용한 순방향 학습

```
    net1 = np.dot(x, W1) + b1
```

- $net_{pj} = \sum W_{ji} O_{pi} + b_{ji}$

```
    o1 = sigmoid(net1)
```

- $O_{pj} = S(net_{pj})$

```
    net2 = np.dot(o1, W2) + b2
```

- $net_{pk} = \sum W_{kj} O_{pj} + b_{kj}$

```
    y = sigmoid(net2)
```

- $O_{pk} = S(net_{pk})$

```
    # backwards using MSE, sigmoid
```

- 경사하강법을 이용한 역전파 학습

```
    pr['W2'] -= learning_rate * np.dot(o1.T, (y - t) / batch_size * sigmoid_grad(net2))
```

- $\Delta W_{kj}(n+1) = \eta \delta_{pk} O_{pj} + \Delta W_{kj}(n) = \eta \left(t_{pk} - O_{pk}\right) f_k'(net_{pk}) O_{pj} + \Delta W_{kj}(n)$

```
    pr['b2'] -= learning_rate * np.sum((y - t) / batch_size, axis=0)
```

- $\Delta b_{kj}(n+1) = \eta \sum \left(t_{pk} - O_{pk}\right) + \Delta b_{kj}(n)$

```
    pr['W1'] -= learning_rate * np.dot(x.T, sigmoid_grad(net1) * np.dot((y - t) / batch_size, W2.T))
```

- $\Delta W_{ji}(n+1) = \eta \delta_{pj} O_{pi} + \Delta W_{ji}(n) = \eta f_j'\left(net_{pj}\right) \sum \delta_{pk} W_{kj} \cdot O_{pi} + \Delta W_{ji}(n)$
- $= \eta O_{pi} O_{pj} (1 - O_{pj}) \sum \delta_{pk} W_{kj} + \Delta W_{ji}(n)$

```
    pr['b1'] -= learning_rate * np.sum(sigmoid_grad(net1) * np.dot((y - t) / batch_size, W2.T), axis=0)
```

- $\Delta b_{ji}(n+1) = \eta f_j'\left(net_{pj}\right) \sum \left(t_{pk} - O_{pk}\right) W_{kj} + \Delta b_{ji}(n)$

```

    # accuracy
    if i % 1000 == 0:
        train_acc = accuracy(x_train, t_train)
```

- 정확도를 한 epoch=1000 마다 계산한다.
- 정확도 계산 함수

```
#%% made functions
def accuracy(x, t):
    W1, W2 = pr['W1'], pr['W2']
    b1, b2 = pr['b1'], pr['b2']

    net1 = np.dot(x, W1) + b1
    o1 = sigmoid(net1)
    net2 = np.dot(o1, W2) + b2
    y = sigmoid(net2)
```

- 학습 과정 같음

```
    y = np.argmax(y, axis=1)
    t = np.argmax(t, axis=1)
```

- 선택된 노드를 (0 중의 1 인 노드를) 선택하여 학습된 결과를 도출하고,

```
    acc = np.sum(y == t) / float(x.shape[0])
```

- 비교하여 학습률 계산

```
    return acc

        test_acc = accuracy(x_test, t_test)
```

- 똑같이 테스트 정확도 계산

```
        print("Epoch: " + str(epoch) +
              "\ttrain acc: " + str(train_acc) +
              ",\ttest acc: " + str(test_acc) +
              ",\t time lapsed: " + str(time.time() - start_time))
        start_time = time.time()
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
```

- 정확도를 리스트에 저장

```
        epoch += 1

#%% graphing
```

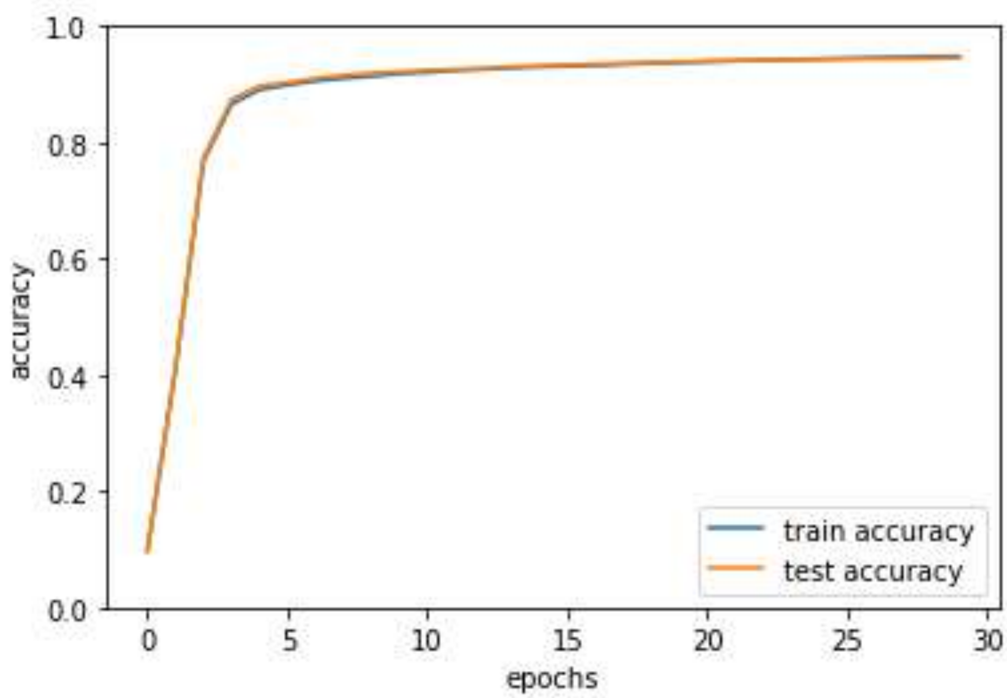
- 저장한 정확도 리스트를 그래프화

```
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train accuracy')
plt.plot(x, test_acc_list, label='test accuracy')
plt.xlabel("epochs")
plt.ylabel("accuracy")
```

```
plt.ylim(0, 1)
plt.legend()
plt.show()
```

2. 실행 결과

```
Epoch: 0    train acc: 0.10218333333333333, test acc: 0.101,     time lapsed: 0.32916975021362305
Epoch: 1    train acc: 0.45655, test acc: 0.4482,    time lapsed: 1.5106048583984375
Epoch: 2    train acc: 0.7813166666666667,  test acc: 0.7905,    time lapsed: 1.2560725212097168
Epoch: 3    train acc: 0.8613666666666666,  test acc: 0.8696,    time lapsed: 1.3701982498168945
Epoch: 4    train acc: 0.8859833333333333,  test acc: 0.8905,    time lapsed: 1.376024246158203
Epoch: 5    train acc: 0.8971666666666667,  test acc: 0.9012,    time lapsed: 1.3447070121765137
Epoch: 6    train acc: 0.90295, test acc: 0.906,     time lapsed: 1.3938734531402588
Epoch: 7    train acc: 0.9082166666666667,  test acc: 0.9091,    time lapsed: 1.2829008102416992
Epoch: 8    train acc: 0.91295, test acc: 0.9148,    time lapsed: 1.391618013381958
Epoch: 9    train acc: 0.9158833333333334,  test acc: 0.9189,    time lapsed: 1.4570722579956055
Epoch: 10   train acc: 0.9185333333333333,  test acc: 0.9206,    time lapsed: 1.5462641716003418
Epoch: 11   train acc: 0.9210833333333334,  test acc: 0.9232,    time lapsed: 1.6465339660644531
Epoch: 12   train acc: 0.92345, test acc: 0.9248,    time lapsed: 1.8278827667236328
Epoch: 13   train acc: 0.9253166666666667,  test acc: 0.9267,    time lapsed: 1.6424288749694824
Epoch: 14   train acc: 0.92705, test acc: 0.9288,    time lapsed: 1.207472801208496
Epoch: 15   train acc: 0.92845, test acc: 0.93,  time lapsed: 1.4072480201721191
Epoch: 16   train acc: 0.9307,  test acc: 0.9321,    time lapsed: 1.32912278175354
Epoch: 17   train acc: 0.9327666666666666,  test acc: 0.9344,    time lapsed: 1.3576061725616455
Epoch: 18   train acc: 0.9342,  test acc: 0.9345,    time lapsed: 1.3134839534759521
Epoch: 19   train acc: 0.93555, test acc: 0.9358,    time lapsed: 1.2822506427764893
Epoch: 20   train acc: 0.9368166666666666,  test acc: 0.9365,    time lapsed: 1.430248737335205
Epoch: 21   train acc: 0.9377333333333333,  test acc: 0.9372,    time lapsed: 1.2870090007781982
Epoch: 22   train acc: 0.93915, test acc: 0.9386,    time lapsed: 1.3585829734802246
Epoch: 23   train acc: 0.9407833333333333,  test acc: 0.9394,    time lapsed: 1.6220896244049072
Epoch: 24   train acc: 0.9411166666666667,  test acc: 0.9397,    time lapsed: 1.5010335445404053
Epoch: 25   train acc: 0.94265, test acc: 0.9406,    time lapsed: 1.349952220916748
Epoch: 26   train acc: 0.94385, test acc: 0.9407,    time lapsed: 1.3926646709442139
Epoch: 27   train acc: 0.9445833333333333,  test acc: 0.9416,    time lapsed: 1.4289710521697998
Epoch: 28   train acc: 0.9455333333333333,  test acc: 0.942,     time lapsed: 1.5196926593780518
Epoch: 29   train acc: 0.9464333333333333,  test acc: 0.9435,    time lapsed: 1.530839204788208
```



3. 추가 사항

[지시사항]

- framework 사용 하지 않음
- sigmoid, mean squared error, mini-batch 사용
- 그래프 출력
- 3-layer
- 은닉층 개수 유연, 수정 가능
- 경과 시간 출력
- Weight 조회 가능

[추가 옵션]

- 매개변수 갱신(SGD)
- 가중치 초기값(Random)
- 하이퍼파라미터 세팅