산업인공지능개론

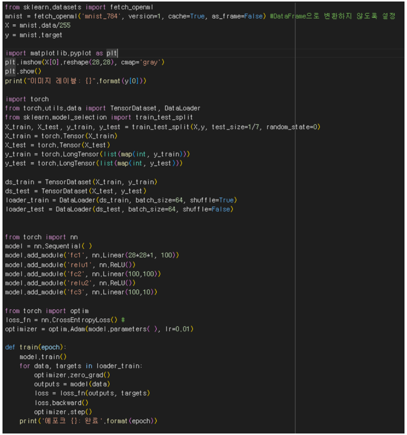
MLP, CNN

2023254006
이선경

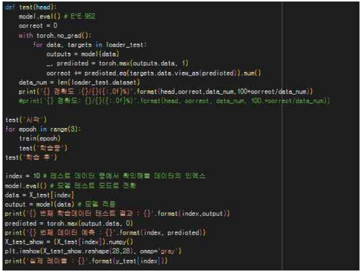
CONTENS

01. MLP

코드 구현



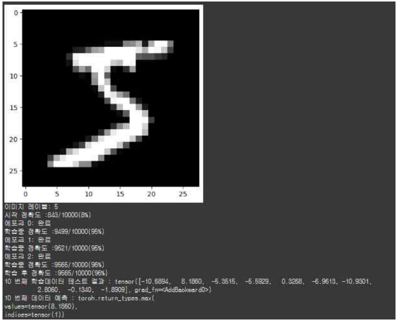
```python
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True, as_frame=False) #DataFrame으로 변환하지 않도록 설정
X = mnist.data/255
y = mnist.target

import matplotlib.pyplot as plt
plt.imshow(X[0].reshape(28,28), cmap='gray')
plt.show()
print("이미지 레이블: {}".format(y[0]))

import torch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=1/7, random_state=0)
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.LongTensor(list(map(int, y_train)))
y_test = torch.LongTensor(list(map(int, y_test)))

ds_train = TensorDataset(X_train, y_train)
ds_test = TensorDataset(X_test, y_test)
loader_train = DataLoader(ds_train, batch_size=64, shuffle=True)
loader_test = DataLoader(ds_test, batch_size=64, shuffle=False)


from torch import nn
model = nn.Sequential( )
model.add_module('fc1', nn.Linear(28*28*1, 100))
model.add_module('relu1', nn.ReLU())
model.add_module('fc2', nn.Linear(100,100))
model.add_module('relu2', nn.ReLU())
model.add_module('fc3', nn.Linear(100,10))

from torch import optim
loss_fn = nn.CrossEntropyLoss() #
optimizer = optim.Adam(model.parameters( ), lr=0.01)

def train(epoch):
    model.train()
    for data, targets in loader_train:
        optimizer.zero_grad()
        outputs = model(data)
        loss = loss_fn(outputs, targets)
        loss.backward()
        optimizer.step()
    print('에포크 {}: 완료'.format(epoch))
```



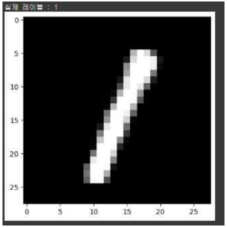
```python
def test(head):
    model.eval() # E^E 952
    correct = 0
    with torch.no_grad():
        for data, targets in loader_test:
            outputs = model(data)
            _, predicted = torch.max(outputs.data, 1)
            correct += predicted.eq(targets.data.view_as(predicted)).sum()
    data_num = len(loader_test.dataset)
    print('{} 정확도 :{}/{}({:.0f}%)'.format(head,correct,data_num,100*correct/data_num))
    #print('{} 정확도: {}/{}({:.0f}%)'.format(head, correct, data_num, 100.*correct/data_num))

test('시작')
for epoch in range(3):
    train(epoch)
    test('학습중')
test('학습 후')

index = 10 # 테스트 데이터 중에서 확인해볼 데이터의 인덱스
model.eval() # 모델 테스트 모드로 전환
data = X_test[index]
output = model(data) # 모델 적용
print('{} 번째 학습데이터 테스트 결과 : {}'.format(index,output))
predicted = torch.max(output.data, 0)
print('{} 번째 데이터 예측 : {}'.format(index, predicted))
X_test_show = (X_test[index]).numpy()
plt.imshow(X_test_show.reshape(28,28), cmap='gray')
print('실제 레이블 : {}'.format(y_test[index]))
```

01. MLP

결과



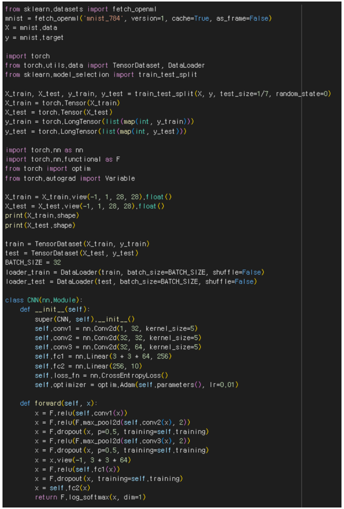
```
이미지 레이블: 5
시작 정확도 :843/10000(8%)
에포크 0: 완료
학습중 정확도 :9499/10000(95%)
에포크 1: 완료
학습중 정확도 :9521/10000(95%)
에포크 2: 완료
학습중 정확도 :9565/10000(96%)
학습 후 정확도 :9565/10000(96%)
10 번째 학습데이터 테스트 결과 : tensor([-10.6894,   8.1860,  -5.3515,  -5.5929,   0.3268,  -6.9613, -10.9301,
          2.8060,  -0.1340,  -1.8909], grad_fn=<AddBackward0>)
10 번째 데이터 예측 : torch.return_types.max(
values=tensor(8.1860),
indices=tensor(1))
```



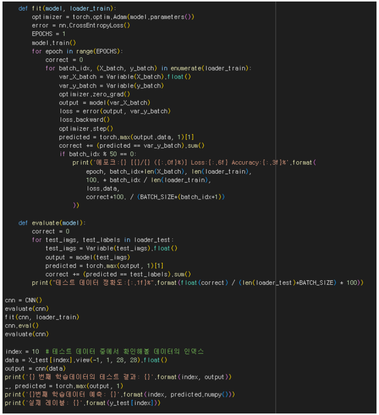
```
실제 레이블 : 1
```

02. CNN

코드 구현



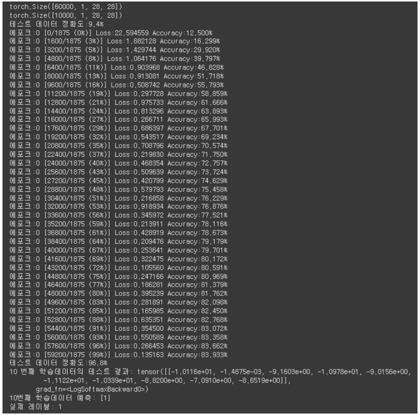
```python
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True, as_frame=False)
X = mnist.data
y = mnist.target

import torch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/7, random_state=0)
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.LongTensor(list(map(int, y_train)))
y_test = torch.LongTensor(list(map(int, y_test)))

import torch.nn as nn
import torch.nn.functional as F
from torch import optim
from torch.autograd import Variable

X_train = X_train.view(-1, 1, 28, 28).float()
X_test = X_test.view(-1, 1, 28, 28).float()
print(X_train.shape)
print(X_test.shape)

train = TensorDataset(X_train, y_train)
test = TensorDataset(X_test, y_test)
BATCH_SIZE = 32
loader_train = DataLoader(train, batch_size=BATCH_SIZE, shuffle=False)
loader_test = DataLoader(test, batch_size=BATCH_SIZE, shuffle=False)

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=5)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=5)
        self.fc1 = nn.Linear(3 * 3 * 64, 256)
        self.fc2 = nn.Linear(256, 10)
        self.loss_fn = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.parameters(), lr=0.01)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(F.max_pool2d(self.conv3(x), 2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = x.view(-1, 3 * 3 * 64)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```



```python
    def fit(model, loader_train):
        optimizer = torch.optim.Adam(model.parameters())
        error = nn.CrossEntropyLoss()
        EPOCHS = 1
        model.train()
        for epoch in range(EPOCHS):
            correct = 0
            for batch_idx, (X_batch, y_batch) in enumerate(loader_train):
                var_X_batch = Variable(X_batch).float()
                var_y_batch = Variable(y_batch)
                optimizer.zero_grad()
                output = model(var_X_batch)
                loss = error(output, var_y_batch)
                loss.backward()
                optimizer.step()
                predicted = torch.max(output.data, 1)[1]
                correct += (predicted == var_y_batch).sum()
                if batch_idx % 50 == 0:
                    print('에포크:{} [{}/{} ({:.0f}%)] Loss:{:.6f} Accuracy:{:.3f}%'.format(
                        epoch, batch_idx*len(X_batch), len(loader_train),
                        100. * batch_idx / len(loader_train),
                        loss.data,
                        correct*100. / (BATCH_SIZE*(batch_idx+1))
                    ))

    def evaluate(model):
        correct = 0
        for test_imgs, test_labels in loader_test:
            test_imgs = Variable(test_imgs).float()
            output = model(test_imgs)
            predicted = torch.max(output, 1)[1]
            correct += (predicted == test_labels).sum()
        print("테스트 데이터 정확도:{:.1f}%".format(float(correct) / (len(loader_test)*BATCH_SIZE) * 100))

cnn = CNN()
evaluate(cnn)
fit(cnn, loader_train)
cnn.eval()
evaluate(cnn)

index = 10  # 테스트 데이터 중에서 확인해볼 데이터의 인덱스
data = X_test[index].view(-1, 1, 28, 28).float()
output = cnn(data)
print('{} 번째 학습데이터의 테스트 결과: {}'.format(index, output))
_, predicted = torch.max(output, 1)
print('{}번째 학습데이터 예측: {}'.format(index, predicted.numpy()))
print('실제 레이블: {}'.format(y_test[index]))
```

02. CNN

결과



```
torch.Size([60000, 1, 28, 28])
torch.Size([10000, 1, 28, 28])
테스트 데이터 정확도:9.4%
에포크:0 [0/1875 (0%)] Loss:22.594559 Accuracy:12.500%
에포크:0 [1600/1875 (3%)] Loss:1.882128 Accuracy:16.299%
에포크:0 [3200/1875 (5%)] Loss:1.429744 Accuracy:29.920%
에포크:0 [4800/1875 (8%)] Loss:1.064176 Accuracy:39.797%
에포크:0 [6400/1875 (11%)] Loss:0.903968 Accuracy:46.828%
에포크:0 [8000/1875 (13%)] Loss:0.913081 Accuracy:51.718%
에포크:0 [9600/1875 (16%)] Loss:0.508742 Accuracy:55.793%
에포크:0 [11200/1875 (19%)] Loss:0.297728 Accuracy:58.859%
에포크:0 [12800/1875 (21%)] Loss:0.975733 Accuracy:61.666%
에포크:0 [14400/1875 (24%)] Loss:0.813296 Accuracy:63.893%
에포크:0 [16000/1875 (27%)] Loss:0.266711 Accuracy:65.993%
에포크:0 [17600/1875 (29%)] Loss:0.686397 Accuracy:67.701%
에포크:0 [19200/1875 (32%)] Loss:0.543517 Accuracy:69.234%
에포크:0 [20800/1875 (35%)] Loss:0.708796 Accuracy:70.574%
에포크:0 [22400/1875 (37%)] Loss:0.219830 Accuracy:71.750%
에포크:0 [24000/1875 (40%)] Loss:0.468354 Accuracy:72.757%
에포크:0 [25600/1875 (43%)] Loss:0.509639 Accuracy:73.724%
에포크:0 [27200/1875 (45%)] Loss:0.420799 Accuracy:74.629%
에포크:0 [28800/1875 (48%)] Loss:0.579793 Accuracy:75.458%
에포크:0 [30400/1875 (51%)] Loss:0.216858 Accuracy:76.229%
에포크:0 [32000/1875 (53%)] Loss:0.918934 Accuracy:76.876%
에포크:0 [33600/1875 (56%)] Loss:0.345972 Accuracy:77.521%
에포크:0 [35200/1875 (59%)] Loss:0.213911 Accuracy:78.116%
에포크:0 [36800/1875 (61%)] Loss:0.428919 Accuracy:78.673%
에포크:0 [38400/1875 (64%)] Loss:0.209476 Accuracy:79.179%
에포크:0 [40000/1875 (67%)] Loss:0.253641 Accuracy:79.701%
에포크:0 [41600/1875 (69%)] Loss:0.322475 Accuracy:80.172%
에포크:0 [43200/1875 (72%)] Loss:0.105560 Accuracy:80.591%
에포크:0 [44800/1875 (75%)] Loss:0.247166 Accuracy:80.969%
에포크:0 [46400/1875 (77%)] Loss:0.186281 Accuracy:81.379%
에포크:0 [48000/1875 (80%)] Loss:0.395239 Accuracy:81.762%
에포크:0 [49600/1875 (83%)] Loss:0.281891 Accuracy:82.098%
에포크:0 [51200/1875 (85%)] Loss:0.165985 Accuracy:82.450%
에포크:0 [52800/1875 (88%)] Loss:0.635351 Accuracy:82.768%
에포크:0 [54400/1875 (91%)] Loss:0.354500 Accuracy:83.072%
에포크:0 [56000/1875 (93%)] Loss:0.550589 Accuracy:83.358%
에포크:0 [57600/1875 (96%)] Loss:0.266453 Accuracy:83.662%
에포크:0 [59200/1875 (99%)] Loss:0.135163 Accuracy:83.933%
테스트 데이터 정확도:96.8%
10 번째 학습데이터의 테스트 결과: tensor([[-1.0116e+01, -1.4875e-03, -9.1603e+00, -1.0978e+01, -9.0156e+00,
         -1.1122e+01, -1.0339e+01, -8.8200e+00, -7.0910e+00, -8.6519e+00]],
       grad_fn=<LogSoftmaxBackward0>)
10번째 학습데이터 예측: [1]
실제 레이블: 1
```