mlp

# example code

|  |
| --- |
| ''' 1. Module Import '''  import numpy as np  import matplotlib.pyplot as plt  import torch  import torch.nn as nn  import torch.nn.functional as F  from torchvision import transforms, datasets  ''' 2. 딥러닝 모델을 설계할 때 활용하는 장비 확인 '''  if torch.cuda.is\_available():      DEVICE = torch.device('cuda')  else:      DEVICE = torch.device('cpu')  print('Using PyTorch version:', torch.\_\_version\_\_, ' Device:', DEVICE)  BATCH\_SIZE = 32  EPOCHS = 10  ''' 3. MNIST 데이터 다운로드 (Train set, Test set 분리하기) '''  train\_dataset = datasets.MNIST(root = "../data/MNIST",                                 train = True,                                 download = True,                                 transform = transforms.ToTensor())  test\_dataset = datasets.MNIST(root = "../data/MNIST",                                train = False,                                transform = transforms.ToTensor())  train\_loader = torch.utils.data.DataLoader(dataset = train\_dataset,                                             batch\_size = BATCH\_SIZE,                                             shuffle = True)  test\_loader = torch.utils.data.DataLoader(dataset = test\_dataset,                                            batch\_size = BATCH\_SIZE,                                            shuffle = False)  ''' 4. 데이터 확인하기 (1) '''  for (X\_train, y\_train) in train\_loader:      print('X\_train:', X\_train.size(), 'type:', X\_train.type())      print('y\_train:', y\_train.size(), 'type:', y\_train.type())      break  ''' 5. 데이터 확인하기 (2) '''  pltsize = 1  plt.figure(figsize=(10 \* pltsize, pltsize))  for i in range(10):      plt.subplot(1, 10, i + 1)      plt.axis('off')      plt.imshow(X\_train[i, :, :, :].numpy().reshape(28, 28), cmap = "gray\_r")      plt.title('Class: ' + str(y\_train[i].item()))  ''' 6. Multi Layer Perceptron (MLP) 모델 설계하기 '''  class Net(nn.Module):      def \_\_init\_\_(self):          super(Net, self).\_\_init\_\_()          self.fc1 = nn.Linear(28 \* 28, 512)          self.fc2 = nn.Linear(512, 256)          self.fc3 = nn.Linear(256, 10)          self.dropout\_prob = 0.5          self.batch\_norm1 = nn.BatchNorm1d(512)          self.batch\_norm2 = nn.BatchNorm1d(256)      def forward(self, x):          x = x.view(-1, 28 \* 28)          x = self.fc1(x)          x = self.batch\_norm1(x)          x = F.relu(x)          x = F.dropout(x, training = self.training, p = self.dropout\_prob)          x = self.fc2(x)          x = self.batch\_norm2(x)          x = F.relu(x)          x = F.dropout(x, training = self.training, p = self.dropout\_prob)          x = self.fc3(x)          x = F.log\_softmax(x, dim = 1)          return x  ''' 7. Optimizer, Objective Function 설정하기 '''  import torch.nn.init as init  def weight\_init(m):      if isinstance(m, nn.Linear):          init.kaiming\_uniform\_(m.weight.data)  model = Net().to(DEVICE)  model.apply(weight\_init)  optimizer = torch.optim.Adam(model.parameters(), lr = 0.01)  criterion = nn.CrossEntropyLoss()  print(model)  ''' 8. MLP 모델 학습을 진행하며 학습 데이터에 대한 모델 성능을 확인하는 함수 정의 '''  def train(model, train\_loader, optimizer, log\_interval):      model.train()      for batch\_idx, (image, label) in enumerate(train\_loader):          image = image.to(DEVICE)          label = label.to(DEVICE)          optimizer.zero\_grad()          output = model(image)          loss = criterion(output, label)          loss.backward()          optimizer.step()          if batch\_idx % log\_interval == 0:              print("Train Epoch: {} [{}/{} ({:.0f}%)]\tTrain Loss: {:.6f}".format(                  epoch, batch\_idx \* len(image),                  len(train\_loader.dataset), 100. \* batch\_idx / len(train\_loader),                  loss.item()))  ''' 9. 학습되는 과정 속에서 검증 데이터에 대한 모델 성능을 확인하는 함수 정의 '''  def evaluate(model, test\_loader):      model.eval()      test\_loss = 0      correct = 0      with torch.no\_grad():          for image, label in test\_loader:              image = image.to(DEVICE)              label = label.to(DEVICE)              output = model(image)              test\_loss += criterion(output, label).item()              prediction = output.max(1, keepdim = True)[1]              correct += prediction.eq(label.view\_as(prediction)).sum().item()        test\_loss /= (len(test\_loader.dataset) / BATCH\_SIZE)      test\_accuracy = 100. \* correct / len(test\_loader.dataset)      return test\_loss, test\_accuracy  ''' 10. MLP 학습 실행하며 Train, Test set의 Loss 및 Test set Accuracy 확인하기 '''  for epoch in range(1, EPOCHS + 1):      train(model, train\_loader, optimizer, log\_interval = 200)      test\_loss, test\_accuracy = evaluate(model, test\_loader)      print("\n[EPOCH: {}], \tTest Loss: {:.4f}, \tTest Accuracy: {:.2f} % \n".format(          epoch, test\_loss, test\_accuracy)) |

# testing result

|  |
| --- |
| Train Epoch: 1 [0/60000 (0%)] Train Loss: 3.139114  Train Epoch: 1 [6400/60000 (11%)] Train Loss: 0.327335  Train Epoch: 1 [12800/60000 (21%)] Train Loss: 0.487634  Train Epoch: 1 [19200/60000 (32%)] Train Loss: 0.096294  Train Epoch: 1 [25600/60000 (43%)] Train Loss: 0.271720  Train Epoch: 1 [32000/60000 (53%)] Train Loss: 0.315437  Train Epoch: 1 [38400/60000 (64%)] Train Loss: 0.444940  Train Epoch: 1 [44800/60000 (75%)] Train Loss: 0.229398  Train Epoch: 1 [51200/60000 (85%)] Train Loss: 0.063530  Train Epoch: 1 [57600/60000 (96%)] Train Loss: 0.425345  [EPOCH: 1], Test Loss: 0.1392, Test Accuracy: 95.78 %  Train Epoch: 2 [0/60000 (0%)] Train Loss: 0.130689  Train Epoch: 2 [6400/60000 (11%)] Train Loss: 0.238156  Train Epoch: 2 [12800/60000 (21%)] Train Loss: 0.282381  Train Epoch: 2 [19200/60000 (32%)] Train Loss: 0.267445  Train Epoch: 2 [25600/60000 (43%)] Train Loss: 0.132703  Train Epoch: 2 [32000/60000 (53%)] Train Loss: 0.212889  Train Epoch: 2 [38400/60000 (64%)] Train Loss: 0.223224  Train Epoch: 2 [44800/60000 (75%)] Train Loss: 0.216119  Train Epoch: 2 [51200/60000 (85%)] Train Loss: 0.102152  Train Epoch: 2 [57600/60000 (96%)] Train Loss: 0.092976  [EPOCH: 2], Test Loss: 0.1033, Test Accuracy: 96.88 %  Train Epoch: 10 [51200/60000 (85%)] Train Loss: 0.055804 Train Epoch: 10 [57600/60000 (96%)] Train Loss: 0.062006 [EPOCH: 10], Test Loss: 0.0628, Test Accuracy: 98.14 % |