|  |  |  |  |
| --- | --- | --- | --- |
| **MSPL** | | **Week 3** | **何柏昇** |
| **MNIST** | | | | |
| CNN Model | | | | |
| **#input\_shape=(1,28,28)**  (cnn1): Conv2d(1, 16, kernel\_size=(5, 5), stride=(1, 1))  **#output\_shape=(16,24,24)**  (relu1): ReLU()  (maxpool1): MaxPool2d(kernel\_size=2, stride=2, padding=0,  dilation=1, ceil\_mode=False)  **#output\_shape=(16,12,12)**  (cnn2): Conv2d(16, 32, kernel\_size=(5, 5), stride=(1, 1))  **#output\_shape=(32,8,8)**  (relu2): ReLU()  (maxpool2): MaxPool2d(kernel\_size=2, stride=2, padding=0,  dilation=1, ceil\_mode=False)  **#output\_shape=(32,4,4)**  **#input\_shape=(32\*4\*4)**  (fc1): Linear(in\_features=512, out\_features=10, bias=True) | | | |

|  |
| --- |
| import torch  import torch.nn as nn  import torch.nn.functional as F  from torch.autograd import Variable  from sklearn.model\_selection import train\_test\_split  import matplotlib.pyplot as plt  # Preprocessing  # Loading MNIst dataset  from keras.datasets import mnist  (X\_train, Y\_train), (X\_test, Y\_test) = mnist.load\_data()  # Normalization  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.astype('float32') / 255  # Train : Validation = 8 : 2  features\_train, features\_valid, targets\_train, targets\_valid = train\_test\_split(X\_train, Y\_train, test\_size = 0.2, random\_state = 42)  # Data's type to tensor  featuresTrain = torch.from\_numpy(features\_train)  targetsTrain = torch.from\_numpy(targets\_train).type(torch.LongTensor) # data type is long  featuresValid = torch.from\_numpy(features\_valid)  targetsValid = torch.from\_numpy(targets\_valid).type(torch.LongTensor) # data type is long  featuresTest = torch.from\_numpy(X\_test)  targetsTest = torch.from\_numpy(Y\_test).type(torch.LongTensor) # data type is long  # Pytorch train and test TensorDataset  train = torch.utils.data.TensorDataset(featuresTrain,targetsTrain)  valid = torch.utils.data.TensorDataset(featuresValid,targetsValid)  test = torch.utils.data.TensorDataset(featuresTest,targetsTest)  # Hyper Parameters :batch\_size, epoch and iteration  LR = 0.01  batch\_size = 100  n\_iters = 10000  num\_epochs = n\_iters / (len(features\_train) / batch\_size)  num\_epochs = int(num\_epochs)  # Pytorch DataLoader  train\_loader = torch.utils.data.DataLoader(train, batch\_size = batch\_size, shuffle = True)  valid\_loader = torch.utils.data.DataLoader(valid, batch\_size = batch\_size, shuffle = True)  test\_loader = torch.utils.data.DataLoader(test, batch\_size = batch\_size, shuffle = True)  # Create CNN Model  class CNN\_Model(nn.Module):      def \_\_init\_\_(self):          super(CNN\_Model, self).\_\_init\_\_()          # Convolution 1 , input\_shape=(1,28,28)          self.cnn1 = nn.Conv2d(in\_channels=1, out\_channels=16, kernel\_size=5, stride=1, padding=0) #output\_shape=(16,24,24)          self.relu1 = nn.ReLU() # activation          # Max pool 1          self.maxpool1 = nn.MaxPool2d(kernel\_size=2) #output\_shape=(16,12,12)          # Convolution 2          self.cnn2 = nn.Conv2d(in\_channels=16, out\_channels=32, kernel\_size=5, stride=1, padding=0) #output\_shape=(32,8,8)          self.relu2 = nn.ReLU() # activation          # Max pool 2          self.maxpool2 = nn.MaxPool2d(kernel\_size=2) #output\_shape=(32,4,4)          # Fully connected 1 ,#input\_shape=(32\*4\*4)          self.fc1 = nn.Linear(32 \* 4 \* 4, 10)        def forward(self, x):          # Convolution 1          out = self.cnn1(x)          out = self.relu1(out)          # Max pool 1          out = self.maxpool1(out)          # Convolution 2          out = self.cnn2(out)          out = self.relu2(out)          # Max pool 2          out = self.maxpool2(out)          out = out.view(out.size(0), -1)          # Linear function (readout)          out = self.fc1(out)          return out  # Build the model  model = CNN\_Model()  print(model)  optimizer = torch.optim.Adam(model.parameters(), lr=LR)   # optimize all cnn parameters  loss\_func = nn.CrossEntropyLoss()   # the target label is not one-hotted  input\_shape = (-1,1,28,28)  # Traning the Model  def fit\_model(model, loss\_func, optimizer, input\_shape, num\_epochs, train\_loader, vaild\_loader):      #history-like list for store loss & acc value      training\_loss = []      training\_accuracy = []      validation\_loss = []      validation\_accuracy = []      for epoch in range(num\_epochs):          #training model & store loss & acc / epoch          correct\_train = 0          total\_train = 0          for i, (images, labels) in enumerate(train\_loader):              # 1.Define variables              train = Variable(images.view(input\_shape))              labels = Variable(labels)              # 2.Clear gradients              optimizer.zero\_grad()              # 3.Forward propagation              outputs = model(train)              # 4.Calculate softmax and cross entropy loss              train\_loss = loss\_func(outputs, labels)              # 5.Calculate gradients              train\_loss.backward()              # 6.Update parameters              optimizer.step()              # 7.Get predictions from the maximum value              predicted = torch.max(outputs.data, 1)[1]              # 8.Total number of labels              total\_train += len(labels)              # 9.Total correct predictions              correct\_train += (predicted == labels).float().sum()          #10.store val\_acc / epoch          train\_accuracy = 100 \* correct\_train / float(total\_train)          training\_accuracy.append(train\_accuracy)          # 11.store loss / epoch          training\_loss.append(train\_loss.data)          #evaluate model & store loss & acc / epoch          correct\_test = 0          total\_test = 0          for images, labels in vaild\_loader:              # 1.Define variables              test = Variable(images.view(input\_shape))              # 2.Forward propagation              outputs = model(test)              # 3.Calculate softmax and cross entropy loss              val\_loss = loss\_func(outputs, labels)              # 4.Get predictions from the maximum value              predicted = torch.max(outputs.data, 1)[1]              # 5.Total number of labels              total\_test += len(labels)              # 6.Total correct predictions              correct\_test += (predicted == labels).float().sum()          #6.store val\_acc / epoch          val\_accuracy = 100 \* correct\_test / float(total\_test)          validation\_accuracy.append(val\_accuracy)          # 11.store val\_loss / epoch          validation\_loss.append(val\_loss.data)          print('Train Epoch: {}/{} Traing\_Loss: {} Traing\_acc: {:.6f}% Val\_Loss: {} Val\_accuracy: {:.6f}%'.format(epoch+1, num\_epochs, train\_loss.data, train\_accuracy, val\_loss.data, val\_accuracy))      return training\_loss, training\_accuracy, validation\_loss, validation\_accuracy  # Model fit  training\_loss, training\_accuracy, validation\_loss, validation\_accuracy = fit\_model(model, loss\_func, optimizer, input\_shape, num\_epochs, train\_loader, valid\_loader)  # visualization  plt.plot(range(num\_epochs), training\_loss, 'b-', label='Training\_loss')  plt.plot(range(num\_epochs), validation\_loss, 'g-', label='validation\_loss')  plt.title('Training & Validation loss')  plt.xlabel('Number of epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()  plt.plot(range(num\_epochs), training\_accuracy, 'b-', label='Training\_accuracy')  plt.plot(range(num\_epochs), validation\_accuracy, 'g-', label='Validation\_accuracy')  plt.title('Training & Validation accuracy')  plt.xlabel('Number of epochs')  plt.ylabel('Accuracy')  plt.legend()  plt.show()  # Predict testing data  def predict\_model(test\_loader):      training\_loss = []      training\_accuracy = []      #evaluate model & store loss & acc / epoch      correct\_test = 0      total\_test = 0      for images, labels in test\_loader:          # 1.Define variables          test = Variable(images.view(input\_shape))          # 2.Forward propagation          outputs = model(test)          # 3.Calculate softmax and cross entropy loss          test\_loss = loss\_func(outputs, labels)          # 4.Get predictions from the maximum value          predicted = torch.max(outputs.data, 1)[1]          # 5.Total number of labels          total\_test += len(labels)          # 6.Total correct predictions          correct\_test += (predicted == labels).float().sum()      # Store test\_acc      test\_accuracy = 100 \* correct\_test / float(total\_test)      return test\_loss, test\_accuracy  test\_loss, test\_accuracy = predict\_model(test\_loader)  print("Loss: ",test\_loss,"\tAccuracy: ", test\_accuracy) |

|  |
| --- |
| **Training + Validation** |
|  |
|  |
| **Testing**  Accuracy: 98.4900% |

|  |
| --- |
| **Github** |
| [MSPL/ week3](https://github.com/DanielHo-BS/MSPL/tree/main/week3) |