lenet

# example code

|  |
| --- |
| from google.colab import drive  drive.mount('cookbook')  import torch  if torch.cuda.is\_available():      device = torch.device('cuda:0')  else:      device = torch.device('cpu')  print(device)  # 우선, MNIST dataset에 적용할 transformation 객체를 생성합니다.  from torchvision import transforms  # transformation 정의하기  data\_transform = transforms.Compose([              transforms.Resize((32, 32)),              transforms.ToTensor(),  ])  from six.moves import urllib  opener = urllib.request.build\_opener()  opener.addheaders = [('User-agent', 'Mozilla/5.0')]  urllib.request.install\_opener(opener)  # MNIST training dataset 불러오기  from torchvision import datasets  # 데이터를 저장할 경로 설정  path2data = '/content/data'  # training data 불러오기  train\_data = datasets.MNIST(path2data, train=True, download=True, transform=data\_transform)  # MNIST test dataset 불러오기  val\_data = datasets.MNIST(path2data, train=False, download=True, transform=data\_transform)  # sample images를 확인합니다.  from torchvision import utils  import matplotlib.pyplot as plt  import numpy as np  %matplotlib inline  # training data를 추출합니다.  x\_train, y\_train = train\_data.data, train\_data.targets  # val data를 추출합니다.  x\_val, y\_val = val\_data.data, val\_data.targets  # 차원을 추가하여 B\*C\*H\*W 가 되도록 합니다.  if len(x\_train.shape) == 3:      x\_train = x\_train.unsqueeze(1)  if len(x\_val.shape) == 3:      x\_val = x\_val.unsqueeze(1)  # tensor를 image로 변경하는 함수를 정의합니다.  def show(img):      # tensor를 numpy array로 변경합니다.      npimg = img.numpy()      # C\*H\*W를 H\*W\*C로 변경합니다.      npimg\_tr = npimg.transpose((1,2,0))      plt.imshow(npimg\_tr, interpolation='nearest')  # images grid를 생성하고 출력합니다.  # 총 40개 이미지, 행당 8개 이미지를 출력합니다.  x\_grid = utils.make\_grid(x\_train[:40], nrow=8, padding=2)  show(x\_grid)  # data loader 를 생성합니다.  from torch.utils.data import DataLoader  train\_dl = DataLoader(train\_data, batch\_size=32, shuffle=True)  val\_dl = DataLoader(val\_data, batch\_size=32)  from torch import nn  import torch.nn.functional as F  class LeNet\_5(nn.Module):      def \_\_init\_\_(self):          super(LeNet\_5,self).\_\_init\_\_()          self.conv1 = nn.Conv2d(1, 6, kernel\_size=5, stride=1)          self.conv2 = nn.Conv2d(6, 16, kernel\_size=5, stride=1)          self.conv3 = nn.Conv2d(16, 120, kernel\_size=5, stride=1)          self.fc1 = nn.Linear(120, 84)          self.fc2 = nn.Linear(84, 10)      def forward(self, x):          x = F.tanh(self.conv1(x))          x = F.avg\_pool2d(x, 2, 2)          x = F.tanh(self.conv2(x))          x = F.avg\_pool2d(x, 2, 2)          x = F.tanh(self.conv3(x))          x = x.view(-1, 120)          x = F.tanh(self.fc1(x))          x = self.fc2(x)          return F.softmax(x, dim=1)  model = LeNet\_5()  print(model)  # 모델을 CUDA로 전달합니다.  model.to(device)  print(next(model.parameters()).device)  # 모델 summary를 확인합니다.  from torchsummary import summary  summary(model, input\_size=(1, 32, 32))  # loss function 정의합니다.  loss\_func = nn.CrossEntropyLoss(reduction='sum')  # optimizer 정의합니다.  from torch import optim  opt = optim.Adam(model.parameters(), lr=0.001)  # 현재 lr을 계산하는 함수를 정의합니다.  def get\_lr(opt):      for param\_group in opt.param\_groups:          return param\_group['lr']    # 러닝레이트 스케쥴러를 정의합니다.  from torch.optim.lr\_scheduler import CosineAnnealingLR  lr\_scheduler = CosineAnnealingLR(opt, T\_max=2, eta\_min=1e-05)  # 배치당 performance metric 을 계산하는 함수 정의  def metrics\_batch(output, target):      pred = output.argmax(dim=1, keepdim=True)      corrects = pred.eq(target.view\_as(pred)).sum().item()      return corrects  # 배치당 loss를 계산하는 함수를 정의  def loss\_batch(loss\_func, output, target, opt=None):      loss = loss\_func(output, target)      metric\_b = metrics\_batch(output, target)      if opt is not None:          opt.zero\_grad()          loss.backward()          opt.step()      return loss.item(), metric\_b  # epoch당 loss와 performance metric을 계산하는 함수 정의  def loss\_epoch(model, loss\_func, dataset\_dl, sanity\_check=False, opt=None):      running\_loss = 0.0      running\_metric = 0.0      len\_data = len(dataset\_dl.dataset)      for xb, yb in dataset\_dl:          xb = xb.type(torch.float).to(device)          yb = yb.to(device)          output = model(xb)          loss\_b, metric\_b = loss\_batch(loss\_func, output, yb, opt)          running\_loss += loss\_b          if metric\_b is not None:              running\_metric += metric\_b            if sanity\_check is True: # sanity\_check가 True이면 1epoch만 학습합니다.              break      loss = running\_loss / float(len\_data)      metric = running\_metric / float(len\_data)      return loss, metric  # train\_val 함수 정의  def train\_val(model, params):      num\_epochs = params['num\_epochs']      loss\_func = params['loss\_func']      opt = params['optimizer']      train\_dl = params['train\_dl']      val\_dl = params['val\_dl']      sanity\_check = params['sanity\_check']      lr\_scheduler = params['lr\_scheduler']      path2weights = params['path2weights']      loss\_history = {          'train': [],          'val': [],      }      metric\_history = {          'train': [],          'val': [],      }      # best model parameter를 저장합니다.      best\_model\_wts = copy.deepcopy(model.state\_dict())      best\_loss = float('inf')      for epoch in range(num\_epochs):          current\_lr = get\_lr(opt)          print('Epoch {}/{}, current lr={}'.format(epoch, num\_epochs-1, current\_lr))          model.train()          train\_loss, train\_metric = loss\_epoch(model, loss\_func, train\_dl, sanity\_check, opt)          loss\_history['train'].append(train\_loss)          metric\_history['train'].append(train\_metric)          model.eval()          with torch.no\_grad():              val\_loss, val\_metric = loss\_epoch(model, loss\_func, val\_dl, sanity\_check)              loss\_history['val'].append(val\_loss)              metric\_history['val'].append(val\_metric)          if val\_loss < best\_loss:              best\_loss = val\_loss              best\_model\_wts = copy.deepcopy(model.state\_dict())              torch.save(model.state\_dict(), path2weights)              print('Copied best model weights')          lr\_scheduler.step()          print('train loss: %.6f, dev loss: %.6f, accuracy: %.2f' %(train\_loss, val\_loss, 100\*val\_metric))          print('-'\*10)      # best model을 반환합니다.      model.load\_state\_dict(best\_model\_wts)      return model, loss\_history, metric\_history  import copy  import os  # 학습된 모델의 가중치를 저장할 폴더를 만듭니다.  os.makedirs('/models', exist\_ok=True)  # 하이퍼 파라미터를 설정합니다.  params\_train={   "num\_epochs": 3,   "optimizer": opt,   "loss\_func": loss\_func,   "train\_dl": train\_dl,   "val\_dl": val\_dl,   "sanity\_check": False,   "lr\_scheduler": lr\_scheduler,   "path2weights": "/models/LeNet-5.pt",  }  # 모델을 학습합니다.  model,loss\_hist,metric\_hist=train\_val(model,params\_train)  num\_epochs=params\_train["num\_epochs"]  plt.title("Train-Val Loss")  plt.plot(range(1,num\_epochs+1),loss\_hist["train"],label="train")  plt.plot(range(1,num\_epochs+1),loss\_hist["val"],label="val")  plt.ylabel("Loss")  plt.xlabel("Training Epochs")  plt.legend()  plt.show()  # plot accuracy progress  plt.title("Train-Val Accuracy")  plt.plot(range(1,num\_epochs+1),metric\_hist["train"],label="train")  plt.plot(range(1,num\_epochs+1),metric\_hist["val"],label="val")  plt.ylabel("Accuracy")  plt.xlabel("Training Epochs")  plt.legend()  plt.show() |

# testing result

|  |
| --- |
| Epoch 0/2, current lr=0.001  /usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1795: UserWarning: nn.functional.tanh is deprecated. Use torch.tanh instead. warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh instead.")  Copied best model weights train loss: 1.563984, dev loss: 1.507563, accuracy: 95.71  ---------- Epoch 1/2,  current lr=0.000505 Copied best model weights  train loss: 1.494101,  dev loss: 1.488702,  accuracy: 97.51  ---------- Epoch 2/2,  current lr=1e-05 Copied best model weights  train loss: 1.484444,  dev loss: 1.485315,  accuracy: 97.79  ---------- |