Artificial Intelligence Lab Work (5)

レポート解答用紙 (Report Answer Sheet)

学生証番号 (Student ID): 18521071

名前(Name): ズオン・ミン・ルオン (Duong Minh Luong)

問題1.

|  |
| --- |
| (プログラム)  #ライブラリのインポート  import torch  import torch.nn.functional as F  import torchtext  #データダウンロード  train\_iter,test\_iter=torchtext.datasets.IMDB(split=('train','test'))  #トークナイザー  tokenizer=torchtext.data.utils.get\_tokenizer('basic\_english')  #MODELNAMEはモデルファイルを保存するためのファイル名です。  MODELNAME='imdb-rnn.model'  #EPOCH数はここでは10にしておきます。  EPOCH=10  #BATCHSIZEはミニバッチサイズですが、ここでは64にしておきます。  BATCHSIZE=64  #LRは学習率で、1e‐5=0.00001にしています。  LR=1e-5  #DEVICEはCPUかGPUのどちらを使うかの指定になります。  DEVICE="cuda" if torch.cuda.is\_available() else "cpu"  #訓練データの読み出しとトークン化  train\_data=[(label,tokenizer(line)) for label,line in train\_iter]  train\_data.sort(key=lambda x: len(x[1])) #訓練データのソーティング  #テストデータの読み出しとトークン化とソーティング  test\_data=[(label,tokenizer(line)) for label,line in test\_iter]  test\_data.sort(key=lambda x: len(x[1]))  for i in range(10):  print(train\_data[i])  #この部分はなくても良いが、表示させて おくとtrain\_dataの中身がどうなっているのかよくわかる  def make\_vocab(train\_data,min\_freq):  vocab={}  for label,tokenlist in train\_data:  for token in tokenlist: #辞書を使ってトークンの出現回数をカウント  if token not in vocab:  vocab[token]=0  vocab[token]+=1  #語彙リストの0〜3番⽬を<unk>,<pad>, <cls>, <eos>で予約  vocablist=[('<unk>',0),('<pad>',0),('<cls>',0),('<eos>',0)]  vocabidx={}  for token , freq in vocab.items():  if freq >= min\_freq:  idx=len(vocablist)  vocablist.append((token,freq))  vocabidx[token]=idx  vocabidx['<unk>']=0  vocabidx['<pad>']=1  vocabidx['<cls>']=2  vocabidx['<eos>']=3  return vocablist,vocabidx  #今回はmin\_freqを10に設定  vocablist,vocabidx=make\_vocab(train\_data,10)  def preprocess(data,vocabidx):  rr=[]  for label,tokenlist in data:  tkl=['<cls>'] #テキストの先頭に<cls>トークンを追加  for token in tokenlist:  tkl.append(token if token in vocabidx else '<unk>')  tkl.append('<eos>') #テキストの末尾に<eos>トークンを追加  rr.append((label,tkl))  return rr  train\_data = preprocess(train\_data,vocabidx)  test\_data = preprocess(test\_data,vocabidx)  for i in range(10):  print(train\_data[i])  def make\_batch(data,batchsize):  bb=[]  blabel=[]  btokenlist=[]  for label,tokenlist in data:  blabel.append(label)  #ラベルおよびトークン列ごとにまとめる  btokenlist.append(tokenlist)  if len(blabel) >= batchsize:  #バッチサイズと同じ⼤きさになったら、たまったバッチデータをbbに追加  bb.append((btokenlist,blabel))  blabel=[]  btokenlist=[]  if len(blabel)>0:  #残ったlabelと tokenlistを忘れずにbbに追加  bb.append((btokenlist,blabel))  return bb  train\_data=make\_batch(train\_data,BATCHSIZE)  test\_data=make\_batch(test\_data,BATCHSIZE)  for i in range(10):  print(train\_data[i])  def padding(bb):  for tokenlists,labels in bb:  #ミニバッチ内で⼀番⻑いトークン列の⻑さを得る  maxlen = max([len(x) for x in tokenlists])  for tkl in tokenlists:  for i in range(maxlen- len(tkl)):  #最⼤⻑と同じ⻑さになるように<pad>トークンの追加を繰り返す  tkl.append('<pad>')  return bb  train\_data=padding(train\_data)  test\_data=padding(test\_data)  for i in range(10):  print(train\_data[i])  def word2id(bb,vocabidx):  rr= []  for tokenlists,labels in bb:  #ラベルはpos→1, neg→0  id\_labels = [1 if label == 'pos' else 0 for label in labels]  id\_tokenlists=[]  for tokenlist in tokenlists:  #語彙インデックスを使ってトークンをID化  id\_tokenlists.append([vocabidx[token] for token in tokenlist])  rr.append((id\_tokenlists,id\_labels))  return rr  train\_data= word2id(train\_data,vocabidx)  test\_data=word2id(test\_data,vocabidx)  for i in range(10):  print(train\_data[i])  class MyRNN(torch.nn.Module):  def \_\_init\_\_(self):  super(MyRNN,self).\_\_init\_\_()  vocabsize= len(vocablist)  self.emb=torch.nn.Embedding(vocabsize,300,padding\_idx=vocabidx['<pad>'])  self.l1=torch.nn.Linear(300,300)  self.l2=torch.nn.Linear(300,2)  def forward(self,x):  e=self.emb(x)#全てのトークンをまとめて埋め込み  #中間状態ℎの初期化  h=torch.zeros(e[0].size(),dtype=torch.float32).to(DEVICE)  #テキスト先頭から順に処理  for i in range(x.size()[0]):  h=F.relu(e[i] + self.l1(h))  return self.l2(h)  def train():  model = MyRNN().to(DEVICE) #＃モデルを定無してDEVICEにのせる  optimizer=torch.optim.Adam(model.parameters(),lr=LR) #＃最適化にAdamを使う  for epoch in range(EPOCH): ##EPOCH回最進化を行う  loss=0  for tokenlists,labels in train\_data:  tokenlists=torch.tensor(tokenlists,dtype= torch.int64).transpose(0,1).to(DEVICE)  labels=torch.tensor(labels,dtype=torch.int64).to(DEVICE)  optimizer.zero\_grad() #＃勾配の初期化  y=model(tokenlists) #forward計算  batchloss=F.cross\_entropy(y,labels)#損失の計算  batchloss.backward() #逆伝搬の計算  optimizer.step() #亜み変数の更新  loss=loss+batchloss.item()  print('Epoch',epoch, ":loss", loss)  torch.save(model.state\_dict(),MODELNAME)  def test():  total=0  correct=0  model=MyRNN().to(DEVICE)  model.load\_state\_dict(torch.load(MODELNAME))  model.eval()  for tokenlists,labels in test\_data:  total+= len(labels)  tokenlists=torch.tensor(tokenlists,dtype=torch.int64).transpose(0,1).to(DEVICE)  labels = torch.tensor(labels,dtype=torch.int64).to(DEVICE)  y=model(tokenlists)  pred\_labels=y.max(dim=1)[1]  correct+= (pred\_labels==labels).sum()  print("correct: ",correct.item())  print('Total: ',total)  print("accuracy: ",(correct.item()/float(total)))  train()  test() |

|  |
| --- |
| (実行結果) |