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長いトークン列の長さを得る

    maxlen = max([len(x) for x in tokenlists])
    for tkl in tokenlists:
      for i in range(maxlen- len(tkl)):

        #最大長と同じ長さになるように<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(DEV
ICE)
    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）

```
[24] train()
```

```
Epoch 0 :loss 255.40932920575142
Epoch 1 :loss 241.10407587885857
Epoch 2 :loss 239.03494316339493
Epoch 3 :loss 238.0786054134369
Epoch 4 :loss 237.42557755112648
Epoch 5 :loss 236.93611961603165
Epoch 6 :loss 236.5265814960003
Epoch 7 :loss 236.15778723359108
Epoch 8 :loss 235.82375440001488
Epoch 9 :loss 235.50289618968964
```

```
[27] test()
```

```
correct:  17089
Total:  25000
accuracy:  0.68356
```