Artificial Intelligence Lab Work (6)
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)

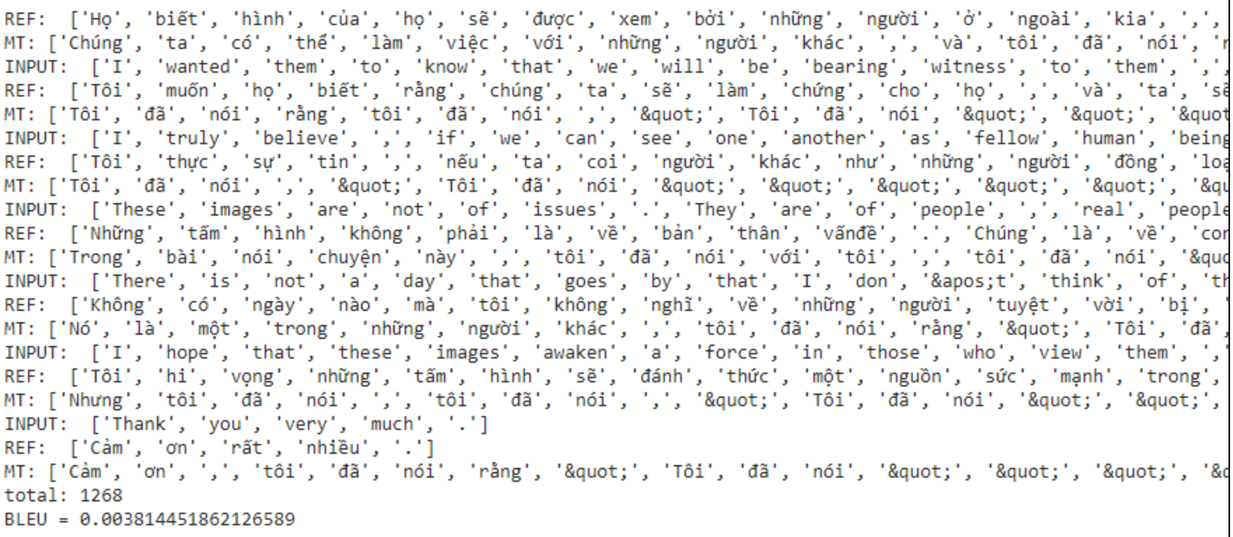
```
import tarfile
def iwlt15(train_test):
  url= "https://github.com/stefan-it/nmt-en-vi/raw/master/data/"
  r= requests.get(url + train_test+"-en-vi.tgz")
  filename = train_test + "-en-vi.tar.gz"
  with open(filename,'wb') as f:
    f.write(r.content)
    tarfile.open(filename,'r:gz').extractall("iwslt15")
  iwslt15("train")
  iwslt15("test-2013")

  f= open("iwslt15/train.en")
  train_en=[line.split() for line in f]
  f.close()
  f= open("iwslt15/train.vi")
  train_vi=[line.split() for line in f]
  f.close()
  f= open("iwslt15/tst2013.en")
  test_en=[line.split() for line in f]
  f.close()
  f= open("iwslt15/tst2013.vi")
  test_vi=[line.split() for line in f]
  f.close()

import requests
import torch
import torch.nn.functional as F
import torchtext

url = 'https://nlp.stanford.edu/projects/nmt/data/iwslt15.en-vi/'
```

```
train_en = [line.split() for line in
requests.get(url+"train.en").text.splitlines()]
train_vi = [line.split() for line in
requests.get(url+"train.vi").text.splitlines()]
test_en = [line.split() for line in
requests.get(url+"tst2013.en").text.splitlines()]
test_vi = [line.split() for line in
requests.get(url+"tst2013.vi").text.splitlines()]

for i in range(10):
    print(train_en[i])
    print(train_vi[i])

print("# line", len(train_en), len(train_vi), len(test_en), len(test_vi))

MODELNAME = 'iwslt15-en-vi-rnn.model'
EPOCH = 10
BATCHSIZE = 128
LR = 0.0001
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'

"""##1. Data preparation

###Make vocab
"""

def make_vocab(train_data, min_freq):
    vocab = {} # tap tu vung
    for 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)
      #min_freq以上の出現回数のトークンだけ語彙リスト(vocablist)と語彙インデックス
(vocabidx)に登録
            vocablist.append((token, freq))
            vocabidx[token] = idx
    #<unk>,<pad>,<cls>,<eos>のインデックス登録
    vocabidx['<unk>'] = 0
    vocabidx['<pad>'] = 1
    vocabidx['<cls>'] = 2
    vocabidx['<eos>'] = 3

    return vocablist, vocabidx
#今回はmin_freqを3に設定
vocablist_en, vocabidx_en = make_vocab(train_en, 3)
vocablist_vi, vocabidx_vi = make_vocab(train_vi, 3)

print("vocab size en :", len(vocablist_en))
print("vocab size vi :", len(vocablist_vi))

"""###Pre-proces"""

def preprocess(data, vocabidx):
    rr = []
    for tokenlist in data:
    #テキストの先頭に<cls>トークンを追加
        tkl = ['<cls>']
        for token in tokenlist:
          #トークンの追加。語彙リストに無いトークンは<unk>に変換
            tkl.append(token if token in vocabidx else '<unk>')
        #テキストの末尾に<eos>トークンを追加
        tkl.append('<eos>')
        rr.append(tkl)
    return rr

train_en_prep = preprocess(train_en, vocabidx_en)
train_vi_prep = preprocess(train_vi, vocabidx_vi)
```

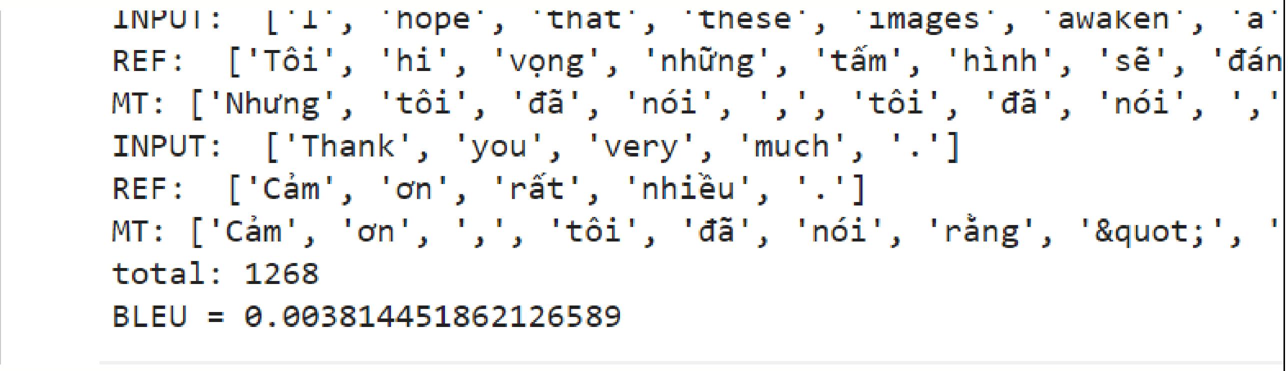
```
test_en_prep = preprocess(test_en, vocabidx_en)
#この部分はあっても無くても良い
for i in range(5):
    print(train_en_prep[i])
    print(train_vi_prep[i])
    print(test_en_prep[i])

#訓練データのzip化(前処理済en, 前処理済vi)
train_data = list(zip(train_en_prep, train_vi_prep))
#訓練データのソーティング
train_data.sort(key = lambda x: (len(x[0]), len(x[1])))
#テストデータのZip化(前処理済en, en, vi)
test_data = list(zip(test_en_prep, test_en, test_vi))
#この部分はあっても無くても良い
for i in range(5):
    print(train_data[i])
for i in range(5):
    print(test_data[i])

"""###Make batch"""

def make_batch(data, batchsize):
    bb = []
    ben = []
    bvi = []
    for en, vi in data:
      #英語文だけbenにまとめ、ベトナム語文だけbviにまとめる
        ben.append(en)
        bvi.append(vi)
        if len(ben) >= 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((ben, bvi))
            ben = []
            bvi = []
    if len(ben) > 0:
      #残ったbenとbviを忘れずにbbに追加
```

```
        bb.append((ben, bvi))
    return bb

train_data = make_batch(train_data, BATCHSIZE)
for i in range(5):
    print(train_data[i])

"""###Padding batch"""

def padding_batch(b):
  #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 b])
    for tokenlists in b:
        for i in range(maxlen - len(tokenlists)):
          #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
            tokenlists.append('<pad>')

    return b

def padding(bb):
    for ben, bvi in bb:
      #英語バッチとベトナム語バッチと両方にパディングをする
        ben = padding_batch(ben)
        bvi = padding_batch(bvi)
    return bb

padding(train_data)
for i in range(3):
    print(train_data[i])

"""###Encoding"""

#語彙インデックスを使ってトークンを ID 化
train_data=[([[vocabidx_en[token] for token in tokenlist] for tokenlist in
ben],
            [[vocabidx_vi[token] for token in tokenlist] for tokenlist in
bvi]) for ben, bvi in train_data]
test_data=[([vocabidx_en[token] for token in enprep],en ,vi) for enprep,
en, vi in test_data]

for i in range(3):
    print(train_data[i])
for i in range(3):
    print(test_data[i])

"""##2. RNN Model"""

class RNNEncDec(torch.nn.Module):
    def __init__(self, vocablist_x, vocabidx_x, vocablist_y, vocabidx_y):
        super(RNNEncDec, self).__init__()
        #encemb: エンコーダーの畳み込み層(300 次元)
        self.encemb = torch.nn.Embedding(len(vocablist_x), 300, padding_idx
= vocabidx_x['<pad>'])
        #encrnn: エンコーダーの RNN 計算ユニット(300×300 の FC 層)
        self.encrnn = torch.nn.Linear(300, 300)
        #decemb: デコーダーの畳み込み層(300 次元)
        self.decemb = torch.nn.Embedding(len(vocablist_x), 300, padding_idx
= vocabidx_y['<pad>'])
        #decrnn: デコーダーの RNN 計算ユニット(300×300 の FC 層)
        self.decrnn = torch.nn.Linear(300, 300)
        #decout: デコーダーの出力(300×目的言語の語彙数)
        self.decout = torch.nn.Linear(300, len(vocabidx_y))

    def forward(self, x):

      #x: input (文長×バッチサイズ)


      #y:output (文長×バッチサイズ)

        x, y = x[0], x[1]
        # encoder
        e_x = self.encemb(x)
        n_x = e_x.size()[0]
        h = torch.zeros(300, dtype = torch.float32).to(DEVICE)
        for i in range(n_x):
            h = F.relu(e_x[i] + self.encrnn(h))
        # decoder
        e_y = self.decemb(y)
        #n_y=文長(J)+2 (<cls>と <eos>)
        n_y = e_y.size()[0]
        loss = torch.tensor(0, dtype = torch.float32).to(DEVICE)
        for i in range(n_y - 1):
            #入力は i=0 から J まで
            h = F.relu(e_y[i] + self.decrnn(h))
            #出力は i=1 から J+1 まで
            loss += F.cross_entropy(self.decout(h), y[i+1])

        return loss

    def evaluate(self, x, vocablist_y, vocabidx_y):
        # encoder
        #推論は 1文ずつ行うので、x には文長×バッチサイズ 1 のミニバッチが入っている。
        e_x = self.encemb(x)
        n_x = e_x.size()[0]
        #エンコーダー部は forward とほぼ同じ。
        h = torch.zeros(300, dtype = torch.float32).to(DEVICE)
        for i in range(n_x):
            h = F.relu(e_x[i] + self.encrnn(h))
        # decoder
        #デコーダーの入力(バッチサイズ 1)を作る。最初は<cls>トークンを入力する
        y = torch.tensor([vocabidx_y['<cls>']]).to(DEVICE)
        e_y = self.decemb(y)
        pred = []
        for i in range(30):
            h = F.relu(e_y + self.decrnn(h))
            pred_id = self.decout(h).squeeze().argmax()
            #pred_id が予測する出力単語 ID pred_id が<eos>の ID と等しければ推論終了
            if pred_id == vocabidx_y['<eos>']:
                break
            pred_y = vocablist_y[pred_id][0]
            pred.append(pred_y)
            #デコーダーは 1 単語ずつ処理をし、得られた出力を次の入力とする
            y[0] = pred_id
            e_y = self.decemb(y)

        return pred

def train():
    model = RNNEncDec(vocablist_en, vocabidx_en, vocablist_vi,
vocabidx_vi).to(DEVICE)
    optimizer = torch.optim.Adam(model.parameters(), lr = LR)
    for epoch in range(EPOCH):
        loss = 0
        step = 0
        for ben, bvi in train_data:
            ben = torch.tensor(ben, dtype =
torch.int64).transpose(0,1).to(DEVICE)
            bvi = torch.tensor(bvi, dtype =
torch.int64).transpose(0,1).to(DEVICE)

            optimizer.zero_grad()
            batchloss = model((ben, bvi))
            batchloss.backward()
            optimizer.step()

            loss = loss + batchloss.item()

            if step % 100 == 0:
                print("step {}, batchloss = {}".format(step,
batchloss.item()))
            step += 1
        print("Epoch {} with loss = {}".format(epoch, loss))
    torch.save(model.state_dict(), MODELNAME)

def test():
    total = 0
    correct = 0
    model = RNNEncDec(vocablist_en, vocabidx_en, vocablist_vi,
vocabidx_vi).to(DEVICE)
    model.load_state_dict(torch.load(MODELNAME))
    model.eval()
    ref = []
    pred = []
#テストデータはミニバッチ化されていないので、1文ずつ処理をする。(enprep,en, vi)にそれぞれ

#前処理済み英文、英文、ベトナム語文がはいっている
    for enprep, en, vi in test_data:
        input = torch.tensor([enprep], dtype =
torch.int64).transpose(0,1).to(DEVICE)
        p = model.evaluate(input, vocablist_vi, vocabidx_vi)
        print("INPUT: ", en)
        print("REF: ", vi)
        print("MT:", p)
        ref.append([vi])
        pred.append(p)
    bleu = torchtext.data.metrics.bleu_score(pred, ref)
    print("total: {}".format(len(test_data)))
    print("BLEU = {}".format(bleu))
train()
test()
```

（実行結果）



```
REF:  ['Họ', 'biết', 'hình', 'của', 'họ', 'sẽ', 'được', 'xem', 'bởi', 'những', 'người', 'ở', 'ngoài', 'kia', ',',
MT: ['Chúng', 'ta', 'có', 'thể', 'làm', 'việc', 'với', 'những', 'người', 'khác', ',', 'và', 'tôi', 'đã', 'nói', 'r
INPUT:  ['I', 'wanted', 'them', 'to', 'know', 'that', 'we', 'will', 'be', 'bearing', 'witness', 'to', 'them', ',',
REF:  ['Tôi', 'muốn', 'họ', 'biết', 'rằng', 'chúng', 'ta', 'sẽ', 'làm', 'chứng', 'cho', 'họ', ',', 'và', 'ta', 'sẽ
MT: ['Tôi', 'đã', 'nói', 'rằng', 'tôi', 'đã', 'nói', ',', '&quot;', 'Tôi', 'đã', 'nói', '&quot;', '&quot;', '&quot
INPUT:  ['I', 'truly', 'believe', ',', 'if', 'we', 'can', 'see', 'one', 'another', 'as', 'fellow', 'human', 'being
REF:  ['Tôi', 'thực', 'sự', 'tin', ',', 'nếu', 'ta', 'coi', 'người', 'khác', 'như', 'những', 'người', 'đồng', 'loạ
MT: ['Tôi', 'đã', 'nói', ',', '&quot;', 'Tôi', 'đã', 'nói', '&quot;', '&quot;', '&quot;', '&quot;', '&quot;', '&qu
INPUT:  ['These', 'images', 'are', 'not', 'of', 'issues', '.', 'They', 'are', 'of', 'people', ',', 'real', 'people
REF:  ['Những', 'tấm', 'hình', 'không', 'phải', 'là', 'về', 'bản', 'thân', 'vấnđề', '.', 'Chúng', 'là', 'về', 'con
MT: ['Trong', 'bài', 'nói', 'chuyện', 'này', ',', 'tôi', 'đã', 'nói', 'với', 'tôi', ',', 'tôi', 'đã', 'nói', '&quo
INPUT:  ['There', 'is', 'not', 'a', 'day', 'that', 'goes', 'by', 'that', 'I', 'don', '&apos;t', 'think', 'of', 'th
REF:  ['Không', 'có', 'ngày', 'nào', 'mà', 'tôi', 'không', 'nghĩ', 'về', 'những', 'người', 'tuyệt', 'vời', 'bị', '
MT: ['Nó', 'là', 'một', 'trong', 'những', 'người', 'khác', ',', 'tôi', 'đã', 'nói', 'rằng', '&quot;', 'Tôi', 'đã',
INPUT:  ['I', 'hope', 'that', 'these', 'images', 'awaken', 'a', 'force', 'in', 'those', 'who', 'view', 'them', ','
REF:  ['Tôi', 'hi', 'vọng', 'những', 'tấm', 'hình', 'sẽ', 'đánh', 'thức', 'một', 'nguồn', 'sức', 'mạnh', 'trong',
MT: ['Nhưng', 'tôi', 'đã', 'nói', ',', 'tôi', 'đã', 'nói', ',', '&quot;', 'Tôi', 'đã', 'nói', '&quot;', '&quot;',
INPUT:  ['Thank', 'you', 'very', 'much', '.']
REF:  ['Cảm', 'ơn', 'rất', 'nhiều', '.']
MT: ['Cảm', 'ơn', ',', 'tôi', 'đã', 'nói', 'rằng', '&quot;', 'Tôi', 'đã', 'nói', '&quot;', '&quot;', '&quot;', '&q
total: 1268
BLEU = 0.003814451862126589
```



```
INPUT:  ['I', 'hope', 'that', 'these', 'images', 'awaken', 'a'
REF:  ['Tôi', 'hi', 'vọng', 'những', 'tấm', 'hình', 'sẽ', 'đán
MT: ['Nhưng', 'tôi', 'đã', 'nói', ',', 'tôi', 'đã', 'nói', ','
INPUT:  ['Thank', 'you', 'very', 'much', '.']
REF:  ['Cảm', 'ơn', 'rất', 'nhiều', '.']
MT: ['Cảm', 'ơn', ',', 'tôi', 'đã', 'nói', 'rằng', '&quot;', '
total: 1268
BLEU = 0.003814451862126589
```

問題 2.

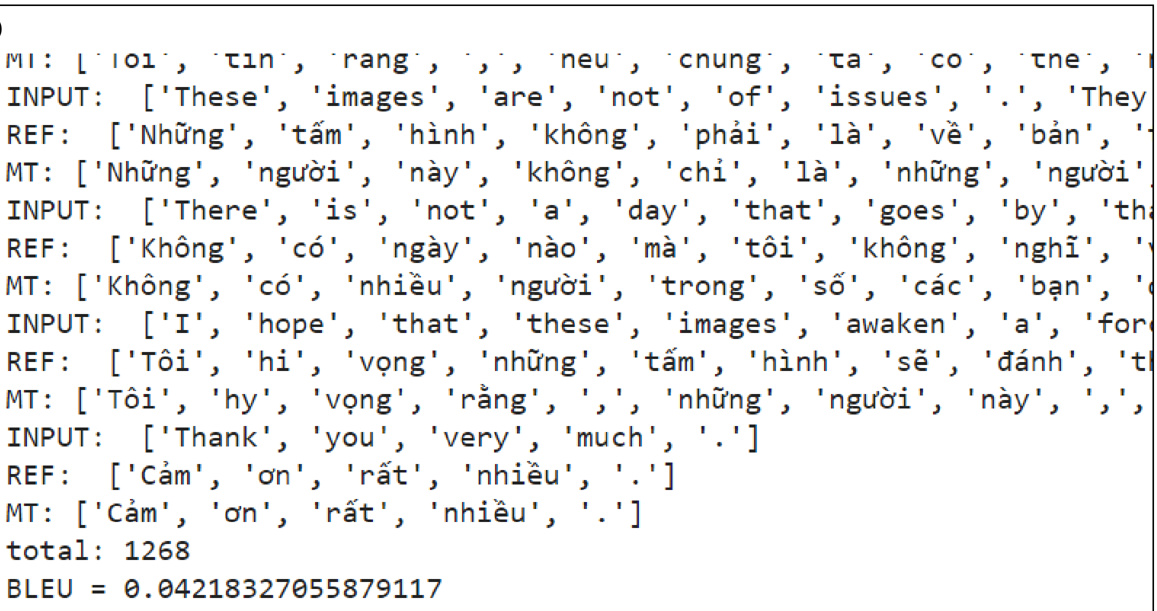
(プログラム)

```
import requests
import torch
import torch.nn.functional as F
import torchtext

url = 'https://nlp.stanford.edu/projects/nmt/data/iwslt15.en-vi/'
train_en = [line.split() for line in
requests.get(url+"train.en").text.splitlines()]
train_vi = [line.split() for line in
requests.get(url+"train.vi").text.splitlines()]
test_en = [line.split() for line in
requests.get(url+"tst2013.en").text.splitlines()]
test_vi = [line.split() for line in
requests.get(url+"tst2013.vi").text.splitlines()]

for i in range(10):
   print(train_en[i])
   print(train_vi[i])

print("# line", len(train_en), len(train_vi), len(test_en), len(test_vi))

MODELNAME = 'LSTM_Dropout-translation'
EPOCH = 10
BATCHSIZE = 128
LR = 0.001
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'

"""##1. Data preparation

###Make vocab
"""

def make_vocab(train_data, min_freq):
   vocab = {} # tap tu vung
   for 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)
      #min_freq以上の出現回数のトークンだけ語彙リスト(vocablist)と語彙インデックス
(vocabidx)に登録
            vocablist.append((token, freq))
            vocabidx[token] = idx
    #<unk>,<pad>,<cls>,<eos>のインデックス登録
    vocabidx['<unk>'] = 0
    vocabidx['<pad>'] = 1
    vocabidx['<cls>'] = 2
    vocabidx['<eos>'] = 3

    return vocablist, vocabidx
#今回はmin_freqを3に設定
vocablist_en, vocabidx_en = make_vocab(train_en, 3)
vocablist_vi, vocabidx_vi = make_vocab(train_vi, 3)

print("vocab size en :", len(vocablist_en))
print("vocab size vi :", len(vocablist_vi))

"""###Pre-proces"""

def preprocess(data, vocabidx):
    rr = []
    for tokenlist in data:
    #テキストの先頭に<cls>トークンを追加
        tkl = ['<cls>']
        for token in tokenlist:
          #トークンの追加。語彙リストに無いトークンは<unk>に変換
```

```
            tkl.append(token if token in vocabidx else '<unk>')
        #テキストの末尾に<eos>トークンを追加
        tkl.append('<eos>')
        rr.append(tkl)
    return rr

train_en_prep = preprocess(train_en, vocabidx_en)
train_vi_prep = preprocess(train_vi, vocabidx_vi)
test_en_prep = preprocess(test_en, vocabidx_en)
#この部分はあっても無くても良い
for i in range(5):
    print(train_en_prep[i])
    print(train_vi_prep[i])
    print(test_en_prep[i])

#訓練データのzip化(前処理済en, 前処理済vi)
train_data = list(zip(train_en_prep, train_vi_prep))
#訓練データのソーティング
train_data.sort(key = lambda x: (len(x[0]), len(x[1])))
#テストデータのZip化(前処理済en, en, vi)
test_data = list(zip(test_en_prep, test_en, test_vi))
#この部分はあっても無くても良い
for i in range(5):
    print(train_data[i])
for i in range(5):
    print(test_data[i])

"""##Make batch"""

def make_batch(data, batchsize):
    bb = []
    ben = []
    bvi = []
    for en, vi in data:

      #英語文だけbenにまとめ、ベトナム語文だけbviにまとめる

        ben.append(en)
        bvi.append(vi)
        if len(ben) >= 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((ben, bvi))
            ben = []
            bvi = []
    if len(ben) > 0:
        #残った ben と bvi を忘れずに bb に追加
        bb.append((ben, bvi))
    return bb

train_data = make_batch(train_data, BATCHSIZE)
for i in range(5):
    print(train_data[i])

"""##Padding batch"""

def padding_batch(b):

    #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 b])
    for tokenlists in b:
        for i in range(maxlen - len(tokenlists)):

            #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

            tokenlists.append('<pad>')

    return b

def padding(bb):
    for ben, bvi in bb:

        #英語バッチとベトナム語バッチと両方にパディングをする

        ben = padding_batch(ben)
        bvi = padding_batch(bvi)
    return bb

train_data_pd = padding(train_data)
```

```
for i in range(3):
    print(train_data_pd[i])

"""##Encoding"""

#語彙インデックスを使ってトークンを ID 化
train_data_encoding=[([[vocabidx_en[token] for token in tokenlist] for
tokenlist in ben],
             [[vocabidx_vi[token] for token in tokenlist] for tokenlist in
bvi]) for ben, bvi in train_data]
test_data_encoding=[([vocabidx_en[token] for token in enprep],en ,vi) for
enprep, en, vi in test_data]

for i in range(3):
    print(train_data_encoding[i])
    print(test_data_encoding[i])

"""##2. LSTM + dropout"""

class LSTM(torch.nn.Module):
    def __init__(self, vocablist_x, vocabidx_x, vocablist_y, vocabidx_y):
        super(LSTM, self).__init__()
        self.encemb = torch.nn.Embedding(len(vocablist_x), 256, padding_idx
= vocabidx_x['<pad>'])
        self.dropout = torch.nn.Dropout(0.5)
        self.enclstm = torch.nn.LSTM(256,516,2,dropout=0.5)
        self.decemb = torch.nn.Embedding(len(vocablist_x), 256, padding_idx
= vocabidx_y['<pad>'])
        self.declstm = torch.nn.LSTM(256,516,2,dropout=0.5)
        self.decout = torch.nn.Linear(516, len(vocabidx_y))

    def forward(self,x):
        x, y = x[0], x[1]

        e_x = self.dropout(self.encemb(x))

        outenc,(hidden,cell) = self.enclstm(e_x)
        n_y=y.shape[0]
        outputs = torch.zeros(n_y,BATCHSIZE,len(vocablist_vi)).to(DEVICE)
        loss = torch.tensor(0.,dtype=torch.float32).to(DEVICE)
        for i in range(n_y-1):
            input = y[i]
            input = input.unsqueeze(0)
            input = self.dropout(self.decemb(input))
            outdec, (hidden,cell) = self.declstm(input,(hidden,cell))
            output = self.decout(outdec.squeeze(0))
            input = y[i+1]
            loss += F.cross_entropy(output, y[i+1])
        return loss

    def evaluate(self,x,vocablist_y,vocabidx_y):
        e_x = self.dropout(self.encemb(x))
        outenc,(hidden,cell)=self.enclstm(e_x)

        y = torch.tensor([vocabidx_y['<cls>']]).to(DEVICE)
        pred=[]
        for i in range(30):
            input = y
            input = input.unsqueeze(0)
            input = self.dropout(self.decemb(input))
            outdec,(hidden,cell)= self.declstm(input,(hidden,cell))
            output = self.decout(outdec.squeeze(0))
            pred_id = output.squeeze().argmax().item()
            if pred_id == vocabidx_y['<eos>']:
                break
            pred_y = vocablist_y[pred_id][0]
            pred.append(pred_y)
            y[0]=pred_id
            input=y
        return pred

def train():
    model = LSTM(vocablist_en, vocabidx_en, vocablist_vi,
vocabidx_vi).to(DEVICE)
    optimizer = torch.optim.Adam(model.parameters(), lr = LR)
    for epoch in range(EPOCH):
        loss = 0
        step = 0
        for ben, bvi in train_data_encoding:
            ben = torch.tensor(ben, dtype =
torch.int64).transpose(0,1).to(DEVICE)
            bvi = torch.tensor(bvi, dtype =
torch.int64).transpose(0,1).to(DEVICE)

            optimizer.zero_grad()
            batchloss = model((ben, bvi))
            batchloss.backward()
            optimizer.step()

            loss = loss + batchloss.item()

            if step % 100 == 0:
                print("step {}, batchloss = {}".format(step,
batchloss.item()))
            step += 1
        print("Epoch {} with loss = {}".format(epoch, loss))
    torch.save(model.state_dict(), MODELNAME)

def test():
    total = 0
    correct = 0
    model = LSTM(vocablist_en, vocabidx_en, vocablist_vi,
vocabidx_vi).to(DEVICE)
    model.load_state_dict(torch.load(MODELNAME))
    model.eval()
    ref = []
    pred = []

    for enprep, en, vi in test_data_encoding:
        input = torch.tensor([enprep], dtype =
torch.int64).transpose(0,1).to(DEVICE)
        p = model.evaluate(input, vocablist_vi, vocabidx_vi)
        print("INPUT: ", en)
        print("REF: ", vi)
        print("MT:", p)

        ref.append([vi])
        pred.append(p)
    bleu = torchtext.data.metrics.bleu_score(pred, ref)
    print("total: {}".format(len(test_data)))
    print("BLEU = {}".format(bleu))

train()

test()
```

（実行結果）



```
MT: ['Tôi', 'tin', 'rằng', ',', 'nếu', 'chúng', 'ta', 'có', 'thể', '
INPUT:  ['These', 'images', 'are', 'not', 'of', 'issues', '.', 'They
REF:  ['Những', 'tấm', 'hình', 'không', 'phải', 'là', 'về', 'bản', '
MT: ['Những', 'người', 'này', 'không', 'chỉ', 'là', 'những', 'người'
INPUT:  ['There', 'is', 'not', 'a', 'day', 'that', 'goes', 'by', 'th
REF:  ['Không', 'có', 'ngày', 'nào', 'mà', 'tôi', 'không', 'nghĩ', '
MT: ['Không', 'có', 'nhiều', 'người', 'trong', 'số', 'các', 'bạn', '
INPUT:  ['I', 'hope', 'that', 'these', 'images', 'awaken', 'a', 'for
REF:  ['Tôi', 'hi', 'vọng', 'những', 'tấm', 'hình', 'sẽ', 'đánh', 't
MT: ['Tôi', 'hy', 'vọng', 'rằng', ',', 'những', 'người', 'này', ',',
INPUT:  ['Thank', 'you', 'very', 'much', '.']
REF:  ['Cảm', 'ơn', 'rất', 'nhiều', '.']
MT: ['Cảm', 'ơn', 'rất', 'nhiều', '.']
total: 1268
BLEU = 0.04218327055879117
```