Importing libraries

```
import numpy as np
import pandas as pd
import math
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence,pad_sequence
from torch.utils.data import Dataset, DataLoader
from torch.optim.lr_scheduler import StepLR
import random
```

Initialize common variables

```
from google.colab import drive
drive.mount('/content/drive')

torch.manual_seed(0)
random.seed(0)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

## Files
input_file_path = "./sample_data/train"
dev_file_path = "./sample_data/dev"
test_file_path = "./sample_data/test"
glove_file_path = './sample_data/glove.6B.100d.gz'

## Other constants
unk_token = '<UNK>'
pad_token = '<MUM>'

## Outfiles
dev1_out_file = "dev1.out"
test1_out_file = "test1.out"
dev2_out_file = "dev2.out"
test2_out_file = "test2.out"
```

```
Mounted at /content/drive
```

Creating a dataframe for train, dev and test files

```
## Train file
tr_df, v_df, te_df  = [], [], []
with open(input_file_path, 'r') as f:
    for data in f.readlines():
        if len(data) > 2:
            idx, term, ner = data.strip().split(" ")
            tr_df.append([idx, term, ner])
tr_df = pd.DataFrame(tr_df, columns=['id', 'word', 'NER'])


with open(dev_file_path, 'r') as f:
    for data in f.readlines():
        if len(data) > 2:
            idx, term, ner = data.strip().split(" ")
            v_df.append([idx, term, ner])
v_df = pd.DataFrame(v_df, columns=['id', 'word', 'NER'])


with open(test_file_path, 'r') as f:
    for data in f.readlines():
        if len(data) > 1:
            idx, term = data.strip().split(" ")
            te_df.append([idx, term])
te_df = pd.DataFrame(te_df, columns=['id', 'word'])
```

Drop all null values from dataframes

```
tr_df = tr_df.dropna()
v_df = v_df.dropna()
te_df = te_df.dropna()
```

Make data list from Train, Dev and Test data

```
# Train
tr_x, tr_y, x, y = [],[],[],[]
first = 0
for row in tr_df.itertuples():
    if(row.id == '1' and first == 1):
        tr_y.append(y)
        tr_x.append(x)
        x, y = [], []
    first = 1
    y.append(row.NER)
    x.append(row.word)
tr_y.append(y)
tr_x.append(x)

# Dev
v_x, v_y, x, y = [],[],[],[]
first = 0
for row in v_df.itertuples():
    if(row.id == '1' and first == 1):
        v_y.append(y)
        v_x.append(x)
        x, y = [], []
    first = 1
    y.append(row.NER)
    x.append(row.word)
v_y.append(y)
v_x.append(x)
```

Make data list Test data

```
# Test
te_x, x = [], []
first = 0
for row in te_df.itertuples():
    if(row.id == '1' and first == 1):
        te_x.append(x)
        x = []
    first = 1
    x.append(row.word)
te_x.append(x)
## print(len(tr_x), len(tr_y))
## print(len(v_x), len(v_y))
```

Creating a dataset object for using dataloader for Train and Dev data

```
class BiLSTM_DataLoader(Dataset):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __len__(self):
        return len(self.x)

    def __getitem__(self, index):
        x_case = torch.tensor(self.x[index])
        y_case = torch.tensor(self.y[index])
        return x_case, y_case
```

Creating a dataset object for using dataloader for Test data

```
class BiLSTM_TestLoader(Dataset):
    def __init__(self, x):
        self.x = x

    def __len__(self):
        return len(self.x)
    def __getitem__(self, index):
        x_case = torch.tensor(self.x[index])
        return x_case
```

Creating a collator object for Train and Dev data

```
class Collator(object):
    def __init__(self, vocab, label):
        self.params = vocab
        self.label = label

    def __call__(self, batch):
        (xx, yy) = zip(*batch)
        len_x, len_y = [len(x) for x in xx], [len(y) for y in yy]
        max_len_batch = max([len(s) for s in xx])
        x_batch = self.params[pad_token]*np.ones((len(xx), max_len_batch))
        y_batch = -1*np.zeros((len(xx), max_len_batch))
        for j in range(len(xx)):
            len_curr = len(xx[j])
            x_batch[j][:len_curr] = xx[j]
            y_batch[j][:len_curr] = yy[j]
        x_batch, y_batch = torch.LongTensor(x_batch), torch.LongTensor(y_batch)
        x_batch, y_batch = Variable(x_batch), Variable(y_batch)
        return x_batch, y_batch, len_x, len_y
```

Creating a collator object for Test data

```
class TestCollator(object):
    def __init__(self, vocab, label):
        self.params = vocab
        self.label = label

    def __call__(self, batch):
        xx = batch
        len_x = [len(x) for x in xx]
        max_len_batch = max([len(s) for s in xx])
        x_batch = self.params[pad_token]*np.ones((len(xx), max_len_batch))
        for j in range(len(xx)):
            len_curr = len(xx[j])
            x_batch[j][:len_curr] = xx[j]
        x_batch = torch.LongTensor(x_batch)
        x_batch = Variable(x_batch)
        return x_batch, len_x
```

Creating a unique word dictionary

```
word_id, ind = {pad_token: 0, unk_token : 1}, 2
for data in [tr_x, v_x, te_x]:
    for sent in data:
        for word in sent:
            if word not in word_id:
                word_id[word] = ind
                ind += 1
```

Vectorizing train data

```
tr_x_vec, tmp_tr_x = [], []
for words in tr_x:
    for word in words:
        tmp_tr_x.append(word_id[word])
    tr_x_vec.append(tmp_tr_x)
    tmp_tr_x = []


te_x_vec, tmp_te_x = [], []
for words in te_x:
    for word in words:
        tmp_te_x.append(word_id[word])
    te_x_vec.append(tmp_te_x)
    tmp_te_x = []


v_x_vec, tmp_v_x = [], []
for words in v_x:
    for word in words:
        tmp_v_x.append(word_id[word])
    v_x_vec.append(tmp_v_x)
    tmp_v_x = []


tr_label = set()
for sentence in tr_y:
    for word in sentence:
        tr_label.add(word)

v_label=set()
for sentence in v_y:
    for word in sentence:
        v_label.add(word)


label = tr_label.union(v_label)
label_tuples, counter = [], 0
for tags in label:
    label_tuples.append((tags, counter))
    counter += 1
label_dict1 = dict(label_tuples)



tr_y_vec, v_y_vec = [], []
for tags in tr_y:
    tmp_tr_yy = []
    for label in tags:
        tmp_tr_yy.append(label_dict1[label])
    tr_y_vec.append(tmp_tr_yy)

for tags in v_y:
    tmp_v_yy = []
    for label in tags:
        tmp_v_yy.append(label_dict1[label])
    v_y_vec.append(tmp_v_yy)
```

Setting Class weights for individual NER tags

```
class_weights, class_wt = dict(), []
for k in label_dict1:
    class_weights[k] = 0
total_tags = 0
for d in [tr_y, v_y]:
    for ts in d:
        for t in ts:
            total_tags += 1
            class_weights[t] += 1
for ke in class_weights.keys():
    if class_weights[ke]:
        sol = round(math.log(0.35*total_tags / class_weights[ke]), 2)
        class_weights[ke] = sol if sol > 1.0 else 1.0
    else:
        class_weights[ke] = 1.0
    class_wt.append(class_weights[ke])
class_wt = torch.tensor(class_wt)
print(class_wt)
```

```
tensor([2.3600, 3.0000, 4.0900, 2.3000, 2.7300, 4.1500, 1.0000, 3.0200, 2.4600])
```

```
glove = pd.read_csv(glove_file_path, sep=" ",quoting=3, header=None, index_col=0)
glove_embed = {key: val.values for key, val in glove.T.items()}
glove_vector = np.array([glove_embed[key] for key in glove_embed])
glove_embed[pad_token] = np.zeros((100,), dtype="float64")
glove_embed[unk_token] = np.mean(glove_vector, axis=0, keepdims=True).reshape(100,)
```

```
emb_matrix1 = np.zeros((len(word_id), 100))
for w, idx in word_id.items():
    if w in glove_embed:
        emb_matrix1[idx] = glove_embed[w]
    else:
        if w.lower() in glove_embed:
            emb_matrix1[idx] = glove_embed[w.lower()] + 5e-3
        else:
            emb_matrix1[idx] = glove_embed[unk_token]
```

```
emb_matrix1
```

```
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.05209848, -0.09711399, -0.1380762 , ...,  0.12381646,
        -0.23434337, -0.00925499],
       [-0.32214   ,  0.087503  ,  1.2611    , ...,  0.30925   ,
         0.93884   ,  0.11394   ],
       ...,
       [ 0.061763  ,  0.27835   ,  0.71128   , ..., -0.16078   ,
        -0.84782   , -0.24545   ],
       [-0.12846   , -0.09701   , -0.19842   , ...,  0.087408  ,
        -1.2825    ,  0.28729   ],
       [ 0.05209848, -0.09711399, -0.1380762 , ...,  0.12381646,
        -0.23434337, -0.00925499]])
```

```
vocab_size = emb_matrix1.shape[0]
vector_size = emb_matrix1.shape[1]
print(vocab_size, vector_size)
```

```
30292 100
```

```
class BiLSTM(nn.Module):
    def __init__(self, size_of_vocab, embedding_dim,lstm_layers, hidden_dim, dropout_val, linear_dim,bidirectional, size_tag_dict):
        super(BiLSTM, self).__init__()
        self.embedding_dim = embedding_dim
        self.lstm_layers = lstm_layers
        self.hidden_dim = hidden_dim
        self.linear_dim = linear_dim
        self.tag_size = size_tag_dict
        self.num_directions = 2 if bidirectional else 1

        self.embedding = nn.Embedding(size_of_vocab, embedding_dim)
        self.embedding.weight.data.uniform_(-1,1)

        self.LSTM = nn.LSTM(embedding_dim,hidden_dim,num_layers=lstm_layers,batch_first=True,bidirectional=True)
        self.fc = nn.Linear(self.num_directions*hidden_dim,linear_dim)
        self.dropout = nn.Dropout(dropout_val)
        self.elu = nn.ELU(alpha=0.01)
        self.classifier = nn.Linear(linear_dim, self.tag_size)
    def init_hidden(self, size_of_batch):
        h = torch.zeros(self.lstm_layers * self.num_directions,size_of_batch, self.hidden_dim).to(device)
        c=  torch.zeros(self.lstm_layers * self.num_directions,size_of_batch, self.hidden_dim).to(device)
        return h, c

    def forward(self, se, se_len):
        size_of_batch = se.shape[0]
        h0, c0 = self.init_hidden(size_of_batch)
        embed = self.embedding(se).float()
        # This is needed for the embeddings, we have to reduce the train time too!
        packed_embedded = pack_padded_sequence(embed, se_len, batch_first=True, enforce_sorted=False)
        op, _ = self.LSTM(packed_embedded, (h0, c0))
        op_unpacked, _ = pad_packed_sequence(op, batch_first=True)
        dropout = self.dropout(op_unpacked)
        linx = self.fc(dropout)
        pred = self.elu(linx)
        pred = self.classifier(pred)
        return pred


class BiLSTM_glove(nn.Module):
    def __init__(self, size_of_vocab, embedding_dim,lstm_layers, hidden_dim, dropout_val, linear_dim,bidirectional, size_tag_dict, emb_matrix):
        super(BiLSTM_glove, self).__init__()
        self.embedding_dim = embedding_dim
        self.lstm_layers = lstm_layers
        self.hidden_dim = hidden_dim
        self.linear_dim = linear_dim
        self.tag_size = size_tag_dict
        self.emb_matrix = emb_matrix
        self.num_directions = 2 if bidirectional else 1

        self.embedding = nn.Embedding(size_of_vocab, embedding_dim)
        ## Glove embedding
        self.embedding.weight = nn.Parameter(torch.tensor(emb_matrix))
        self.LSTM = nn.LSTM(embedding_dim,hidden_dim,num_layers=lstm_layers,batch_first=True,bidirectional=True)
        self.fc = nn.Linear(self.num_directions*hidden_dim,linear_dim)
        self.dropout = nn.Dropout(dropout_val)
        self.elu = nn.ELU(alpha=0.01)
        self.classifier = nn.Linear(linear_dim, self.tag_size)

    def init_hidden(self, size_of_batch):
        h = torch.zeros(self.lstm_layers * self.num_directions,size_of_batch, self.hidden_dim).to(device)
        c=  torch.zeros(self.lstm_layers * self.num_directions,size_of_batch, self.hidden_dim).to(device)
        return h, c

    def forward(self, se, se_len):
        size_of_batch = se.shape[0]
        h0, c0 = self.init_hidden(size_of_batch)
        embed = self.embedding(se).float()
        packed_embedded = pack_padded_sequence(embed, se_len, batch_first=True, enforce_sorted=False)
        op, _ = self.LSTM(packed_embedded, (h0, c0))
        op_unpacked, _ = pad_packed_sequence(op, batch_first=True)
        dropout = self.dropout(op_unpacked)
        linx = self.fc(dropout)
        pred = self.elu(linx)
        pred = self.classifier(pred)
        return pred


BiLSTM_norm_model = BiLSTM(size_of_vocab=len(word_id),
                            embedding_dim=100,
                            lstm_layers=1,
                            hidden_dim=256,
                            dropout_val=0.33,
                            linear_dim=128,
                            bidirectional=True,
                            size_tag_dict=len(label_dict1))
BiLSTM_norm_model.to(device)
print(BiLSTM_norm_model)

BiLSTM_norm_train = BiLSTM_DataLoader(tr_x_vec, tr_y_vec)
custom_norm_collator = Collator(word_id, label_dict1)
dataloader_norm = DataLoader(dataset=BiLSTM_norm_train,
                        batch_size=4,
                        drop_last=True,
                        collate_fn=custom_norm_collator)
criterion = nn.CrossEntropyLoss(weight=class_wt)
criterion = criterion.to(device)
criterion.requres_grad = True
learning_rate=0.1
m=0.9
optimizer = torch.optim.SGD(BiLSTM_norm_model.parameters(), lr=learning_rate, momentum=m)
epochs = 84
for i in range(1, epochs+1):
    train_loss = 0.0
    for ip, lbl, len_ip, len_label in dataloader_norm:
        optimizer.zero_grad()
        op1 = BiLSTM_norm_model(ip.to(device), len_ip)
        op1 = op1.view(-1, len(label_dict1))
        lbl = lbl.view(-1)
        loss = criterion(op1, lbl.to(device))
        loss.backward()
        optimizer.step()
        train_loss = train_loss + loss.item() * ip.size(1)
    train_loss = train_loss / len(dataloader_norm.dataset)
    print('Epoch: {} \tTrain Loss: {:.6f}'.format(i, train_loss))
    #torch.save(BiLSTM_norm_model.state_dict(),'BiLSTM_' + str(i) + '.pt')
    torch.save(BiLSTM_norm_model.state_dict(),'blstm1.pt')
```

```
Epoch: 27       Train Loss: 0.056377
Epoch: 28       Train Loss: 0.051975
Epoch: 29       Train Loss: 0.037529
Epoch: 30       Train Loss: 0.035584
Epoch: 31       Train Loss: 0.036328
Epoch: 32       Train Loss: 0.030499
Epoch: 33       Train Loss: 0.024480
Epoch: 34       Train Loss: 0.027182
Epoch: 35       Train Loss: 0.031755
Epoch: 36       Train Loss: 0.026984
Epoch: 37       Train Loss: 0.020519
Epoch: 38       Train Loss: 0.023633
```

```
     Epoch: 73       Train Loss: 0.017657
     Epoch: 74       Train Loss: 0.016504
     Epoch: 75       Train Loss: 0.011799
     Epoch: 76       Train Loss: 0.011553
     Epoch: 77       Train Loss: 0.012862
     Epoch: 78       Train Loss: 0.013006
     Epoch: 79       Train Loss: 0.013353
     Epoch: 80       Train Loss: 0.013769
     Epoch: 81       Train Loss: 0.012284
     Epoch: 82       Train Loss: 0.027456
     Epoch: 83       Train Loss: 0.020919
     Epoch: 84       Train Loss: 0.013321
```

```
BiLSTM_norm_model = BiLSTM(size_of_vocab=len(word_id),
                            embedding_dim=100,
                            linear_dim=128,
                            hidden_dim=256,
                            lstm_layers=1,
                            bidirectional = True,
                            dropout_val=0.33,
                            size_tag_dict=len(label_dict1))
#BiLSTM_norm_model.load_state_dict(torch.load("./BiLSTM_83.pt"))
BiLSTM_norm_model.load_state_dict(torch.load("./blstm1.pt"))
BiLSTM_norm_model.to(device)
```

```
     BiLSTM(
       (embedding): Embedding(30292, 100)
       (LSTM): LSTM(100, 256, batch_first=True, bidirectional=True)
       (fc): Linear(in_features=512, out_features=128, bias=True)
       (dropout): Dropout(p=0.33, inplace=False)
       (elu): ELU(alpha=0.01)
       (classifier): Linear(in_features=128, out_features=9, bias=True)
     )
```

```
BiLSTM_norm_dev = BiLSTM_DataLoader(v_x_vec, v_y_vec)
custom_norm_collator = Collator(word_id, label_dict1)
dataloader_norm_dev = DataLoader(dataset=BiLSTM_norm_dev,
                                batch_size=1,
                                shuffle=False,
                                drop_last=True,
                                collate_fn=custom_norm_collator)

rev_label = {v: k for k, v in label_dict1.items()}
rev_vocab = {v: k for k, v in word_id.items()}

file = open(dev1_out_file, 'w')
for dev, lbl, len_dev_data, len_label_data in dataloader_norm_dev:
    pred = BiLSTM_norm_model(dev.to(device), len_dev_data)
    pred = pred.cpu()
    pred = pred.detach().numpy()
    lbl = lbl.detach().numpy()
    dev = dev.detach().numpy()
    pred = np.argmax(pred, axis=2)
    pred = pred.reshape((len(label), -1))
    for i in range(len(dev)):
        for j in range(len(dev[i])):
            if dev[i][j] != 0:
                word = rev_vocab[dev[i][j]]
                gold = rev_label[lbl[i][j]]
                op = rev_label[pred[i][j]]
                file.write(" ".join([str(j+1), word, gold, op]))
                file.write("\n")
        file.write("\n")
file.close()
```

```
!perl conll03eval.txt < dev1.out
```

```
processed 51578 tokens with 5942 phrases; found: 5741 phrases; correct: 4417.
accuracy:  95.19%; precision:  76.94%; recall:  74.34%; FB1:  75.61
              LOC: precision:  87.70%; recall:  81.55%; FB1:  84.51  1708
             MISC: precision:  77.52%; recall:  75.92%; FB1:  76.71  903
              ORG: precision:  68.70%; recall:  70.40%; FB1:  69.54  1374
              PER: precision:  72.61%; recall:  69.22%; FB1:  70.87  1756
```

```
#print(word_id)
```

```
#print(rev_vocab)
```

```
rev_label = {v: k for k, v in label_dict1.items()}
rev_vocab = {v: k for k, v in word_id.items()}
BiLSTM_norm_test = BiLSTM_TestLoader(te_x_vec)
custom_test_norm_collator = TestCollator(word_id, label_dict1)
dataloader_test = DataLoader(dataset=BiLSTM_norm_test,
                            batch_size=1,
                            shuffle=False,
                            drop_last=True,
                            collate_fn=custom_test_norm_collator)
file = open(test1_out_file, 'w')
for test_data, test_data_len in dataloader_test:
    pred = BiLSTM_norm_model(test_data.to(device), test_data_len)
    pred = pred.cpu()
    pred = pred.detach().numpy()
    test_data = test_data.detach().numpy()
    pred = np.argmax(pred, axis=2)
    pred = pred.reshape((len(test_data), -1))
    for i in range(len(test_data)):
        for j in range(len(test_data[i])):
            if test_data[i][j] != 0:
                word = rev_vocab[test_data[i][j]]
                op = rev_label[pred[i][j]]
                file.write(" ".join([str(j+1), word, op]))
                file.write("\n")
        file.write("\n")
file.close()



BiLSTM_glove_model = BiLSTM_glove(size_of_vocab=len(word_id),
                            embedding_dim=100,
                            lstm_layers=1,
                            hidden_dim=256,
                            dropout_val=0.33,
                            linear_dim=128,
                            bidirectional=True,
                            size_tag_dict=len(label_dict1),
                            emb_matrix=emb_matrix1)
BiLSTM_glove_model.to(device)
print(BiLSTM_glove_model)

BiLSTM_glove_train = BiLSTM_DataLoader(tr_x_vec, tr_y_vec)
custom_glove_collator = Collator(word_id, label_dict1)
dataloader_glove = DataLoader(dataset=BiLSTM_glove_train,
                        batch_size=8,
                        drop_last=True,
                        collate_fn=custom_glove_collator)
criterion = nn.CrossEntropyLoss(weight=class_wt)
criterion = criterion.to(device)
criterion.requres_grad = True
learning_rate=0.1
m=0.9
optimizer = torch.optim.SGD(BiLSTM_glove_model.parameters(), lr=learning_rate, momentum=m)
scheduler = StepLR(optimizer, step_size=15, gamma=0.9)
epochs = 50
for i in range(1, epochs+1):
    train_loss = 0.0
    for ip, lbl, len_ip, len_label in dataloader_glove:
        optimizer.zero_grad()
        op1 = BiLSTM_glove_model(ip.to(device), len_ip)
        op1 = op1.view(-1, len(label_dict1))
        lbl = lbl.view(-1)
        loss = criterion(op1, lbl.to(device))
        loss.backward()
        optimizer.step()
        train_loss = train_loss + loss.item() * ip.size(1)
    train_loss = train_loss / len(dataloader_glove.dataset)
    print('Epoch: {} \tTrain Loss: {:.6f}'.format(i, train_loss))
    #torch.save(BiLSTM_glove_model.state_dict(),'BiLSTM_glove_' + str(i) + '.pt')
    torch.save(BiLSTM_glove_model.state_dict(),'blstm2.pt')
```

```
BiLSTM_glove(
  (embedding): Embedding(30292, 100)
  (LSTM): LSTM(100, 256, batch_first=True, bidirectional=True)
  (fc): Linear(in_features=512, out_features=128, bias=True)
  (dropout): Dropout(p=0.33, inplace=False)
  (elu): ELU(alpha=0.01)
  (classifier): Linear(in_features=128, out_features=9, bias=True)
)
```

```
BiLSTM_glove_model = BiLSTM_glove(size_of_vocab=len(word_id),
                            embedding_dim=100,
                            linear_dim=128,
                            hidden_dim=256,
                            lstm_layers=1,
                            bidirectional = True,
                            dropout_val=0.33,
                            size_tag_dict=len(label_dict1),
                            emb_matrix=emb_matrix1)
BiLSTM_glove_model.load_state_dict(torch.load("./blstm2.pt"))
BiLSTM_glove_model.to(device)


BiLSTM_glove_dev = BiLSTM_DataLoader(v_x_vec, v_y_vec)
custom_glove_collator = Collator(word_id, label_dict1)
dataloader_glove_dev = DataLoader(dataset=BiLSTM_glove_dev,
                            batch_size=8,
                            shuffle=False,
                            drop_last=True,
                            collate_fn=custom_glove_collator)
print(label_dict1)
rev_label = {v: k for k, v in label_dict1.items()}
rev_vocab = {v: k for k, v in word_id.items()}

file = open(dev2_out_file, 'w')
for dev_data, label, dev_data_len, label_data_len in dataloader_glove_dev:
    pred = BiLSTM_glove_model(dev_data.to(device), dev_data_len)
    pred = pred.cpu()
    pred = pred.detach().numpy()
    label = label.detach().numpy()
    dev_data = dev_data.detach().numpy()
    pred = np.argmax(pred, axis=2)
    pred = pred.reshape((len(label), -1))
    for i in range(len(dev_data)):
        for j in range(len(dev_data[i])):
            if dev_data[i][j] != 0:
                word = rev_vocab[dev_data[i][j]]
                gold = rev_label[label[i][j]]
                op = rev_label[pred[i][j]]
                file.write(" ".join([str(j + 1), word, gold, op]))
                file.write("\n")
        file.write("\n")
file.close()
```

```
!perl conll03eval.txt < dev2.out
```

```
BiLSTM_test = BiLSTM_TestLoader(te_x_vec)
custom_test_collator = TestCollator(word_id, label_dict1)
dataloader_test = DataLoader(dataset=BiLSTM_test,
                             batch_size=1,
                             shuffle=False,
                             drop_last=True,
                             collate_fn=custom_test_collator)
rev_label = {v: k for k, v in label_dict1.items()}
rev_vocab = {v: k for k, v in word_id.items()}
file = open(test2_out_file, 'w')
for test_data, test_data_len in dataloader_test:
    pred = BiLSTM_glove_model(test_data.to(device), test_data_len)
    pred = pred.cpu()
    pred = pred.detach().numpy()
    test_data = test_data.detach().numpy()
    pred = np.argmax(pred, axis=2)
    pred = pred.reshape((len(test_data), -1))
    for i in range(len(test_data)):
        for j in range(len(test_data[i])):
            if test_data[i][j] != 0:
                word = rev_vocab[test_data[i][j]]
                op = rev_label[pred[i][j]]
                file.write(" ".join([str(j + 1), word, op]))
                file.write("\n")
        file.write("\n")
file.close()
```

● ✕