Name Entity Recognition

Mar 24th 2023

William Lu

```
import numpy as np
import torch
from torch.utils.data import DataLoader, Dataset
import torchvision.transforms as transforms
import torch.nn as nn
import torchvision
from torch.utils.data.sampler import SubsetRandomSampler
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader, Dataset
from torch import optim
import pickle
import pandas as pd
print("succesfully imported")
import string
```

```
succesfully imported
```

```
# reference: https://yoseflaw.medium.com/step-by-step-ner-model-for-bahasa-indonesia-with-pytorch-and-torchtext-6f94fca08406
# https://www.kaggle.com/code/ziliwang/baseline-pytorch-bilstm/input?select=train.csv
```

Data Processing

```
data = pd.read_csv('train',on_bad_lines='skip',sep=' ',header= None)
data.iloc[74] #  a bad word; need to separate it
```

```
0                                                       1
1     O\n2 We O\n3 do O\n4 n't O\n5 support O\n6 an...
2                                                       O
Name: 74, dtype: object
```

```
data
```

```
        0           1       2
0       1          EU   B-ORG
1       2     rejects       O
2       3      German  B-MISC
3       4        call       O
4       5          to       O
...    ..         ...     ...
124087  1     Swansea   B-ORG
124088  2           1       O
124089  3     Lincoln   B-ORG
124090  4           2       O
124091  1  -DOCSTART-       O
```

```
[124092 rows x 3 columns]
```

```
# change training and deving data to list of list of list
def to_sentence(path):
    df = list()
    with open(path, 'r') as f:
        for line in f.readlines():
            if len(line) > 2: # some line have corrupted content, for
instance, line 74. So we need to clean it this way.
                idx, word, NER = line.strip().split(" ")
                df.append([idx, word, NER])

    df = pd.DataFrame(df, columns=['idx', 'word', 'NER'])
    df = df.dropna()
    X_train, y_train = [],[]
    sent_X, sent_y = [],[]
    temp = 1
    for x in df.itertuples():
        if(x.idx == '1' and temp == 0):
            X_train.append(sent_X)
            y_train.append(sent_y)
            sent_X = []
            sent_y = []
        temp = 0
        sent_X.append(x.word)
        sent_y.append(x.NER)

    X_train.append(sent_X)
    y_train.append(sent_y)

    return X_train, y_train

X_train, y_train = to_sentence('train')
X_dev, y_dev = to_sentence('dev')

def to_sentence_test(path):
    df = []
    with open(path, 'r') as f:
        for x in f.readlines():
            if len(x) > 1: # some line have corrupted content, for
instance, line 74. So we need to clean it this way.
                idx, word= x.strip().split(" ")
                df.append([idx, word])

    df = pd.DataFrame(df, columns=['idx', 'word'])
    df = df.dropna()
    X_test=[]
    sent_X= []
    temp = 1
```

```
    for x in df.itertuples():
        if(x.idx == '1' and temp == 0):
            X_test.append(sent_X)
            sent_X = []
        temp = 0
        sent_X.append(x.word)


    X_test.append(sent_X)
    return X_test

X_test = to_sentence_test('test')
```

Make sentence to numbers by creating macthing dictionaries

```
vocab_dict= dict()
def create_dictionary(data1,data2,data3,vocabulary):
    data = [data1,data2,data3]
    idx = 2
    vocab_dict["<pad>"]=0
    vocab_dict["<unk>"]=1

    for i in data:
        for j in i:
            for k in j:
                if k not in vocab_dict:
                    vocab_dict[k]= idx
                    idx+=1
                else:
                    continue
    return vocab_dict

def create_dictionary2(data,vocabulary):
    idx = 2
    vocab_dict["<pad>"]=0
    vocab_dict["<unk>"]=1

    for i in data:
            for k in i:
                if k not in vocab_dict:
                    vocab_dict[k]= idx
                    idx+=1
                else:
                    continue
    return vocab_dict

vocab_dict2 = dict()
vocab_dict2 = create_dictionary2(X_train,vocab_dict2)
#len(vocab_dict2)

vocab_dict = create_dictionary(X_train,X_dev,X_test,vocab_dict)
# vocab_dict
```

```
def transform_to_num_data(data,dictionary):
    integer_list = []
    for sub in data:
        integer_sub = []
        for word in sub:
            integer_sub.append(dictionary[word])
        integer_list.append(integer_sub)
    return integer_list

X_train_num = transform_to_num_data(X_train,vocab_dict)
X_dev_num = transform_to_num_data(X_dev,vocab_dict)
X_test_num = transform_to_num_data(X_test,vocab_dict)

# only need to pass one set of data as NER dict should be short and
the same
def NER_dict(data):
    idx = 0
    ner_dict = dict()
    ner = list(data["NER"])
    for i in ner:
        if i not in ner_dict:
            ner_dict[i]=idx
            idx+=1
        else:
            continue
    return ner_dict

# get df data
df = list()
with open('train', 'r') as f:
    for line in f.readlines():
        if len(line) > 2: # some line have corrupted content, for
instance, line 74. So we need to clean it this way.
            idx, word, NER = line.strip().split(" ")
            df.append([idx, word, NER])
df = pd.DataFrame(df, columns=['idx', 'word', 'NER'])
df = df.dropna()

ner_dict = NER_dict(df)
#ner_dict

#y_train[0]

y_train_num = transform_to_num_data(y_train,ner_dict)
y_dev_num = transform_to_num_data(y_dev,ner_dict)
```

Bi-directional LSTM

Model Building

```
class BiLSTM(nn.Module):
    def __init__(self, input_dim, embedding_dim, output_dim,
hidden_dim, lstm_layers, bidirectional, dropout,tag_size):
```

```
        super().__init__()
        self.embedding_dim = embedding_dim
        self.tag_size = tag_size
        self.lstm_layer = lstm_layers
        # embedding
        self.embedding = nn.Embedding(num_embeddings=input_dim,
embedding_dim=embedding_dim,padding_idx=0)
        # Bi-LSTM
        self.blstm = nn.LSTM(
            input_size=embedding_dim,
            hidden_size=hidden_dim,
            num_layers=lstm_layers,
            bidirectional=True,
            batch_first =True
        )
        #Linear
        self.fc = nn.Linear(hidden_dim *2 , output_dim) # bidirectional
lstm
        self.dropout = nn.Dropout(dropout)
        # ELU
        self.elu = nn.ELU()
        # classifier
        self.classifier = nn.Linear(output_dim,tag_size)  # times 2
for bidirectional

    def forward(self,text):
        embedding_out = self.dropout(self.embedding(text))
        lstm_out, (hidden,cell) = self.blstm(embedding_out)
        lstm_out = self.dropout(lstm_out)
        lstm_out = self.elu(self.fc(lstm_out))
        pred = self.classifier(lstm_out)
        return pred

    # count the number of parameters
    def count_parameters(self):
        return sum(x.numel() for x in self.parameters() if
x.requires_grad)


embedding_dimension = 100
num_lstm_layer = 1
hidden_dimension = 256
dropout = 0.33
output_dimension = 128

bilstm = BiLSTM(
    input_dim = len(vocab_dict),#input dimension
    embedding_dim = embedding_dimension, #embedding dimension
    output_dim = output_dimension, # output_dimension
    hidden_dim = hidden_dimension, #hidden dimension
```

```
    lstm_layers = num_lstm_layer,#lstm_layers
    bidirectional= True,#bidirectional
    dropout = dropout,#dropout
    tag_size = len(ner_dict)#tag_size
)
# input_dim, embedding_dim, hidden_dim, output_dim, lstm_layers,
bidirectional, dropout,tag_size
number_pf_parameters = bilstm.count_parameters()
print("The number of trainable parameters is: ",number_pf_parameters)
print(bilstm)
```

```
The number of trainable parameters is:  3829209
BiLSTM(
  (embedding): Embedding(30292, 100, padding_idx=0)
  (blstm): 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=1.0)
  (classifier): Linear(in_features=128, out_features=9, bias=True)
)
```

Convert training, dev, and test to loader mode.

```
# pad the texts so that they have the same length
def padding(text,length,num):
    padded_x = []
    for row in text:
        if len(row) > length:
            padded_x.append(row[:length])
        else:
            padded_row = row + [num]*(length-len(row))
            padded_x.append(padded_row)

    return padded_x

# Make a dataset and dataloader
tempX = padding(X_train_num, 120,0)
tempy = padding(y_train_num, 120,-1)

X_train_tensor = torch.LongTensor(tempX)
y_train_tensor = torch.LongTensor(tempy)

train_tensor = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_tensor, batch_size=10, shuffle=False)
```

Training

```
def cal_accuracy(pred, y, ner_pad, words, pred_table):
    counter = correct = 0
    max_pred = pred.argmax(dim=1, keepdim=True)
    temp_tuple = zip(max_pred, y, words)
    for p, r, w in temp_tuple:
```

```
            if r.item() == ner_pad:
                continue
            pred_table.append((w.item(), p.item(), r.item()))
            if r.item() == p.item():
                correct += 1
            counter += 1
    return counter, correct, pred_table

def train(model, iterator, pred_table,optimizer):
    epoch_loss = 0
    epoch_acc = 0
    counter_total = 0
    model.train()
    for word, ner in iterator:
        optimizer.zero_grad()
        preds = model(word)
        preds = preds.view(-1, preds.shape[-1])
        ner = ner.view(-1)
        loss = criterion(preds, ner)
        counter, correct, pred_table = cal_accuracy(preds, ner,
ner_pad, word.view(-1), pred_table)

        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += correct
        counter_total += counter

    avg_los = epoch_loss / len(iterator)
    avg_accuracy = epoch_acc / counter_total
    return avg_los, avg_accuracy, pred_table

def evaluate(model, iterator, pred_table,criterion):

    epoch_loss = 0
    epoch_acc = 0
    counter_total = 0
    model.eval()

    with torch.no_grad():

        for word, ner in iterator:
            preds = model(word)
            # need reshape
            preds = preds.view(-1, preds.shape[-1])
            ner = ner.view(-1)

            loss = criterion(preds, ner)
```

```
            counter, correct, pred_table = cal_accuracy(preds, ner,
ner_pad, word.view(-1), pred_table)

            epoch_loss += loss.item()
            epoch_acc += correct
            counter_total += counter

    avg_los = epoch_loss / len(iterator)
    avg_accuracy = epoch_acc / counter_total

    return avg_los, avg_accuracy, pred_table

dev
tempdevX = padding(X_dev_num, 120,0)
tempdevy = padding(y_dev_num, 120,-1)
X_dev_tensor = torch.LongTensor(tempdevX)
y_dev_tensor = torch.LongTensor(tempdevy)

dev_tensor = TensorDataset(X_dev_tensor, y_dev_tensor)
dev_loader = DataLoader(dev_tensor, batch_size=10, shuffle=False)

# create a index to Ner tag dictionary.
idx_ner = dict()
for k, v in ner_dict.items():
    idx_ner[v]=k

epoch_num = 20
ner_pad= -1
optimizer = optim.SGD(bilstm.parameters(), lr=0.08,
momentum=0.9,dampening=0.1) # SGD is the Optimizer
criterion = nn.CrossEntropyLoss(ignore_index= -1)
temp_loss = 0

#predict_result_dev =
run_training(epoch_num,bilstm,train_loader,dev_loader)

for epoch in range(epoch_num):
    temp_train = list()
    temp_test = list()

    train_loss, train_acc, train_pred_table = train(bilstm,
train_loader, temp_train,optimizer)
    val_loss, val_acc, val_pred_table = evaluate(bilstm, dev_loader,
temp_test,criterion)

    if val_loss <= float('inf'):
        temp_loss = val_loss
        predict_result = val_pred_table
        torch.save(bilstm.state_dict(), 'blstm1.pt')
```

```
    print(f'Epoch: {epoch+1:02}')
    print(f'\t Trn Loss: {train_loss:.3f} |  Trn Acc:
{train_acc*100:.2f}%')
    print(f'\t Val Loss: {val_loss:.3f} |    Val Acc:
{val_acc*100:.2f}%')
```

```
Epoch: 01
	Trn Loss: 0.721 |  Trn Acc: 84.17%
	Val Loss: 0.525 |    Val Acc: 86.51%
Epoch: 02
	Trn Loss: 0.516 |  Trn Acc: 86.78%
	Val Loss: 0.386 |    Val Acc: 89.38%
Epoch: 03
	Trn Loss: 0.413 |  Trn Acc: 88.58%
	Val Loss: 0.315 |    Val Acc: 91.06%
Epoch: 04
	Trn Loss: 0.348 |  Trn Acc: 89.87%
	Val Loss: 0.278 |    Val Acc: 91.91%
Epoch: 05
	Trn Loss: 0.308 |  Trn Acc: 90.69%
	Val Loss: 0.261 |    Val Acc: 92.41%
Epoch: 06
	Trn Loss: 0.277 |  Trn Acc: 91.46%
	Val Loss: 0.250 |    Val Acc: 92.78%
Epoch: 07
	Trn Loss: 0.256 |  Trn Acc: 91.91%
	Val Loss: 0.238 |    Val Acc: 93.09%
Epoch: 08
	Trn Loss: 0.240 |  Trn Acc: 92.32%
	Val Loss: 0.229 |    Val Acc: 93.41%
Epoch: 09
	Trn Loss: 0.223 |  Trn Acc: 92.72%
	Val Loss: 0.225 |    Val Acc: 93.52%
Epoch: 10
	Trn Loss: 0.211 |  Trn Acc: 93.00%
	Val Loss: 0.224 |    Val Acc: 93.61%
Epoch: 11
	Trn Loss: 0.200 |  Trn Acc: 93.31%
	Val Loss: 0.222 |    Val Acc: 93.78%
Epoch: 12
	Trn Loss: 0.192 |  Trn Acc: 93.57%
	Val Loss: 0.227 |    Val Acc: 93.67%
Epoch: 13
	Trn Loss: 0.182 |  Trn Acc: 93.79%
	Val Loss: 0.220 |    Val Acc: 93.87%
Epoch: 14
	Trn Loss: 0.178 |  Trn Acc: 93.88%
	Val Loss: 0.213 |    Val Acc: 94.01%
Epoch: 15
	Trn Loss: 0.172 |  Trn Acc: 94.14%
```

```
      Val Loss: 0.217 |    Val Acc: 94.09%
Epoch: 16
      Trn Loss: 0.167 |  Trn Acc: 94.24%
      Val Loss: 0.218 |    Val Acc: 94.18%
Epoch: 17
      Trn Loss: 0.159 |  Trn Acc: 94.45%
      Val Loss: 0.220 |    Val Acc: 94.13%
Epoch: 18
      Trn Loss: 0.153 |  Trn Acc: 94.61%
      Val Loss: 0.218 |    Val Acc: 94.16%
Epoch: 19
      Trn Loss: 0.150 |  Trn Acc: 94.69%
      Val Loss: 0.215 |    Val Acc: 94.20%
Epoch: 20
      Trn Loss: 0.145 |  Trn Acc: 94.84%
      Val Loss: 0.217 |    Val Acc: 94.26%
```

```
try:
    with open("dev","r") as dev, open("dev1.out","w") as dev1_out:
        y_dev_pred = []
        for i in predict_result:
            y_dev_pred.append(int(i[1]))
        temp =0
        for x in dev:
            x = x.strip()
            if x:
                idx,ner = x.split(" ")[:2]
                pred_ner = idx_ner[y_dev_pred[temp]]
                temp+=1
                dev1_out.write(f"{idx} {ner} {pred_ner}\n")
            else:
                dev1_out.write("\n")
except IOError as error:
    print("There's an error opening the file. Please correct the path.
Thanks.")

## for perl testing:
try:
    with open("dev","r") as dev, open("dev1_perl.out","w") as
dev1_out:
        y_dev_pred = []
        for i in predict_result:
            y_dev_pred.append(int(i[1]))
        temp2 =0
        for x in dev:
            x = x.strip()
            if x:
                item = x.split(" ")
                idx,word,ner = item[0],item[1],item[2]
                pred_ner = idx_ner[y_dev_pred[temp2]]
                temp2+=1
```

```
                dev1_out.write(f"{idx} {word} {ner} {pred_ner}\n")
            else:
                dev1_out.write("\n")
except IOError as error:
    print("There's an error opening the file. Please correct the path.
Thanks.")
!perl conll03eval.txt < dev1_perl.out
```

```
processed 51578 tokens with 5942 phrases; found: 5316 phrases;
correct: 4053.
accuracy:  94.26%; precision:  76.24%; recall:  68.21%; FB1:  72.00
              LOC: precision:  82.92%; recall:  77.68%; FB1:  80.21
1721
             MISC: precision:  76.83%; recall:  71.91%; FB1:  74.29
863
              ORG: precision:  64.32%; recall:  60.63%; FB1:  62.42
1264
              PER: precision:  78.34%; recall:  62.43%; FB1:  69.49
1468
```

test data

```
temptestX = padding(X_test_num, 120,0)
X_test_tensor = torch.LongTensor(temptestX)
test_loader = DataLoader(X_test_tensor, batch_size=10, shuffle=False)
```

```
def cal_evaluate2(preds, words, pred_result):
    max_preds = preds.argmax(dim = 1, keepdim = True) # get the index
of the max probability
    temp_tuple = zip(max_preds, words)
    for p, w in temp_tuple:
        if w == 0:
            continue
        else:
            pred_result.append((w, p[0]))

    return pred_result

def evaluate2(model, iterator, pred_table):

    epoch_loss = 0
    epoch_acc = 0
    model.eval()

    with torch.no_grad():

        for word in iterator:

            pred = model(word)
            pred = pred.view(-1, pred.shape[-1])
```

```
            pred_table = cal_evaluate2(pred, word.view(-1),
pred_table)

    return pred_table

pred_result2 = []
pred_result2 = evaluate2(bilstm, test_loader, pred_result2)

try:
    with open("test","r") as test, open("test1.out","w") as test1_out:
        y_test_pred = []
        temp4 =0
        for i in pred_result2:
            y_test_pred.append(int(i[1]))
        for x in test:
            x = x.strip()
            if x and temp4<len(y_test_pred):
                idx, word = x.split()[:2]
                #idx, word = x[0],x[1]
                pred_ner = idx_ner[y_test_pred[temp4]]
                temp4+=1
                test1_out.write(f"{idx} {word} {pred_ner}\n")
            else:
                test1_out.write("\n")
except IOError as error:
    print("There's an error opening the file. Please correct the path.
Thanks.")
```

BiLSTM with GloVe word embeddings

```
glove = pd.read_csv('glove.6B.100d', sep=" ", quoting=3, header=None,
index_col=0)
#glove

# make the glove dataframe to be like a dictionary where each word is
the key.
glove2 =glove.T
glove_dict = dict()
for k,v in glove2.items():
    glove_dict[k] = v.values
# glove_dict

# embedding matrix should be like (length of vocab dict, embedding
dimension)
def embedding_matrix(embedding_size,vocab_dict,glove_vec):
    width  = int(len(vocab_dict))
    embedding_matrix = np.zeros((width,embedding_size))
    for w, j in vocab_dict.items():
        embedding_vec = glove_vec.get(w.lower())
        if embedding_vec is not None:
            embedding_matrix[j] = embedding_vec
```

```
    embedding_matrix = torch.LongTensor(embedding_matrix)
    return embedding_matrix

embedding_matrix = embedding_matrix(100,vocab_dict,glove_dict)
#embedding_matrix

#embedding_matrix.shape

class BiLSTM_glove(nn.Module):
    def __init__(self, input_dim, embedding_dim, output_dim,
hidden_dim, lstm_layers, bidirectional, dropout,tag_size):
        super().__init__()
        self.embedding_dim = embedding_dim
        self.tag_size = tag_size
        self.lstm_layer = lstm_layers
        # embedding
        self.embedding = nn.Embedding(num_embeddings=input_dim,
embedding_dim=embedding_dim,padding_idx=0)
        # Bi-LSTM
        self.blstm = nn.LSTM(
            input_size=embedding_dim,
            hidden_size=hidden_dim,
            num_layers=lstm_layers,
            bidirectional=True,
            batch_first =True
        )
        #Linear
        self.fc = nn.Linear(hidden_dim *2 , output_dim) # bidrectional
lstm
        self.dropout = nn.Dropout(dropout)
        # ELU
        self.elu = nn.ELU()
        # classifier
        self.classifier = nn.Linear(output_dim,tag_size)  # times 2
for bidirectional

    def forward(self,text):
        embedding_out = self.dropout(self.embedding(text))
        lstm_out, (hidden,cell) = self.blstm(embedding_out)
        lstm_out = self.dropout(lstm_out)
        lstm_out = self.elu(self.fc(lstm_out))
        pred = self.classifier(lstm_out)
        return pred

     # initialize all parameters from normal distribution for better
converging during training

    # count the number of parameters
    def count_parameters(self):
        return sum(x.numel() for x in self.parameters() if
```

```
x.requires_grad)


bilstm_glove = BiLSTM_glove(
    input_dim = len(vocab_dict),#input dimension
    embedding_dim = embedding_dimension, #embedding dimension
    output_dim = output_dimension, # output_dimension
    hidden_dim = hidden_dimension, #hidden dimension
    lstm_layers = num_lstm_layer,#lstm_layers
    bidirectional= True,#bidirectional
    dropout = dropout,#dropout
    tag_size = len(ner_dict)#tag_size
)
bilstm_glove.embedding.weight.data.copy_(embedding_matrix) # add
embedding matrix
# input_dim, embedding_dim, hidden_dim, output_dim, lstm_layers,
bidirectional, dropout,tag_size):
number_pf_parameters2 = bilstm_glove.count_parameters()
#bilstm.to(device)
print("The number of trainable parameters is: ",number_pf_parameters2)
print(bilstm_glove)

The number of trainable parameters is:  3829209
BiLSTM_glove(
  (embedding): Embedding(30292, 100, padding_idx=0)
  (blstm): 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=1.0)
  (classifier): Linear(in_features=128, out_features=9, bias=True)
)

epoch_num = 20
ner_pad=-1
optimizer2 = optim.SGD(bilstm_glove.parameters(), lr=0.05,
momentum=0.9, nesterov=True)#weight_decay=0.3
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer2,
'min', patience=4)
criterion2 = nn.CrossEntropyLoss(ignore_index= -1)
temp_loss2 = float('inf')

def run_training(epoch_num,model,training,testing,optim,criter,name):
    for epoch in range(epoch_num):
        temp_train = list()
        temp_test = list()

        train_loss, train_acc, train_pred_result = train(model,
training, temp_train,optim)
        val_loss, val_acc, val_pred_result = evaluate(model, testing,
temp_test,criter)
```

```
        if val_loss <= float('inf'):
            temp_loss2 = val_loss
            predict_result = val_pred_result
            torch.save(bilstm.state_dict(), str(name))
        scheduler.step(val_loss)
        print(f'Epoch: {epoch+1:02}')
        print(f'\t Trn Loss: {train_loss:.3f} |  Trn Acc:
{train_acc*100:.2f}%')
        print(f'\t Val Loss: {val_loss:.3f} |  Val Acc:
{val_acc*100:.2f}%')
    return predict_result

result_golve =
run_training(20,bilstm_glove,train_loader,dev_loader,optimizer2,criter
ion2,'blstm2.pt')
```

```
Epoch: 01
      Trn Loss: 0.734 |  Trn Acc: 83.94%
      Val Loss: 0.548 |  Val Acc: 86.79%
Epoch: 02
      Trn Loss: 0.417 |  Trn Acc: 88.36%
      Val Loss: 0.322 |  Val Acc: 91.15%
Epoch: 03
      Trn Loss: 0.284 |  Trn Acc: 91.44%
      Val Loss: 0.257 |  Val Acc: 92.91%
Epoch: 04
      Trn Loss: 0.217 |  Trn Acc: 93.22%
      Val Loss: 0.229 |  Val Acc: 93.69%
Epoch: 05
      Trn Loss: 0.168 |  Trn Acc: 94.63%
      Val Loss: 0.214 |  Val Acc: 94.22%
Epoch: 06
      Trn Loss: 0.137 |  Trn Acc: 95.49%
      Val Loss: 0.211 |  Val Acc: 94.52%
Epoch: 07
      Trn Loss: 0.114 |  Trn Acc: 96.22%
      Val Loss: 0.211 |  Val Acc: 94.75%
Epoch: 08
      Trn Loss: 0.097 |  Trn Acc: 96.72%
      Val Loss: 0.221 |  Val Acc: 94.76%
Epoch: 09
      Trn Loss: 0.084 |  Trn Acc: 97.16%
      Val Loss: 0.222 |  Val Acc: 94.87%
Epoch: 10
      Trn Loss: 0.072 |  Trn Acc: 97.49%
      Val Loss: 0.235 |  Val Acc: 94.93%
Epoch: 11
      Trn Loss: 0.065 |  Trn Acc: 97.75%
      Val Loss: 0.232 |  Val Acc: 95.04%
Epoch: 12
      Trn Loss: 0.056 |  Trn Acc: 98.01%
```

```
      Val Loss: 0.242 |  Val Acc: 95.17%
Epoch: 13
      Trn Loss: 0.053 |  Trn Acc: 98.08%
      Val Loss: 0.246 |  Val Acc: 95.16%
Epoch: 14
      Trn Loss: 0.053 |  Trn Acc: 98.08%
      Val Loss: 0.247 |  Val Acc: 95.16%
Epoch: 15
      Trn Loss: 0.052 |  Trn Acc: 98.13%
      Val Loss: 0.247 |  Val Acc: 95.18%
Epoch: 16
      Trn Loss: 0.051 |  Trn Acc: 98.13%
      Val Loss: 0.248 |  Val Acc: 95.19%
Epoch: 17
      Trn Loss: 0.049 |  Trn Acc: 98.19%
      Val Loss: 0.244 |  Val Acc: 95.24%
Epoch: 18
      Trn Loss: 0.050 |  Trn Acc: 98.18%
      Val Loss: 0.242 |  Val Acc: 95.26%
Epoch: 19
      Trn Loss: 0.049 |  Trn Acc: 98.23%
      Val Loss: 0.242 |  Val Acc: 95.26%
Epoch: 20
      Trn Loss: 0.050 |  Trn Acc: 98.20%
      Val Loss: 0.242 |  Val Acc: 95.27%
```

```
try:
    with open("dev","r") as dev, open("dev2.out","w") as dev2_out:
        y_dev_pred_g = []
        for i in result_golve:
            y_dev_pred_g.append(int(i[1]))
        temp6 =0
        for x in dev:
            x = x.strip()
            if x:
                idx,ner = x.split(" ")[:2]
                pred_ner = idx_ner[y_dev_pred_g[temp6]]
                temp6+=1
                dev2_out.write(f"{idx} {ner} {pred_ner}\n")
            else:
                dev2_out.write("\n")

        print("success")
except IOError as error:
    print("There's an error opening the file. Please correct the path.
Thanks.")
```

```
success
```

```
## for perl testing:
try:
```

```
    with open("dev","r") as dev, open("dev2_perl.out","w") as
dev2_out:
        y_dev_pred_g2 = []
        for i in result_golve:
            y_dev_pred_g2.append(int(i[1]))
        temp5 =0
        for x in dev:
            x = x.strip()
            if x:
                item = x.split(" ")
                idx,word,ner = item[0],item[1],item[2]
                pred_ner = idx_ner[y_dev_pred_g2[temp5]]
                temp5+=1
                dev2_out.write(f"{idx} {word} {ner} {pred_ner}\n")
            else:
                dev2_out.write("\n")
        print("success")
except IOError as error:
    print("There's an error opening the file. Please correct the path.
Thanks.")
!perl conll03eval.txt < dev2_perl.out

success
processed 51578 tokens with 5942 phrases; found: 5860 phrases;
correct: 4510.
accuracy:  95.27%; precision:  76.96%; recall:  75.90%; FB1:  76.43
              LOC: precision:  82.93%; recall:  84.87%; FB1:  83.88
1880
             MISC: precision:  75.54%; recall:  75.38%; FB1:  75.46
920
              ORG: precision:  69.25%; recall:  70.02%; FB1:  69.63
1356
              PER: precision:  77.29%; recall:  71.50%; FB1:  74.28
1704

pred_result2_g = []
pred_result2_g = evaluate2(bilstm_glove, test_loader, pred_result2_g)

try:
    with open("test","r") as test, open("test2.out","w") as test2_out:
        y_test_pred_g = []
        temp8 =0
        for i in pred_result2_g:
            y_test_pred_g.append(int(i[1]))
        for x in test:
            x = x.strip()
            if x and temp8<len(y_test_pred_g):
                idx, word = x.split()[:2]
                pred_ner = idx_ner[y_test_pred_g[temp8]]
                temp8+=1
                test2_out.write(f"{idx} {word} {pred_ner}\n")
```

```
            else:
                test2_out.write("\n")
        print("success")
except IOError as error:
    print("There's an error opening the file. Please correct the path.
Thanks.")
```

```
success
```

```
'''
epoch_num = 20
ner_pad=-1
optimizer3 = optim.SGD(bilstm_glove.parameters(), lr=0.01,
momentum=0.9, nesterov=True,weight_decay=0.001)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer3,
'min', patience=4)
criterion2 = nn.CrossEntropyLoss(ignore_index= -1)
'''
```

```
"\nepoch_num = 20\nner_pad=-1\noptimizer3 =
optim.SGD(bilstm_glove.parameters(), lr=0.01, momentum=0.9,
nesterov=True)\nscheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer3, 'min',
patience=4)\ncriterion2 = nn.CrossEntropyLoss(ignore_index= -1)\n"
```

```
'''
a=
run_training(20,bilstm_glove,train_loader,dev_loader,optimizer3,criter
ion2,'blstm3.pt')
a
'''
```

```
"\na=
run_training(20,bilstm_glove,train_loader,dev_loader,optimizer3,criter
ion2,'blstm3.pt')\na\n"
```

```
'''
## for perl testing:
try:
    with open("dev","r") as dev, open("dev3_perl.out","w") as
dev3_out:
        y_dev_pred_g3 = []
        for i in a:
            y_dev_pred_g3.append(int(i[1]))
        temp10 =0
        for x in dev:
            x = x.strip()
            if x:
                item = x.split(" ")
                idx,word,ner = item[0],item[1],item[2]
                pred_ner = idx_ner[y_dev_pred_g3[temp10]]
```

```
                temp10+=1
                dev3_out.write(f"{idx} {word} {ner} {pred_ner}\n")
            else:
                dev3_out.write("\n")
except IOError as error:
    print("There's an error opening the file. Please correct the path.
Thanks.")
!perl conll03eval.txt < dev3_perl.out
'''

'\n## for perl testing:\ntry:\n    with open("dev","r") as dev,
open("dev3_perl.out","w") as dev3_out:\n        y_dev_pred_g3 = []\n
for i in a:\n            y_dev_pred_g3.append(int(i[1]))\n
temp10 =0\n        for x in dev:\n            x = x.strip()\n
if x:\n                item = x.split(" ")\n
idx,word,ner = item[0],item[1],item[2]\n                pred_ner =
idx_ner[y_dev_pred_g3[temp10]]\n                temp10+=1\n
dev3_out.write(f"{idx} {word} {ner} {pred_ner}\n")\n            else:\
n                dev3_out.write("\n")\nexcept IOError as error:\n
print("There\'s an error opening the file. Please correct the path.
Thanks.")\n!perl conll03eval.txt < dev3_perl.out\n'
```