I have created a BiLSTM class that represents our Bidirectional-Lstm model with a generic embedding layer and a BiLSTM_glove class that represents the same model, but with a Glove-based embedding. The CustomCollator function is utilized during training and validation to modify the input of each batch to the BiLSTM model, while the CustomTestCollator function is used during testing to ensure that all batch sentences are of equal length by padding the shorter ones. We use the BiLSTM_DataLoader class to provide the model with data from the training and validation datasets and the BiLSTM_TestLoader class for the testing dataset.

The function "create_emb_matrix" generates a matrix using the dictionary of the glove model, which is then used to input data into the Bilstm model. It is important to note that the glove model dictionary only contains lowercase words, and to address this issue, we adjust the embedding for titled words by adding a slight displacement value to each dimension of the lowercase counterpart.

Hyperparameters for Bidirectional-lstm are:

- Embedding dimension = 100
- Hidden dimension = 256
- Linear Output dimension = 128
- Bidirectional = True
- Dropout = 0.33
- Number of LSTM layers = 1
- Batch Size = 4
- Loss Function = Cross Entropy with class weights
- Optimizer = SGD with Learning Rate = 0.1 and Momentum = 0.9
- Epochs = 200

What are the precision, recall and F1 score on the dev data?

accuracy: 95.36%

precision: 78.60%

recall: 74.80%

FB1: 76.65

Hyperparameters for Birectional-lstm with glove-based embedding:

- Embedding dimension = 100
- Hidden dimension = 256
- Linear Output dimension = 128
- Bidirectional = True
- Dropout = 0.33
- Number of LSTM layers = 1
- Batch Size = 8
- Loss Function = Cross Entropy with class weights
- Optimizer = SGD with Learning Rate = 0.1 and Momentum = 0.9
- Epochs = 50

What are the precision, recall and F1 score on the dev data?

accuracy: 98.02%

precision: 89.23%

recall: 90.12%

FB1: 89.67

csci544-hw4

March 28, 2023

[1]:
```
import pandas as pd
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

1 Importing libraries

[2]:
```
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
import json
torch.manual_seed(0)
random.seed(0)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

2 Preparing Data

[3]:
```
df_train = list()
with open('/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/data/train',␣
 ↪'r') as f:
    for line in f.readlines():
        if len(line) > 1:
            id, word, ner= line.strip().split(" ")
```

```
            df_train.append([id, word, ner])

df_train = pd.DataFrame(df_train, columns=['id', 'word', 'NER'])
df_train = df_train.dropna()
```

[4]:
```
train_x, train_y = [], []
x, y = [], []
first=1

for row in df_train.itertuples():
    if(row.id == '1' and first == 0):
        train_x.append(x)
        train_y.append(y)
        x=[]
        y=[]
    first=0
    x.append(row.word)
    y.append(row.NER)
```

[5]:
```
df_dev = list()
with open('/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/data/dev', 'r') as f:
    for line in f.readlines():
        if len(line) > 1:
            id, word, ner = line.strip().split(" ")
            df_dev.append([id, word, ner])

df_dev = pd.DataFrame(df_dev, columns=['id', 'word', 'NER'])
df_dev = df_dev.dropna()
```

[6]:
```
dev_x, dev_y = [], []
x, y = [], []
first=1

for row in df_dev.itertuples():
    if(row.id == '1' and first == 0):
        dev_x.append(x)
        dev_y.append(y)
        x=[]
        y=[]
    first=0
    x.append(row.word)
    y.append(row.NER)
```

[7]:
```
df_test = list()
with open('/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/data/test', 'r') as f:
```

```
    for line in f.readlines():
        if len(line) > 1:
            id, word = line.strip().split(" ")
            df_test.append([id, word])

df_test = pd.DataFrame(df_test, columns=['id', 'word'])
df_test = df_test.dropna()
```

[8]:
```
test_x = []
x = []
first=1

for row in df_test.itertuples():
    if(row.id == '1' and first == 0):
        test_x.append(x)
        x=[]
    first=0
    x.append(row.word)
```

3 Creating vocabulary and labels

[9]:
```
index=1
word2idx={'<pad>': 0}
rev_vocab_dict = {0: '<pad>'}
for x in [train_x, dev_x, test_x]:
    for sentence in x:
        for word in sentence:
            if word not in word2idx:
                word2idx[word] = index
                rev_vocab_dict[index] = word
                index+=1
```

[10]:
```
len(word2idx)
```

[10]: 30291

[11]:
```
labels = set()
label_dict = {}
rev_label_dict = {}
index=0

for x in [train_y, dev_y]:
    for sentence in x:
        for label in sentence:
            labels.add(label)
            if label not in label_dict:
```

```
                label_dict[label] = index
                rev_label_dict[index] = label
                index+=1
```

[12]:
```
label_dict
```

[12]:
```
{'B-ORG': 0,
 'O': 1,
 'B-MISC': 2,
 'B-PER': 3,
 'I-PER': 4,
 'B-LOC': 5,
 'I-ORG': 6,
 'I-MISC': 7,
 'I-LOC': 8}
```

4 Vectorizing sentences and labels

[13]:
```
train_x_vec = []
x = []

for words in train_x:
    for word in words:
        x.append(word2idx[word])
    train_x_vec.append(x)
    x = []
```

[14]:
```
dev_x_vec = []
x = []

for words in dev_x:
    for word in words:
        x.append(word2idx[word])
    dev_x_vec.append(x)
    x = []
```

[15]:
```
test_x_vec = []
x = []

for words in test_x:
    for word in words:
        x.append(word2idx[word])
    test_x_vec.append(x)
    x = []
```

[16]:
```
train_y_vec = []

for tags in train_y:
    y = []
    for label in tags:
        y.append(label_dict[label])
    train_y_vec.append(y)
```

[17]:
```
dev_y_vec = []

for tags in dev_y:
    y = []
    for label in tags:
        y.append(label_dict[label])
    dev_y_vec.append(y)
```

5 Bidirectional LSTM

[18]:
```
class BiLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, linear_out_dim, hidden_dim,␣
↪lstm_layers, bidirectional, dropout_val, tag_size, glove_flag, emb_matrix):
        super(BiLSTM, self).__init__()
        """ Hyper Parameters """
        self.hidden_dim = hidden_dim  # hidden_dim = 256
        self.lstm_layers = lstm_layers  # LSTM Layers = 1
        # self.embedding_dim = embedding_dim  # Embedding Dimension = 100
        # self.linear_out_dim = linear_out_dim  # Linear Ouput Dimension = 128
        # self.tag_size = tag_size  # Tag Size = 9
        self.num_directions = 2 if bidirectional else 1

        """ Initializing Network """
        self.embedding = nn.Embedding(vocab_size, embedding_dim)  # Embedding␣
↪Layer

        if(glove_flag): self.embedding.weight = nn.Parameter(torch.
↪tensor(emb_matrix))
        else: 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(hidden_dim*self.num_directions, linear_out_dim)  #␣
↪2 for bidirection
        self.dropout = nn.Dropout(dropout_val)
```

```
        self.elu = nn.ELU(alpha=0.01)
        self.classifier = nn.Linear(linear_out_dim, tag_size)


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

    def forward(self, sen, sen_len):  # sen_len
        # Set initial states
        batch_size = sen.shape[0]
        h_0, c_0 = self.init_hidden(batch_size)

        # Forward propagate LSTM
        embedded = self.embedding(sen).float()
        packed_embedded = pack_padded_sequence(embedded, sen_len,␣
↪batch_first=True, enforce_sorted=False)
        output, _ = self.LSTM(packed_embedded, (h_0, c_0))
        output_unpacked, _ = pad_packed_sequence(output, batch_first=True)
        dropout = self.dropout(output_unpacked)
        lin = self.fc(dropout)
        pred = self.elu(lin)
        pred = self.classifier(pred)
        return pred
```

[19]:
```
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_instance = torch.tensor(self.x[index])  # , dtype=torch.long
        y_instance = torch.tensor(self.y[index])  # , dtype=torch.float
        return x_instance, y_instance
```

[20]:
```
class CustomCollator(object):

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

```
    def __call__(self, batch):
        (xx, yy) = zip(*batch)
        x_len = [len(x) for x in xx]
        y_len = [len(y) for y in yy]
        batch_max_len = max([len(s) for s in xx])
        batch_data = self.params['<pad>']*np.ones((len(xx), batch_max_len))
        batch_labels = -1*np.zeros((len(xx), batch_max_len))
        for j in range(len(xx)):
            cur_len = len(xx[j])
            batch_data[j][:cur_len] = xx[j]
            batch_labels[j][:cur_len] = yy[j]

        batch_data, batch_labels = torch.LongTensor(batch_data), torch.
↪LongTensor(batch_labels)
        batch_data, batch_labels = Variable(batch_data), Variable(batch_labels)

        return batch_data, batch_labels, x_len, y_len
```

[21]:
```
class_weights = dict()
for key in label_dict:
    class_weights[key] = 0
total_nm_tags = 0
for data in [train_y, dev_y]:
    for tags in data:
        for tag in tags:
            total_nm_tags += 1
            class_weights[tag] += 1

class_wt = list()
for key in class_weights.keys():
    if class_weights[key]:
        score = round(math.log(0.35*total_nm_tags / class_weights[key]), 2)
        class_weights[key] = score if score > 1.0 else 1.0
    else:
        class_weights[key] = 1.0
    class_wt.append(class_weights[key])
class_wt = torch.tensor(class_wt)
```

5.1 Hyperparameters:

5.1.1 Embedding dimension = 100

5.1.2 Hidden dimension = 256

5.1.3 Linear Output dimension = 128

5.1.4 Bidirectional = True

5.1.5 Dropout = 0.33

5.1.6 Number of LSTM layers = 1

5.1.7 Batch Size = 4

5.1.8 Loss Function = Cross Entropy with class weights

5.1.9 Optimizer = SGD with Learning Rate = 0.1 and Momentum = 0.9

5.1.10 Epochs = 200

[22]:
```
# BiLSTM_model = BiLSTM(vocab_size=len(word2idx),
#                       embedding_dim=100,
#                       linear_out_dim=128,
#                       hidden_dim=256,
#                       lstm_layers=1,
#                       bidirectional=True,
#                       dropout_val=0.33,
#                       tag_size=len(label_dict),
#                       glove_flag=False,
#                       emb_matrix=[])

# BiLSTM_model.to(device)
# print(BiLSTM_model)

# BiLSTM_train = BiLSTM_DataLoader(train_x_vec, train_y_vec)
# custom_collator = CustomCollator(word2idx, label_dict)
# dataloader = DataLoader(dataset=BiLSTM_train,
#                         batch_size=4,
#                         drop_last=True,
#                         collate_fn=custom_collator)

# criterion = nn.CrossEntropyLoss(weight=class_wt)

# criterion = criterion.to(device)
# criterion.requres_grad = True
# optimizer = torch.optim.SGD(BiLSTM_model.parameters(), lr=0.1, momentum=0.9)
# epochs = 200

# for i in range(1, epochs+1):
#     train_loss = 0.0
```

```
#     for input, label, input_len, label_len in dataloader:
#         optimizer.zero_grad()
#         output = BiLSTM_model(input.to(device), input_len)  # input_len
#         output = output.view(-1, len(label_dict))
#         label = label.view(-1)
#         loss = criterion(output, label.to(device))

#         loss.backward()
#         optimizer.step()
#         train_loss += loss.item() * input.size(1)

#     train_loss = train_loss / len(dataloader.dataset)
#     print('Epoch: {} \tTraining Loss: {:.6f}'.format(i, train_loss))
#     torch.save(BiLSTM_model.state_dict(), '/content/drive/My Drive/Colab
↪Notebooks/HW4-CSCI544/BiLSTM/BiLSTM_epoch_' + str(i) + '.pt')
# torch.save(BiLSTM_model.state_dict(), '/content/drive/My Drive/Colab
↪Notebooks/HW4-CSCI544/BiLSTM/blstm1.pt')
```

```
BiLSTM(
  (embedding): Embedding(30291, 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)
)
Epoch: 1        Training Loss: 2.937634
Epoch: 2        Training Loss: 2.050500
Epoch: 3        Training Loss: 1.529682
Epoch: 4        Training Loss: 1.135917
Epoch: 5        Training Loss: 0.876766
Epoch: 6        Training Loss: 0.680885
Epoch: 7        Training Loss: 0.523744
Epoch: 8        Training Loss: 0.412321
Epoch: 9        Training Loss: 0.334861
Epoch: 10       Training Loss: 0.279188
Epoch: 11       Training Loss: 0.218849
Epoch: 12       Training Loss: 0.176804
Epoch: 13       Training Loss: 0.149748
Epoch: 14       Training Loss: 0.132784
Epoch: 15       Training Loss: 0.130440
Epoch: 16       Training Loss: 0.137013
Epoch: 17       Training Loss: 0.109229
Epoch: 18       Training Loss: 0.085764
Epoch: 19       Training Loss: 0.087752
Epoch: 20       Training Loss: 0.072129
```

```
Epoch: 21       Training Loss: 0.058152
Epoch: 22       Training Loss: 0.060265
Epoch: 23       Training Loss: 0.059853
Epoch: 24       Training Loss: 0.058284
Epoch: 25       Training Loss: 0.057056
Epoch: 26       Training Loss: 0.041521
Epoch: 27       Training Loss: 0.040289
Epoch: 28       Training Loss: 0.035399
Epoch: 29       Training Loss: 0.034569
Epoch: 30       Training Loss: 0.033421
Epoch: 31       Training Loss: 0.029282
Epoch: 32       Training Loss: 0.029765
Epoch: 33       Training Loss: 0.034528
Epoch: 34       Training Loss: 0.041939
Epoch: 35       Training Loss: 0.035449
Epoch: 36       Training Loss: 0.024733
Epoch: 37       Training Loss: 0.018573
Epoch: 38       Training Loss: 0.022565
Epoch: 39       Training Loss: 0.019776
Epoch: 40       Training Loss: 0.020294
Epoch: 41       Training Loss: 0.019139
Epoch: 42       Training Loss: 0.027049
Epoch: 43       Training Loss: 0.034256
Epoch: 44       Training Loss: 0.023093
Epoch: 45       Training Loss: 0.016589
Epoch: 46       Training Loss: 0.014778
Epoch: 47       Training Loss: 0.020198
Epoch: 48       Training Loss: 0.017276
Epoch: 49       Training Loss: 0.014908
Epoch: 50       Training Loss: 0.012037
Epoch: 51       Training Loss: 0.013803
Epoch: 52       Training Loss: 0.014685
Epoch: 53       Training Loss: 0.018965
Epoch: 54       Training Loss: 0.014970
Epoch: 55       Training Loss: 0.011001
Epoch: 56       Training Loss: 0.011687
Epoch: 57       Training Loss: 0.013866
Epoch: 58       Training Loss: 0.015169
Epoch: 59       Training Loss: 0.011385
Epoch: 60       Training Loss: 0.010471
Epoch: 61       Training Loss: 0.010499
Epoch: 62       Training Loss: 0.009423
Epoch: 63       Training Loss: 0.006956
Epoch: 64       Training Loss: 0.008901
Epoch: 65       Training Loss: 0.006489
Epoch: 66       Training Loss: 0.009078
Epoch: 67       Training Loss: 0.007815
Epoch: 68       Training Loss: 0.014153
```

```
Epoch: 69       Training Loss: 0.015083
Epoch: 70       Training Loss: 0.017909
Epoch: 71       Training Loss: 0.013750
Epoch: 72       Training Loss: 0.021224
Epoch: 73       Training Loss: 0.015740
Epoch: 74       Training Loss: 0.009268
Epoch: 75       Training Loss: 0.017625
Epoch: 76       Training Loss: 0.014833
Epoch: 77       Training Loss: 0.010979
Epoch: 78       Training Loss: 0.012391
Epoch: 79       Training Loss: 0.017235
Epoch: 80       Training Loss: 0.016227
Epoch: 81       Training Loss: 0.012035
Epoch: 82       Training Loss: 0.012087
Epoch: 83       Training Loss: 0.012256
Epoch: 84       Training Loss: 0.010445
Epoch: 85       Training Loss: 0.010138
Epoch: 86       Training Loss: 0.009581
Epoch: 87       Training Loss: 0.016624
Epoch: 88       Training Loss: 0.017369
Epoch: 89       Training Loss: 0.024310
Epoch: 90       Training Loss: 0.024012
Epoch: 91       Training Loss: 0.017010
Epoch: 92       Training Loss: 0.016552
Epoch: 93       Training Loss: 0.014026
Epoch: 94       Training Loss: 0.015760
Epoch: 95       Training Loss: 0.015089
Epoch: 96       Training Loss: 0.017082
Epoch: 97       Training Loss: 0.013934
Epoch: 98       Training Loss: 0.010298
Epoch: 99       Training Loss: 0.016238
Epoch: 100      Training Loss: 0.009558
Epoch: 101      Training Loss: 0.008238
Epoch: 102      Training Loss: 0.008140
Epoch: 103      Training Loss: 0.011613
Epoch: 104      Training Loss: 0.007768
Epoch: 105      Training Loss: 0.008690
Epoch: 106      Training Loss: 0.007737
Epoch: 107      Training Loss: 0.004433
Epoch: 108      Training Loss: 0.005518
Epoch: 109      Training Loss: 0.006569
Epoch: 110      Training Loss: 0.005921
Epoch: 111      Training Loss: 0.006373
Epoch: 112      Training Loss: 0.005256
Epoch: 113      Training Loss: 0.006061
Epoch: 114      Training Loss: 0.005659
Epoch: 115      Training Loss: 0.005824
Epoch: 116      Training Loss: 0.005770
```

```
Epoch: 117      Training Loss: 0.006105
Epoch: 118      Training Loss: 0.007870
Epoch: 119      Training Loss: 0.007648
Epoch: 120      Training Loss: 0.004508
Epoch: 121      Training Loss: 0.005872
Epoch: 122      Training Loss: 0.004881
Epoch: 123      Training Loss: 0.004338
Epoch: 124      Training Loss: 0.009391
Epoch: 125      Training Loss: 0.016556
Epoch: 126      Training Loss: 0.014115
Epoch: 127      Training Loss: 0.021874
Epoch: 128      Training Loss: 0.017888
Epoch: 129      Training Loss: 0.013549
Epoch: 130      Training Loss: 0.013278
Epoch: 131      Training Loss: 0.009374
Epoch: 132      Training Loss: 0.007138
Epoch: 133      Training Loss: 0.013867
Epoch: 134      Training Loss: 0.006532
Epoch: 135      Training Loss: 0.006375
Epoch: 136      Training Loss: 0.012093
Epoch: 137      Training Loss: 0.009184
Epoch: 138      Training Loss: 0.005701
Epoch: 139      Training Loss: 0.006530
Epoch: 140      Training Loss: 0.005386
Epoch: 141      Training Loss: 0.007757
Epoch: 142      Training Loss: 0.004676
Epoch: 143      Training Loss: 0.005934
Epoch: 144      Training Loss: 0.005852
Epoch: 145      Training Loss: 0.007252
Epoch: 146      Training Loss: 0.004580
Epoch: 147      Training Loss: 0.005398
Epoch: 148      Training Loss: 0.004336
Epoch: 149      Training Loss: 0.005552
Epoch: 150      Training Loss: 0.004472
Epoch: 151      Training Loss: 0.003489
Epoch: 152      Training Loss: 0.003297
Epoch: 153      Training Loss: 0.003978
Epoch: 154      Training Loss: 0.004495
Epoch: 155      Training Loss: 0.004281
Epoch: 156      Training Loss: 0.005147
Epoch: 157      Training Loss: 0.002909
Epoch: 158      Training Loss: 0.004316
Epoch: 159      Training Loss: 0.003738
Epoch: 160      Training Loss: 0.002837
Epoch: 161      Training Loss: 0.005997
Epoch: 162      Training Loss: 0.008670
Epoch: 163      Training Loss: 0.004561
Epoch: 164      Training Loss: 0.003776
```

```
Epoch: 165      Training Loss: 0.003619
Epoch: 166      Training Loss: 0.002772
Epoch: 167      Training Loss: 0.005794
Epoch: 168      Training Loss: 0.002623
Epoch: 169      Training Loss: 0.002816
Epoch: 170      Training Loss: 0.002767
Epoch: 171      Training Loss: 0.002167
Epoch: 172      Training Loss: 0.003550
Epoch: 173      Training Loss: 0.002371
Epoch: 174      Training Loss: 0.003115
Epoch: 175      Training Loss: 0.004234
Epoch: 176      Training Loss: 0.004656
Epoch: 177      Training Loss: 0.008807
Epoch: 178      Training Loss: 0.006821
Epoch: 179      Training Loss: 0.008764
Epoch: 180      Training Loss: 0.006216
Epoch: 181      Training Loss: 0.004968
Epoch: 182      Training Loss: 0.008540
Epoch: 183      Training Loss: 0.006465
Epoch: 184      Training Loss: 0.012661
Epoch: 185      Training Loss: 0.011022
Epoch: 186      Training Loss: 0.016306
Epoch: 187      Training Loss: 0.008529
Epoch: 188      Training Loss: 0.008529
Epoch: 189      Training Loss: 0.008218
Epoch: 190      Training Loss: 0.011014
Epoch: 191      Training Loss: 0.008394
Epoch: 192      Training Loss: 0.013267
Epoch: 193      Training Loss: 0.007676
Epoch: 194      Training Loss: 0.007908
Epoch: 195      Training Loss: 0.013709
Epoch: 196      Training Loss: 0.011268
Epoch: 197      Training Loss: 0.012109
Epoch: 198      Training Loss: 0.012575
Epoch: 199      Training Loss: 0.007893
Epoch: 200      Training Loss: 0.007397
```

[25]:

```
BiLSTM_model = BiLSTM(vocab_size=len(word2idx),
                      embedding_dim=100,
                      linear_out_dim=128,
                      hidden_dim=256,
                      lstm_layers=1,
                      bidirectional=True,
                      dropout_val=0.33,
                      tag_size=len(label_dict),
                      glove_flag=False,
                      emb_matrix=[])
```

```
BiLSTM_model.load_state_dict(torch.load("/content/drive/My Drive/Colab␣
↪Notebooks/HW4-CSCI544/BiLSTM/blstm1.pt"))
BiLSTM_model.to(device)
```

[25]:
```
BiLSTM(
  (embedding): Embedding(30291, 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)
)
```

[26]:
```
#tesing on validation data
BiLSTM_dev = BiLSTM_DataLoader(dev_x_vec, dev_y_vec)
custom_collator = CustomCollator(word2idx, label_dict)
dataloader_dev = DataLoader(dataset=BiLSTM_dev,
                            batch_size=4,
                            shuffle=False,
                            drop_last=True,
                            collate_fn=custom_collator)



file = open("/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/dev1.out", 'w')
for dev_data, label, dev_data_len, label_data_len in dataloader_dev:

    pred = BiLSTM_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_dict[dev_data[i][j]]
                gold = rev_label_dict[label[i][j]]
                op = rev_label_dict[pred[i][j]]
                file.write(" ".join([str(j+1), str(word), gold, op]))
                file.write("\n")
        file.write("\n")

file.close()
```

[27]:
```
!perl '/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/conll03eval.txt' < '/
↪content/drive/My Drive/Colab Notebooks/HW4-CSCI544/dev1.out'
```

```
processed 51573 tokens with 5941 phrases; found: 5654 phrases; correct: 4444.
accuracy:  95.36%; precision:  78.60%; recall:  74.80%; FB1:  76.65
              LOC: precision:  89.31%; recall:  80.02%; FB1:  84.41  1646
             MISC: precision:  78.29%; recall:  77.44%; FB1:  77.86  912
              ORG: precision:  69.80%; recall:  71.57%; FB1:  70.67  1374
              PER: precision:  75.55%; recall:  70.63%; FB1:  73.01  1722
```

5.2 What are the precision, recall and F1 score on the dev data?

5.2.1 accuracy: 95.36%

5.2.2 precision: 78.60%

5.2.3 recall: 74.80%

5.2.4 FB1: 76.65

[30]:
```python
class BiLSTM_TestLoader(Dataset):
    def __init__(self, x):
        self.x = x

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

    def __getitem__(self, index):
        x_instance = torch.tensor(self.x[index])  # , dtype=torch.long
        # y_instance = torch.tensor(self.y[index])  # , dtype=torch.float
        return x_instance


class CustomTestCollator(object):

    def __init__(self, vocab, label):
        self.params = vocab
        self.label = label

    def __call__(self, batch):
        xx = batch
        x_len = [len(x) for x in xx]
        # y_len = [len(y) for y in yy]
        batch_max_len = max([len(s) for s in xx])
        batch_data = self.params['<pad>']*np.ones((len(xx), batch_max_len))
        # batch_labels = -1*np.zeros((len(xx), batch_max_len))
        for j in range(len(xx)):
            cur_len = len(xx[j])
            batch_data[j][:cur_len] = xx[j]
```

```
            # batch_labels[j][:cur_len] = yy[j]

        batch_data = torch.LongTensor(batch_data)
        batch_data = Variable(batch_data)

        return batch_data, x_len
```

[29]:
```
#Testing on Testing Dataset

BiLSTM_test = BiLSTM_TestLoader(test_x_vec)
custom_test_collator = CustomTestCollator(word2idx, label_dict)
dataloader_test = DataLoader(dataset=BiLSTM_test,
                                batch_size=4,
                                shuffle=False,
                                drop_last=True,
                                collate_fn=custom_test_collator)



file = open("/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/test1.out",␣
 ↪'w')
for test_data, test_data_len in dataloader_test:

    pred = BiLSTM_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_dict[test_data[i][j]]
                op = rev_label_dict[pred[i][j]]
                file.write(" ".join([str(j+1), word, op]))
                file.write("\n")

        file.write("\n")

file.close()
```

6 GloVe Word Embeddings

6.0.1 The function "create_emb_matrix" generates a matrix using the dictionary of the glove model, which is then used to input data into the Bilstm model. It is important to note that the glove model dictionary only contains lowercase words, and to address this issue, we adjust the embedding for titled words by adding a slight displacement value to each dimension of the lowercase counterpart.

[22]:
```
def create_emb_matrix(word_idx, emb_dict, dimension):

    emb_matrix = np.zeros((len(word_idx), dimension))
    for word, idx in word_idx.items():
        if word in emb_dict:
            emb_matrix[idx] = emb_dict[word]
        else:
            if word.lower() in emb_dict:
                emb_matrix[idx] = emb_dict[word.lower()] + 5e-3
            else:
                pass
    return emb_matrix
```

[23]:
```
glove = pd.read_csv('/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/glove.
↪6B.100d', sep=" ",quoting=3, header=None, index_col=0)
glove_emb = {key: val.values for key, val in glove.T.items()}
glove_vec = np.array([glove_emb[key] for key in glove_emb])
glove_emb["<pad>"] = np.zeros((100,), dtype="float64")
emb_matrix = create_emb_matrix(word_idx=word2idx, emb_dict=glove_emb,␣
↪dimension=100)

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

```
30291 100
```

6.1 Hyperparameters:

6.1.1 Embedding dimension = 100

6.1.2 Hidden dimension = 256

6.1.3 Linear Output dimension = 128

6.1.4 Bidirectional = True

6.1.5 Dropout = 0.33

6.1.6 Number of LSTM layers = 1

6.1.7 Batch Size = 8

6.1.8 Loss Function = Cross Entropy with class weights

6.1.9 Optimizer = SGD with Learning Rate = 0.1 and Momentum = 0.9

6.1.10 Epochs = 50

[24]:
```
# BiLSTM_glove_model = BiLSTM(vocab_size=len(word2idx),
#                         embedding_dim=100,
#                         linear_out_dim=128,
#                         hidden_dim=256,
#                         lstm_layers=1,
#                         bidirectional=True,
#                         dropout_val=0.33,
#                         tag_size=len(label_dict),
#                         glove_flag=True,
#                         emb_matrix=emb_matrix)

# BiLSTM_glove_model.to(device)
# print(BiLSTM_glove_model)

# BiLSTM_train = BiLSTM_DataLoader(train_x_vec, train_y_vec)
# custom_collator = CustomCollator(word2idx, label_dict)
# dataloader = DataLoader(dataset=BiLSTM_train,
#                           batch_size=8,
#                           drop_last=True,
#                           collate_fn=custom_collator)

# criterion = nn.CrossEntropyLoss(weight=class_wt)

# criterion = criterion.to(device)
# criterion.requres_grad = True
# optimizer = torch.optim.SGD(BiLSTM_glove_model.parameters(), lr=0.1,
 ↪momentum=0.9)

# scheduler = StepLR(optimizer, step_size=15, gamma=0.9)
# epochs = 50
```

```
# for i in range(1, epochs+1):
#     train_loss = 0.0

#     for input, label, input_len, label_len in dataloader:
#         optimizer.zero_grad()

#         output = BiLSTM_glove_model(input.to(device), input_len)  # input_len
#         output = output.view(-1, len(label_dict))
#         label = label.view(-1)
#         loss = criterion(output, label.to(device))

#         loss.backward()
#         optimizer.step()
#         train_loss += loss.item() * input.size(1)

#     train_loss = train_loss / len(dataloader.dataset)
#     print('Epoch: {} \tTraining Loss: {:.6f}'.format(i, train_loss))
#     torch.save(BiLSTM_glove_model.state_dict(), '/content/drive/My Drive/
 ↪Colab Notebooks/HW4-CSCI544/BiLSTM_glove/BiLSTM_glove_' + str(i) + '.pt')
# torch.save(BiLSTM_glove_model.state_dict(), '/content/drive/My Drive/Colab␣
 ↪Notebooks/HW4-CSCI544/BiLSTM_glove/blstm2.pt')
```

```
BiLSTM(
  (embedding): Embedding(30291, 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)
)
Epoch: 1        Training Loss: 0.770156
Epoch: 2        Training Loss: 0.358319
Epoch: 3        Training Loss: 0.250573
Epoch: 4        Training Loss: 0.191545
Epoch: 5        Training Loss: 0.150460
Epoch: 6        Training Loss: 0.121963
Epoch: 7        Training Loss: 0.100150
Epoch: 8        Training Loss: 0.082316
Epoch: 9        Training Loss: 0.067569
Epoch: 10       Training Loss: 0.055619
Epoch: 11       Training Loss: 0.046039
Epoch: 12       Training Loss: 0.039456
Epoch: 13       Training Loss: 0.034663
Epoch: 14       Training Loss: 0.028570
Epoch: 15       Training Loss: 0.024409
Epoch: 16       Training Loss: 0.020666
```

```
Epoch: 17       Training Loss: 0.018353
Epoch: 18       Training Loss: 0.016851
Epoch: 19       Training Loss: 0.013751
Epoch: 20       Training Loss: 0.011895
Epoch: 21       Training Loss: 0.012048
Epoch: 22       Training Loss: 0.009454
Epoch: 23       Training Loss: 0.010078
Epoch: 24       Training Loss: 0.007114
Epoch: 25       Training Loss: 0.008648
Epoch: 26       Training Loss: 0.008268
Epoch: 27       Training Loss: 0.006064
Epoch: 28       Training Loss: 0.005235
Epoch: 29       Training Loss: 0.006507
Epoch: 30       Training Loss: 0.004890
Epoch: 31       Training Loss: 0.003288
Epoch: 32       Training Loss: 0.003688
Epoch: 33       Training Loss: 0.003613
Epoch: 34       Training Loss: 0.003530
Epoch: 35       Training Loss: 0.003453
Epoch: 36       Training Loss: 0.003053
Epoch: 37       Training Loss: 0.002605
Epoch: 38       Training Loss: 0.002718
Epoch: 39       Training Loss: 0.002617
Epoch: 40       Training Loss: 0.002354
Epoch: 41       Training Loss: 0.001713
Epoch: 42       Training Loss: 0.001978
Epoch: 43       Training Loss: 0.002339
Epoch: 44       Training Loss: 0.001798
Epoch: 45       Training Loss: 0.001375
Epoch: 46       Training Loss: 0.001959
Epoch: 47       Training Loss: 0.002747
Epoch: 48       Training Loss: 0.004011
Epoch: 49       Training Loss: 0.003898
Epoch: 50       Training Loss: 0.002543
```

[25]:

```
BiLSTM_glove_model = BiLSTM(vocab_size=len(word2idx),
                        embedding_dim=100,
                        linear_out_dim=128,
                        hidden_dim=256,
                        lstm_layers=1,
                        bidirectional = True,
                        dropout_val=0.33,
                        tag_size=len(label_dict),
                        glove_flag = True,
                        emb_matrix=emb_matrix)
```

```
BiLSTM_glove_model.load_state_dict(torch.load("/content/drive/My Drive/Colab␣
 ↪Notebooks/HW4-CSCI544/BiLSTM_glove/BiLSTM_glove_50.pt"))
BiLSTM_glove_model.to(device)
```

[26]:
```
#predicting for validation dataset
BiLSTM_dev = BiLSTM_DataLoader(dev_x_vec, dev_y_vec)
custom_collator = CustomCollator(word2idx, label_dict)
dataloader_dev = DataLoader(dataset=BiLSTM_dev,
                            batch_size=8,
                            shuffle=False,
                            drop_last=True,
                            collate_fn=custom_collator)
print(label_dict)

res = []
file = open("/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/dev2.out", 'w')
for dev_data, label, dev_data_len, label_data_len in dataloader_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_dict[dev_data[i][j]]
                gold = rev_label_dict[label[i][j]]
                op = rev_label_dict[pred[i][j]]
                res.append((word, gold, op))
                file.write(" ".join([str(j + 1), str(word), gold, op]))
                file.write("\n")
        file.write("\n")
file.close()
```

```
{'B-ORG': 0, 'O': 1, 'B-MISC': 2, 'B-PER': 3, 'I-PER': 4, 'B-LOC': 5, 'I-ORG':
6, 'I-MISC': 7, 'I-LOC': 8}
```

[27]:
```
res = []
file = open("/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/dev2.out", 'w')
for dev_data, label, dev_data_len, label_data_len in dataloader_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:
#                 print(dev_data[i][j])
#                 print(rev_vocab_dict[dev_data[i][j]])
                word = rev_vocab_dict[dev_data[i][j]]
                gold = rev_label_dict[label[i][j]]
                op = rev_label_dict[pred[i][j]]
                res.append((word, gold, op))

                file.write(" ".join([str(j + 1), word, gold, op]))
                file.write("\n")
        file.write("\n")

file.close()
```

[28]:
```
!perl '/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/conll03eval.txt' < '/
↪content/drive/My Drive/Colab Notebooks/HW4-CSCI544/dev2.out'
```

```
processed 51573 tokens with 5941 phrases; found: 6000 phrases; correct: 5354.
accuracy:  98.02%; precision:  89.23%; recall:  90.12%; FB1:  89.67
              LOC: precision:  93.65%; recall:  94.77%; FB1:  94.21  1859
             MISC: precision:  82.92%; recall:  82.65%; FB1:  82.78  919
              ORG: precision:  83.06%; recall:  83.81%; FB1:  83.43  1352
              PER: precision:  92.41%; recall:  93.81%; FB1:  93.10  1870
```

6.2 What are the precision, recall and F1 score on the dev data?

6.2.1 accuracy: 98.02%

6.2.2 precision: 89.23%

6.2.3 recall: 90.12%

6.2.4 FB1: 89.67

[31]:
```
#predicting for testing dataset
BiLSTM_test = BiLSTM_TestLoader(test_x_vec)
custom_test_collator = CustomTestCollator(word2idx, label_dict)
dataloader_test = DataLoader(dataset=BiLSTM_test,
                                batch_size=1,
                                shuffle=False,
                                drop_last=True,
```

```
                                        collate_fn=custom_test_collator)

res = []
file = open("/content/drive/My Drive/Colab Notebooks/HW4-CSCI544/test2.out",␣
 ↪'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()
    # label = label.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_dict[test_data[i][j]]
                # gold = rev_label_dict[label[i][j]]
                op = rev_label_dict[pred[i][j]]
                res.append((word, op))
                file.write(" ".join([str(j + 1), word, op]))
                file.write("\n")
        file.write("\n")
file.close()
```

[31]: