NLP HW4 Report

To complete the assignment, I have followed the following actions at each step of the assignment.

1. Use of Unknown Tokens

First I have subdivided the unknown token <unk> based on the following characteristics:

- <unk_num> for digit
- <unk_mainly_num> for half digit
- <unk_verb> for verb
- <unk_adj> for adjective
- <unk_adv> for adverb
- <unk_all_lower> for lower case words
- <unk_all_upper> for upper case words
- <unk_initial_upper> for title case words
- <unk_contain_num> for words containing numbers
- <unk> for all other cases

2. Vocabulary & Dataset Preparation

- Then I prepared the vocabulary for input data considering all the unknown tokens.
- After that I created dictionaries to store Word_to_Index, NER_to_Index, Index_to_Word and Index_to_Ner.
- I used the above dictionaries to create input (data_X) and target(data_y) sequences and finally prepared X_train and Y_train dataset.

3. BLSTM Model & Hyperparameters

First defined the architecture of the BLSTM model and then set the hyperparameters given in the assignment. After that, I tried the following changes with hyperparameters and the model:

- I tried various combinations of batch sizes to run epochs in a reasonable amount of time for both Task 1 and Task 2 and finally generated the best results with a batch size of 10 on 20 epochs for Task 1 and batch size of 16 on 35 epochs for Task 2.
- I tried changing the learning rate from 0.02 to 0.6 to get better results and finally got best results with lr=0.1 and lr=0.23 for Task 1 and Task 2 respectively.
- I worked with different momentum sizes and got the best results for momentum 0.9.
- I also tried to change the loss function for nn.CrossEntropyLoss.

4. Results Of Task 1

Results of Task 1 after running **!perl conll03eval.txt < dev1_perl.out** the command.

Accuracy	**Precision**	**Recall**	**F1-score**
96.19%	81.42%	76.86%	78.94%

5.Results Of Task 2

Results of Task 1 after running **!perl conll03eval.txt < dev2_perl.out** the command.

Accuracy	**Precision**	**Recall**	**F1-score**
97.59%	88.99%	85.17%	87.04

TASK 1

In [1]:
```
import pickle
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import pandas as pd
import numpy as np
import string
from torch.utils.data import TensorDataset, DataLoader
from torchtext import data
from torchtext import datasets
import time
```

In [2]:
```
noun_list = ["tion", "ity", "er", "ness", "ism", "ment", "ant", "ship", "ag
verb_list = ["ate", "ify", "ize", "ise"]
adj_list = ["able", "ible", 'ant', 'ent', 'ive', "al","ial","an","ian","ish
adv_list = ["ly","lng","ward", "wards", "way", "ways", "wise"]

def processUnknowns(word):
    num = 0
    for char in word:
      if char.isdigit():
          num += 1

    fraction = num / float(len(word))

    if word.isdigit():
        return "<unk_num>"
    elif fraction > 0.5:
        return "<unk_mainly_num>"
    elif any(word.endswith(suffix) for suffix in verb_list):
        return "<unk_verb>"
    elif any(word.endswith(suffix) for suffix in adj_list):
        return "<unk_adj>"
    elif any(word.endswith(suffix) for suffix in adv_list):
        return "<unk_adv>"
    elif word.islower():
        return "<unk_all_lower>"
    elif word.isupper():
        return "<unk_all_upper>"
    elif word[0].isupper():
        return "<unk_initial_upper>"
    elif any(char.isdigit() for char in word):
        return "<unk_contain_num>"
    else:
        return "<unk>"
```

In [3]:
```
def prepareVocabulary(file, min_count=2):
    vocab, NER_set, sentence, sentences = {}, set(), [], []
    with open(file, "r") as train:
        for line in train:
            if not line.split():
                sentences.append(sentence)
                sentence =[]
                continue
            word_type, NER_type = line.split(" ")[1], line.split(" ")[2].st
            if word_type not in vocab:
                vocab[word_type] = 1
            else:
                vocab[word_type]+=1
            sentence.append([word_type,NER_type])
            NER_set.add(NER_type)
        sentences.append(sentence)

        vocab['<unk>'], vocab['<unk_mainly_num>'] = 0,0
        vocab['<unk_num>'], vocab['<unk_contain_num>'] = 0,0
        vocab['<unk_verb>'], vocab['<unk_adj>'] = 0,0
        vocab['<unk_adv>'], vocab['<unk_all_lower>'] = 0,0
        vocab['<unk_all_upper>'], vocab['<unk_initial_upper>'] = 0,0

        delete = []
        for word, occurrences in vocab.items():
            if occurrences >= min_count:
                continue
            else:
                new_token = processUnknowns(word)
                vocab[new_token] += occurrences
                delete.append(word)

        for i in delete:
            del vocab[i]

    return vocab, NER_set, sentences
```

In [4]:
```
vocab, NER_set, sentences = prepareVocabulary('/content/drive/MyDrive/Colab
sortedVocabulary = sorted(vocab.items(), key=lambda x:x[1], reverse=True)
Word_to_Index = {w: i+1 for i, (w, n) in enumerate(sortedVocabulary)}
Word_to_Index['PAD'] = 0
print(len(Word_to_Index))
```

```
11994
```

In [5]:
```
NER_to_Index = {}
i = 0
for ner in NER_set:
    NER_to_Index[ner] = i
    i += 1

Index_to_Word = {}
for key, value in Word_to_Index.items():
    Index_to_Word[value] = key

Index_to_Ner = {}
for key, value in NER_to_Index.items():
    Index_to_Ner[value] = key

print(len(Word_to_Index), len(NER_to_Index))
```

```
11994 9
```

In [6]:
```
data_X = []

for s in sentences:
    temp_X = []
    for w, label in s:
        if w in Word_to_Index:
            temp_X.append(Word_to_Index.get(w))
        else:
            unk = processUnknowns(w)
            temp_X.append(Word_to_Index[unk])
    data_X.append(temp_X)

data_y = []
for s in sentences:
    temp_y = []
    for w, label in s:
        temp_y.append(NER_to_Index.get(label))
    data_y.append(temp_y)
```

In [7]:
```
def padding_for_words(dataset, max_len):
    for i, line in enumerate(dataset):
        if len(line) > max_len:
            dataset[i] = line[:max_len]
        elif len(line) < max_len:
            dataset[i] = line[:len(line)] + [0]*(max_len-len(line))

    return dataset

def padding_for_NER(dataset, max_len):
    for i, line in enumerate(dataset):
        if len(line) > max_len:
            dataset[i] = line[:max_len]
        elif len(line) < max_len:
            dataset[i] = line[:len(line)] + [-100]*(max_len-len(line))

    return dataset

data_X = padding_for_words(data_X, 130)
data_y = padding_for_NER(data_y, 130)
X_train = torch.LongTensor(data_X)
Y_train = torch.LongTensor(data_y)
ds_train = TensorDataset(X_train, Y_train)
loader_train = DataLoader(ds_train, batch_size=10, shuffle=False)

print(len(Word_to_Index), len(NER_to_Index))
```

```
11994 9
```

In [8]:
```
isCuda = torch.cuda.is_available()

if isCuda:
    device = torch.device("cuda")
    print("-- cuda --")
else:
    device = torch.device("cpu")
    print("-- cpu --")
```

```
-- cuda --
```

In [9]:
```
class BLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, first_output_
        super().__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_id
        self.blstm = nn.LSTM(embedding_dim, hidden_dim, num_layers = num_la
        self.fc1 = nn.Linear(hidden_dim * 2, first_output_dim)
        self.dropout = nn.Dropout(drop_out)
        self.activation = nn.ELU()
        self.fc2 = nn.Linear(first_output_dim, output_dim)

    def forward(self, text):
        embedded = self.dropout(self.embedding(text))
        outputs, (hidden, cell) = self.blstm(embedded)
        outputs = self.dropout(outputs)
        outputs = self.activation(self.fc1(outputs))
        predictions = self.fc2(outputs)
        return predictions
```

In [10]:
```
INPUT_DIM = len(Word_to_Index)
EMBEDDING_DIM = 100
HIDDEN_DIM = 256
FIRST_OUTPUT_DIM = 128
OUTPUT_DIM = len(NER_to_Index)
N_LAYERS = 1
BIDIRECTIONAL = True
DROPOUT = 0.33

model = BLSTM(INPUT_DIM,
              EMBEDDING_DIM,
              HIDDEN_DIM,
              FIRST_OUTPUT_DIM,
              OUTPUT_DIM,
              N_LAYERS,
              BIDIRECTIONAL,
              DROPOUT)

model.to(device)
print(len(Word_to_Index), len(NER_to_Index))
```

```
11994 9
```

In [11]:
```
def categoricalAccuracy(preds, y, tag_pad_idx, text, predict_table):
    tot = 0
    correct = 0
    max_preds = preds.argmax(dim = 1, keepdim = True)
    for predict, real, word in zip(max_preds, y, text):
        if real.item() == tag_pad_idx:
            continue
        else:
            predict_table.append((word.item(), predict.item(), real.item()))
            if real.item() == predict.item():
                correct += 1
            tot += 1
    return tot, correct, predict_table
```

In [12]:
```
def trainModel(model, dataloader, predict_table):

    epoch_loss = 0
    epoch_acc = 0
    epoch_tot = 0
    model.train()

    for text, tags in dataloader:

        optimizer.zero_grad()
        tags = tags.to(device)
        text = text.to(device)
        predictions = model(text)
        predictions = predictions.view(-1, predictions.shape[-1])
        tags = tags.view(-1)

        loss = criterion(predictions, tags)

        tot, correct, predict_table = categoricalAccuracy(predictions, tags

        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += correct
        epoch_tot +=tot

    return epoch_loss / len(dataloader), epoch_acc / epoch_tot, predict_tab
```

In [13]:
```
def evaluateModel(model, dataloader, predict_table):

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

    with torch.no_grad():

        for text, tags in dataloader:
            tags = tags.to(device)
            text = text.to(device)
            predictions = model(text)

            predictions = predictions.view(-1, predictions.shape[-1])
            tags = tags.view(-1)

            loss = criterion(predictions, tags)

            tot, correct, predict_table = categoricalAccuracy(predictions,

            epoch_loss += loss.item()
            epoch_acc += correct
            epoch_tot +=tot

    return epoch_loss / len(dataloader), epoch_acc / epoch_tot, predict_tab
```

In [14]:
```
dev_sentences = []
sentence=[]
cnt=0
with open('/content/drive/MyDrive/Colab Notebooks/data/dev', "r") as dev:
    for line in dev:
        if not line.split():
            dev_sentences.append(sentence)
            sentence =[]
            continue
        word_type, NER_type = line.split(" ")[1], line.split(" ")[2].strip(
        cnt+=1
        sentence.append([word_type,NER_type])
    dev_sentences.append(sentence)
```

In [15]:
```
dev_X = []
for s in dev_sentences:
    temp_X = []
    for w, label in s:
        if w in Word_to_Index:
            temp_X.append(Word_to_Index.get(w))
        else:
            unk = processUnknowns(w)
            temp_X.append(Word_to_Index[unk])
    dev_X.append(temp_X)

dev_y = []
for s in dev_sentences:
    temp_y = []
    for w, label in s:
        temp_y.append(NER_to_Index.get(label))
    dev_y.append(temp_y)

dev_X = padding_for_words(dev_X, 130)
dev_y = padding_for_NER(dev_y, 130)

X_dev = torch.LongTensor(dev_X)
Y_dev = torch.LongTensor(dev_y)
ds_dev = TensorDataset(X_dev, Y_dev)
loader_dev = DataLoader(ds_dev, batch_size=10, shuffle=False)
```

In [16]:
```
epochs = 20
tag_pad_idx=-100
optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, nesterov=Tr
criterion = nn.CrossEntropyLoss(ignore_index= -100)
best_valid_loss = float('inf')

for epoch in range(epochs):
    train_predict_table = []
    test_predict_table = []

    train_loss, train_acc, train_predict_table = trainModel(model, loader_t
    valid_loss, valid_acc, valid_predict_table = evaluateModel(model, loade

    if valid_loss <= best_valid_loss:
        best_valid_loss = valid_loss
        best_predict_table = valid_predict_table
        torch.save(model.state_dict(), './blstm1.pt')

    print(f'Epoch: {epoch+1:02}')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}
```

```
Epoch: 01
        Train Loss: 0.644 | Train Acc: 85.03%
         Val. Loss: 0.448 |  Val. Acc: 88.28%
Epoch: 02
        Train Loss: 0.442 | Train Acc: 87.91%
         Val. Loss: 0.307 |  Val. Acc: 91.27%
Epoch: 03
        Train Loss: 0.348 | Train Acc: 89.77%
         Val. Loss: 0.244 |  Val. Acc: 93.03%
Epoch: 04
        Train Loss: 0.296 | Train Acc: 90.91%
         Val. Loss: 0.210 |  Val. Acc: 93.89%
Epoch: 05
        Train Loss: 0.261 | Train Acc: 91.80%
         Val. Loss: 0.198 |  Val. Acc: 94.19%
Epoch: 06
        Train Loss: 0.234 | Train Acc: 92.50%
         Val. Loss: 0.179 |  Val. Acc: 94.66%
Epoch: 07
        Train Loss: 0.217 | Train Acc: 92.94%
         Val. Loss: 0.165 |  Val. Acc: 95.01%
Epoch: 08
        Train Loss: 0.201 | Train Acc: 93.36%
         Val. Loss: 0.159 |  Val. Acc: 95.22%
Epoch: 09
        Train Loss: 0.190 | Train Acc: 93.70%
         Val. Loss: 0.151 |  Val. Acc: 95.47%
Epoch: 10
        Train Loss: 0.180 | Train Acc: 93.90%
         Val. Loss: 0.147 |  Val. Acc: 95.61%
Epoch: 11
        Train Loss: 0.171 | Train Acc: 94.19%
         Val. Loss: 0.143 |  Val. Acc: 95.69%
Epoch: 12
        Train Loss: 0.165 | Train Acc: 94.39%
         Val. Loss: 0.138 |  Val. Acc: 95.94%
Epoch: 13
        Train Loss: 0.157 | Train Acc: 94.52%
         Val. Loss: 0.144 |  Val. Acc: 95.74%
Epoch: 14
        Train Loss: 0.151 | Train Acc: 94.79%
         Val. Loss: 0.139 |  Val. Acc: 95.87%
Epoch: 15
        Train Loss: 0.145 | Train Acc: 94.94%
         Val. Loss: 0.133 |  Val. Acc: 96.03%
Epoch: 16
        Train Loss: 0.140 | Train Acc: 95.09%
         Val. Loss: 0.134 |  Val. Acc: 96.14%
Epoch: 17
        Train Loss: 0.135 | Train Acc: 95.18%
         Val. Loss: 0.136 |  Val. Acc: 96.09%
Epoch: 18
        Train Loss: 0.131 | Train Acc: 95.35%
         Val. Loss: 0.132 |  Val. Acc: 96.19%
Epoch: 19
        Train Loss: 0.127 | Train Acc: 95.49%
         Val. Loss: 0.132 |  Val. Acc: 96.25%
```

```
Epoch: 20
        Train Loss: 0.123 | Train Acc: 95.58%
         Val. Loss: 0.133 |  Val. Acc: 96.12%
```

In [17]:
```
term = [int(x[0]) for x in best_predict_table]
y_pred = [int(x[1]) for x in best_predict_table]
i=0
newfile = open('./dev1.out', "w")
with open('/content/drive/MyDrive/Colab Notebooks/data/dev', "r") as train:
    for line in train:
        if not line.split():
            newfile.write('\n')
            continue
        index, word_type = line.split(" ")[0], line.split(" ")[1].strip('\n
        newfile.write(str(index)+' '+str(word_type)+' '+str(Index_to_Ner[y_
        i += 1
newfile.close()

i=0
newfile = open('./dev1_perl.out', "w")
with open('/content/drive/MyDrive/Colab Notebooks/data/dev', "r") as train:
    for line in train:
        if not line.split():
            newfile.write('\n')
            continue
        index, word_type, NER_type = line.split(" ")[0], line.split(" ")[1]
        newfile.write(str(index)+' '+str(word_type)+' '+str(NER_type)+' '+s
        i += 1
newfile.close()
```

In [18]:
```
!perl conll03eval.txt < dev1_perl.out
```

```
processed 51578 tokens with 5942 phrases; found: 5609 phrases; correct: 4
567.
accuracy:  96.19%; precision:  81.42%; recall:  76.86%; FB1:  79.08
              LOC: precision:  87.78%; recall:  84.10%; FB1:  85.90  1760
             MISC: precision:  73.21%; recall:  76.46%; FB1:  74.80  963
              ORG: precision:  72.74%; recall:  64.28%; FB1:  68.25  1185
              PER: precision:  85.54%; recall:  78.99%; FB1:  82.13  1701
```

In [19]:
```
def categoricalEvaluate(preds, text, predictTable):

    max_preds = preds.argmax(dim = 1, keepdim = True)
    for predict, word in zip(max_preds, text):
        if word == 0:
            continue
        else:
            predictTable.append((word, predict[0]))

    return predictTable
```

In [20]:
```
def evaluateModel(model, loader, predictTable):

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

    with torch.no_grad():

        for text in loader:
            text = text.to(device)
            predictions = model(text)
            predictions = predictions.view(-1, predictions.shape[-1])

            predictTable = categoricalEvaluate(predictions, text.view(-1),

    return predictTable
```

In [21]:
```
test_X = []
sentence = []
cnt=0
with open('/content/drive/MyDrive/Colab Notebooks/data/dev', "r") as test:
    for line in test:
        if not line.split():
            test_X.append(sentence)
            sentence = []
            continue
        word_type = line.split(" ")[1]
        if word_type in Word_to_Index:
            sentence.append(Word_to_Index.get(word_type))
        else:
            unk = processUnknowns(word_type)
            sentence.append(Word_to_Index.get(unk))
    test_X.append(sentence)

test_X = padding_for_words(test_X, 130)
X_test = torch.LongTensor(test_X)
loader_test = DataLoader(X_test, batch_size=10, shuffle=False)

evaluate_predict_table2 = []
model = BLSTM(INPUT_DIM,
              EMBEDDING_DIM,
              HIDDEN_DIM,
              FIRST_OUTPUT_DIM,
              OUTPUT_DIM,
              N_LAYERS,
              BIDIRECTIONAL,
              DROPOUT)
model.to(device)
model.load_state_dict(torch.load('./blstm1.pt'))
prediction_table = evaluateModel(model, loader_test, evaluate_predict_table

term = [int(x[0]) for x in evaluate_predict_table2]
y_pred = [int(x[1]) for x in evaluate_predict_table2]

i=0
newfile = open('./test1.out', "w")
with open('/content/drive/MyDrive/Colab Notebooks/data/dev', "r") as test:
    for line in test:
        if not line.split():
            newfile.write('\n')
            continue
        index, word_type = line.split(" ")[0], line.split(" ")[1].strip('\n
        for_tag = Index_to_Ner[y_pred[i]]
        newfile.write(str(index)+' '+str(word_type)+' '+for_tag+'\n')
        i += 1
newfile.close()
```

In [22]:
```
import pickle
with open('./vocab_dictionary.pickle','wb') as fw1:
    pickle.dump(Word_to_Index, fw1)
with open('./ner_dictionary.pickle','wb') as fw2:
    pickle.dump(NER_to_Index, fw2)
with open('./int_vocab_dictionary.pickle','wb') as fw3:
    pickle.dump(Index_to_Word, fw3)
with open('./int_ner_dictionary.pickle','wb') as fw4:
    pickle.dump(Index_to_Ner, fw4)
with open('./loader_train.pickle','wb') as fw5:
    pickle.dump(loader_train, fw5)
with open('./loader_dev.pickle','wb') as fw6:
    pickle.dump(loader_dev, fw6)
with open('./loader_test.pickle','wb') as fw7:
    pickle.dump(loader_test, fw7)
```

In [25]:
```
checkpoint = {'INPUT_DIM':len(Word_to_Index),
              'EMBEDDING_DIM':100,
              'HIDDEN_DIM':256,
              'FIRST_OUTPUT_DIM':128,
              'OUTPUT_DIM':len(NER_to_Index),
              'N_LAYERS':1,
              'BIDIRECTIONAL':True,
              'DROPOUT':0.33,
              'state_dict': model.state_dict()}

torch.save(checkpoint, './checkpoint.pth')
```

In [ ]:
```

```

TASK 2

In [1]:
```
import pickle
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import pandas as pd
import numpy as np
import string
from torch.utils.data import TensorDataset, DataLoader
from torchtext import data
from torchtext import datasets
import time
```

In [ ]:
```
noun_list = ["tion", "ity", "er", "ness", "ism", "ment", "ant", "ship", "ag
verb_list = ["ate", "ify", "ize", "ise"]
adj_list = ["able", "ible", 'ant', 'ent', 'ive', "al","ial","an","ian","ish
adv_list = ["ly","lng","ward", "wards", "way", "ways", "wise"]

def processUnknowns(word):
    num = 0
    for char in word:
      if char.isdigit():
          num += 1

    fraction = num / float(len(word))

    if word.isdigit():
        return "<unk_num>"
    elif fraction > 0.5:
        return "<unk_mainly_num>"
    elif any(word.endswith(suffix) for suffix in verb_list):
        return "<unk_verb>"
    elif any(word.endswith(suffix) for suffix in adj_list):
        return "<unk_adj>"
    elif any(word.endswith(suffix) for suffix in adv_list):
        return "<unk_adv>"
    elif word.islower():
        return "<unk_all_lower>"
    elif word.isupper():
        return "<unk_all_upper>"
    elif word[0].isupper():
        return "<unk_initial_upper>"
    elif any(char.isdigit() for char in word):
        return "<unk_contain_num>"
    else:
        return "<unk>"
```

In [2]:
```
def prepareVocabulary(file, min_count=2):
    vocab, NER_set, sentence, sentences = {}, set(), [], []
    with open(file, "r") as train:
        for line in train:
            if not line.split():
                sentences.append(sentence)
                sentence =[]
                continue
            word_type, NER_type = line.split(" ")[1], line.split(" ")[2].st
            if word_type not in vocab:
                vocab[word_type] = 1
            else:
                vocab[word_type]+=1
            sentence.append([word_type,NER_type])
            NER_set.add(NER_type)
        sentences.append(sentence)

        vocab['<unk>'], vocab['<unk_mainly_num>'] = 0,0
        vocab['<unk_num>'], vocab['<unk_contain_num>'] = 0,0
        vocab['<unk_verb>'], vocab['<unk_adj>'] = 0,0
        vocab['<unk_adv>'], vocab['<unk_all_lower>'] = 0,0
        vocab['<unk_all_upper>'], vocab['<unk_initial_upper>'] = 0,0

        delete = []
        for word, occurrences in vocab.items():
            if occurrences >= min_count:
                continue
            else:
                new_token = processUnknowns(word)
                vocab[new_token] += occurrences
                delete.append(word)

        for i in delete:
            del vocab[i]

    return vocab, NER_set, sentences
```

In [3]:
```
vocab, NER_set, sentences = prepareVocabulary('/content/drive/MyDrive/Colab
sortedVocabulary = sorted(vocab.items(), key=lambda x:x[1], reverse=True)
Word_to_Index = {w: i+1 for i, (w, n) in enumerate(sortedVocabulary)}
Word_to_Index['PAD'] = 0
print(len(Word_to_Index))
```

```
11994
```

In [4]:
```
NER_to_Index = {}
i = 0
for ner in NER_set:
    NER_to_Index[ner] = i
    i += 1

Index_to_Word = {}
for key, value in Word_to_Index.items():
    Index_to_Word[value] = key

Index_to_Ner = {}
for key, value in NER_to_Index.items():
    Index_to_Ner[value] = key

print(len(Word_to_Index), len(NER_to_Index))
```

```
11994 9
```

In [5]:
```
data_X = []

for s in sentences:
    temp_X = []
    for w, label in s:
        if w in Word_to_Index:
            temp_X.append(Word_to_Index.get(w))
        else:
            unk = processUnknowns(w)
            temp_X.append(Word_to_Index[unk])
    data_X.append(temp_X)

data_y = []
for s in sentences:
    temp_y = []
    for w, label in s:
        temp_y.append(NER_to_Index.get(label))
    data_y.append(temp_y)
```

In [6]:
```
def padding_for_words(dataset, max_len):
    for i, line in enumerate(dataset):
        if len(line) > max_len:
            dataset[i] = line[:max_len]
        elif len(line) < max_len:
            dataset[i] = line[:len(line)] + [0]*(max_len-len(line))

    return dataset

def padding_for_NER(dataset, max_len):
    for i, line in enumerate(dataset):
        if len(line) > max_len:
            dataset[i] = line[:max_len]
        elif len(line) < max_len:
            dataset[i] = line[:len(line)] + [-100]*(max_len-len(line))

    return dataset

data_X = padding_for_words(data_X, 130)
data_y = padding_for_NER(data_y, 130)
X_train = torch.LongTensor(data_X)
Y_train = torch.LongTensor(data_y)
ds_train = TensorDataset(X_train, Y_train)
loader_train = DataLoader(ds_train, batch_size=16, shuffle=False)

print(len(Word_to_Index), len(NER_to_Index))
```

```
11994 9
```

In [7]:
```
import gzip
import os
import shutil

with gzip.open('/content/drive/MyDrive/Colab Notebooks/glove.6B.100d.gz', '
    with open('glove.6B.100d', 'wb') as f_out:
        shutil.copyfileobj(f_in, f_out)

embedding_dict = dict()
f = open(os.path.join('glove.6B.100d'), encoding='utf-8')
for line in f:
    word_vector = line.split()
    word = word_vector[0]
    word_vector_arr = np.asarray(word_vector[1:], dtype='float32')
    embedding_dict[word] = word_vector_arr
f.close()

embedding_dim = 100
embedding_matrix = np.zeros((len(Word_to_Index), embedding_dim))

for word, i in Word_to_Index.items():
    embedding_vector = embedding_dict.get(word.lower())
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_matrix = torch.LongTensor(embedding_matrix)
```

In [8]:
```
is_cuda = torch.cuda.is_available()
if is_cuda:
    device = torch.device("cuda")
    print("-- cuda ---")
else:
    device = torch.device("cpu")
    print("--- cpu ---")
```

```
-- cuda ---
```

In [9]:
```
class BLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, first_output_
        super().__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_id
        self.blstm = nn.LSTM(embedding_dim, hidden_dim, num_layers = num_la
        self.fc1 = nn.Linear(hidden_dim * 2, first_output_dim)
        self.dropout = nn.Dropout(drop_out)
        self.activation = nn.ELU()
        self.fc2 = nn.Linear(first_output_dim, output_dim)

    def forward(self, text):
        embedded = self.dropout(self.embedding(text))
        outputs, (hidden, cell) = self.blstm(embedded)
        outputs = self.dropout(outputs)
        outputs = self.activation(self.fc1(outputs))
        predictions = self.fc2(outputs)
        return predictions
```

In [10]:
```
INPUT_DIM = len(Word_to_Index)
EMBEDDING_DIM = 100
HIDDEN_DIM = 256
FIRST_OUTPUT_DIM = 128
OUTPUT_DIM = len(NER_to_Index)
N_LAYERS = 1
BIDIRECTIONAL = True
DROPOUT = 0.33

model = BLSTM(INPUT_DIM,
              EMBEDDING_DIM,
              HIDDEN_DIM,
              FIRST_OUTPUT_DIM,
              OUTPUT_DIM,
              N_LAYERS,
              BIDIRECTIONAL,
              DROPOUT)

model.to(device)
model.embedding.weight.data.copy_(embedding_matrix)

print(len(Word_to_Index), len(NER_to_Index))
```

```
11994 9
```

In [11]:
```
def trainModel(model, dataloader, predict_table):
    epoch_loss = 0
    epoch_acc = 0
    epoch_tot = 0
    model.train()

    for text, tags in dataloader:

        optimizer.zero_grad()
        tags = tags.to(device)
        text = text.to(device)
        predictions = model(text)
        predictions = predictions.view(-1, predictions.shape[-1])
        tags = tags.view(-1)
        loss = criterion(predictions, tags)
        tot, correct, predict_table = categoricalAccuracy(predictions, tags
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += correct
        epoch_tot +=tot

    return epoch_loss / len(dataloader), epoch_acc / epoch_tot, predict_tab
```

In [12]:
```
def categoricalAccuracy(preds, y, tag_pad_idx, text, predict_table):
    tot = 0
    correct = 0
    max_preds = preds.argmax(dim = 1, keepdim = True)
    for predict, real, word in zip(max_preds, y, text):
        if real.item() == tag_pad_idx:
            continue
        else:
            predict_table.append((word.item(), predict.item(), real.item()))
            if real.item() == predict.item():
                correct += 1
            tot += 1
    return tot, correct, predict_table

def model_evaluate(model, dataloader, predict_table):
    epoch_loss = 0
    epoch_acc = 0
    epoch_tot = 0
    model.eval()

    with torch.no_grad():

        for text, tags in dataloader:
            tags = tags.to(device)
            text = text.to(device)

            predictions = model(text)

            predictions = predictions.view(-1, predictions.shape[-1])
            tags = tags.view(-1)

            loss = criterion(predictions, tags)

            tot, correct, predict_table = categoricalAccuracy(predictions,

            epoch_loss += loss.item()
            epoch_acc += correct
            epoch_tot +=tot

    return epoch_loss / len(dataloader), epoch_acc / epoch_tot, predict_tab
```

In [14]:
```
dev_sentences = []
sentence=[]
cnt=0
with open('/content/drive/MyDrive/Colab Notebooks/data/dev', "r") as dev:
    for line in dev:
        if not line.split():
            dev_sentences.append(sentence)
            sentence =[]
            continue
        word_type, NER_type = line.split(" ")[1], line.split(" ")[2].strip(
        cnt+=1
        sentence.append([word_type,NER_type])
    dev_sentences.append(sentence)

dev_X = []
for s in dev_sentences:
    temp_X = []
    for w, label in s:
        if w in Word_to_Index:
            temp_X.append(Word_to_Index.get(w))
        else:
            unk = processUnknowns(w)
            temp_X.append(Word_to_Index[unk])
    dev_X.append(temp_X)

dev_y = []
for s in dev_sentences:
    temp_y = []
    for w, label in s:
        temp_y.append(NER_to_Index.get(label))
    dev_y.append(temp_y)

dev_X = padding_for_words(dev_X, 130)
dev_y = padding_for_NER(dev_y, 130)

X_dev = torch.LongTensor(dev_X)
Y_dev = torch.LongTensor(dev_y)
ds_dev = TensorDataset(X_dev, Y_dev)
loader_dev = DataLoader(ds_dev, batch_size=16, shuffle=False)
```

In [38]:
```
print("---- ", type(loader_dev))
```

```
----  <class 'torch.utils.data.dataloader.DataLoader'>
```

In [16]:
```
N_EPOCHS = 35
tag_pad_idx=-100
optimizer = optim.SGD(model.parameters(), lr=0.23, momentum=0.9, nesterov=T
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', pa
criterion = nn.CrossEntropyLoss(ignore_index= -100)
best_valid_loss = float('inf')
for epoch in range(N_EPOCHS):
    train_predict_table = []
    test_predict_table = []

    train_loss, train_acc, train_predict_table = trainModel(model, loader_t
    valid_loss, valid_acc, valid_predict_table = model_evaluate(model, load

    if valid_loss <= best_valid_loss:
        best_valid_loss = valid_loss
        best_predict_table = valid_predict_table
        torch.save(model.state_dict(), './blstm2.pt')

    scheduler.step(valid_loss)

    print(f'Epoch: {epoch+1:02}')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}
```

```
Epoch: 01
        Train Loss: 0.436 | Train Acc: 88.63%
         Val. Loss: 0.238 |  Val. Acc: 93.47%
Epoch: 02
        Train Loss: 0.201 | Train Acc: 93.84%
         Val. Loss: 0.149 |  Val. Acc: 95.70%
Epoch: 03
        Train Loss: 0.129 | Train Acc: 95.85%
         Val. Loss: 0.122 |  Val. Acc: 96.27%
Epoch: 04
        Train Loss: 0.099 | Train Acc: 96.76%
         Val. Loss: 0.113 |  Val. Acc: 96.45%
Epoch: 05
        Train Loss: 0.080 | Train Acc: 97.37%
         Val. Loss: 0.112 |  Val. Acc: 96.36%
Epoch: 06
        Train Loss: 0.067 | Train Acc: 97.77%
         Val. Loss: 0.103 |  Val. Acc: 96.61%
Epoch: 07
        Train Loss: 0.059 | Train Acc: 98.03%
         Val. Loss: 0.100 |  Val. Acc: 96.72%
Epoch: 08
        Train Loss: 0.050 | Train Acc: 98.32%
         Val. Loss: 0.097 |  Val. Acc: 96.97%
Epoch: 09
        Train Loss: 0.046 | Train Acc: 98.47%
         Val. Loss: 0.101 |  Val. Acc: 96.92%
Epoch: 10
        Train Loss: 0.041 | Train Acc: 98.60%
         Val. Loss: 0.097 |  Val. Acc: 97.00%
Epoch: 11
        Train Loss: 0.038 | Train Acc: 98.74%
         Val. Loss: 0.103 |  Val. Acc: 96.96%
Epoch: 12
        Train Loss: 0.035 | Train Acc: 98.82%
         Val. Loss: 0.098 |  Val. Acc: 97.05%
Epoch: 13
        Train Loss: 0.032 | Train Acc: 98.88%
         Val. Loss: 0.100 |  Val. Acc: 97.15%
Epoch: 14
        Train Loss: 0.027 | Train Acc: 99.06%
         Val. Loss: 0.090 |  Val. Acc: 97.44%
Epoch: 15
        Train Loss: 0.025 | Train Acc: 99.10%
         Val. Loss: 0.090 |  Val. Acc: 97.46%
Epoch: 16
        Train Loss: 0.024 | Train Acc: 99.14%
         Val. Loss: 0.090 |  Val. Acc: 97.50%
Epoch: 17
        Train Loss: 0.024 | Train Acc: 99.14%
         Val. Loss: 0.090 |  Val. Acc: 97.50%
Epoch: 18
        Train Loss: 0.023 | Train Acc: 99.15%
         Val. Loss: 0.090 |  Val. Acc: 97.52%
Epoch: 19
        Train Loss: 0.023 | Train Acc: 99.18%
         Val. Loss: 0.091 |  Val. Acc: 97.52%
```

```
Epoch: 20
        Train Loss: 0.023 | Train Acc: 99.19%
         Val. Loss: 0.091 |  Val. Acc: 97.47%
Epoch: 21
        Train Loss: 0.022 | Train Acc: 99.21%
         Val. Loss: 0.091 |  Val. Acc: 97.53%
Epoch: 22
        Train Loss: 0.021 | Train Acc: 99.24%
         Val. Loss: 0.090 |  Val. Acc: 97.56%
Epoch: 23
        Train Loss: 0.021 | Train Acc: 99.23%
         Val. Loss: 0.090 |  Val. Acc: 97.58%
Epoch: 24
        Train Loss: 0.021 | Train Acc: 99.26%
         Val. Loss: 0.089 |  Val. Acc: 97.59%
Epoch: 25
        Train Loss: 0.021 | Train Acc: 99.24%
         Val. Loss: 0.090 |  Val. Acc: 97.58%
Epoch: 26
        Train Loss: 0.021 | Train Acc: 99.25%
         Val. Loss: 0.089 |  Val. Acc: 97.59%
Epoch: 27
        Train Loss: 0.022 | Train Acc: 99.23%
         Val. Loss: 0.089 |  Val. Acc: 97.59%
Epoch: 28
        Train Loss: 0.022 | Train Acc: 99.24%
         Val. Loss: 0.089 |  Val. Acc: 97.58%
Epoch: 29
        Train Loss: 0.022 | Train Acc: 99.21%
         Val. Loss: 0.089 |  Val. Acc: 97.58%
Epoch: 30
        Train Loss: 0.021 | Train Acc: 99.27%
         Val. Loss: 0.089 |  Val. Acc: 97.59%
Epoch: 31
        Train Loss: 0.021 | Train Acc: 99.24%
         Val. Loss: 0.089 |  Val. Acc: 97.60%
Epoch: 32
        Train Loss: 0.021 | Train Acc: 99.24%
         Val. Loss: 0.089 |  Val. Acc: 97.59%
Epoch: 33
        Train Loss: 0.021 | Train Acc: 99.25%
         Val. Loss: 0.089 |  Val. Acc: 97.59%
Epoch: 34
        Train Loss: 0.021 | Train Acc: 99.25%
         Val. Loss: 0.089 |  Val. Acc: 97.59%
Epoch: 35
        Train Loss: 0.021 | Train Acc: 99.23%
         Val. Loss: 0.089 |  Val. Acc: 97.60%
```

In [18]:
```
def categoricalEvaluate(preds, text, predict_table):

    max_preds = preds.argmax(dim = 1, keepdim = True)
    for predict, word in zip(max_preds, text):
        if word == 0:
            continue
        else:
            predict_table.append((word, predict[0]))

    return predict_table

def model_evaluate(model, dataloader, predict_table):

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

    with torch.no_grad():
        for text in dataloader:
            text = text.to(device)
            predictions = model(text)
            predictions = predictions.view(-1, predictions.shape[-1])
            predict_table = categoricalEvaluate(predictions, text.view(-1),

    return predict_table
```

In [20]:
```
term = [int(x[0]) for x in best_predict_table]
y_pred = [int(x[1]) for x in best_predict_table]
i=0
newfile = open('./dev2.out', "w")
with open('/content/drive/MyDrive/Colab Notebooks/data/dev', "r") as train:
    for line in train:
        if not line.split():
            newfile.write('\n')
            continue
        index, word_type = line.split(" ")[0], line.split(" ")[1].strip('\n
        newfile.write(str(index)+' '+str(word_type)+' '+str(Index_to_Ner[y_
        i += 1
newfile.close()

i=0
newfile = open('./dev2_perl.out', "w")
with open('/content/drive/MyDrive/Colab Notebooks/data/dev', "r") as train:
    for line in train:
        if not line.split():
            newfile.write('\n')
            continue
        index, word_type, NER_type = line.split(" ")[0], line.split(" ")[1]
        newfile.write(str(index)+' '+str(word_type)+' '+str(NER_type)+' '+s
        i += 1
newfile.close()
```

In [24]:
```
!perl conll03eval.txt < dev2_perl.out
```

```
processed 51578 tokens with 5942 phrases; found: 5687 phrases; correct: 5
061.
accuracy:  97.59%; precision:  88.99%; recall:  85.17%; FB1:  87.04
              LOC: precision:  91.37%; recall:  91.67%; FB1:  91.52  1843
             MISC: precision:  83.71%; recall:  81.34%; FB1:  82.51  896
              ORG: precision:  83.40%; recall:  79.79%; FB1:  81.55  1283
              PER: precision:  93.51%; recall:  84.53%; FB1:  88.79  1665
```

In [25]:
```
print("**** ", len(Word_to_Index), len(NER_to_Index))
```

```
****  11994 9
```

In [26]:
```
test_X = []
sentence = []
cnt=0
with open('/content/drive/MyDrive/Colab Notebooks/data/test', "r") as test:
    for line in test:
        if not line.split():
            test_X.append(sentence)
            sentence = []
            continue
        word_type = line.split(" ")[1]
        if word_type in Word_to_Index:
            sentence.append(Word_to_Index.get(word_type))
        else:
            unk = processUnknowns(word_type)
            sentence.append(Word_to_Index.get(unk))
    test_X.append(sentence)

test_X = padding_for_words(test_X, 130)
X_test = torch.LongTensor(test_X)
loader_test = DataLoader(X_test, batch_size=16, shuffle=False)

evaluate_predict_table2 = []
model = BLSTM(INPUT_DIM,
              EMBEDDING_DIM,
              HIDDEN_DIM,
              FIRST_OUTPUT_DIM,
              OUTPUT_DIM,
              N_LAYERS,
              BIDIRECTIONAL,
              DROPOUT)
model.to(device)
model.embedding.weight.data.copy_(embedding_matrix)
model.load_state_dict(torch.load('./blstm2.pt'))
prediction_table = model_evaluate(model, loader_test, evaluate_predict_tabl

term = [int(x[0]) for x in evaluate_predict_table2]
y_pred = [int(x[1]) for x in evaluate_predict_table2]

i=0
newfile = open('./test2.out', "w")
with open('/content/drive/MyDrive/Colab Notebooks/data/test', "r") as test:
    for line in test:
        if not line.split():
            newfile.write('\n')
            continue
        index, word_type = line.split(" ")[0], line.split(" ")[1].strip('\n
        for_tag = Index_to_Ner[y_pred[i]]
        newfile.write(str(index)+' '+str(word_type)+' '+for_tag+'\n')
        i += 1
newfile.close()
```

In [37]:
```
print(type(loader_test))
```

```
<class 'torch.utils.data.dataloader.DataLoader'>
```

In [28]:
```
import pickle
# save data
with open('./vocab_dictionary1.pickle','wb') as fw1:
    pickle.dump(Word_to_Index, fw1)
with open('./ner_dictionary1.pickle','wb') as fw2:
    pickle.dump(NER_to_Index, fw2)
with open('./int_vocab_dictionary1.pickle','wb') as fw3:
    pickle.dump(Index_to_Word, fw3)
with open('./int_ner_dictionary1.pickle','wb') as fw4:
    pickle.dump(Index_to_Ner, fw4)
with open('./loader_train1.pickle','wb') as fw5:
    pickle.dump(loader_train, fw5)
with open('./loader_dev1.pickle','wb') as fw6:
    pickle.dump(loader_dev, fw6)
with open('./loader_test1.pickle','wb') as fw7:
    pickle.dump(loader_test, fw7)
with open('./embedding_matrix.pickle','wb') as fw8:
    pickle.dump(embedding_matrix, fw8)
```

In [32]:
```
checkpoint = {'INPUT_DIM':len(Word_to_Index),
              'EMBEDDING_DIM':100,
              'HIDDEN_DIM':256,
              'FIRST_OUTPUT_DIM':128,
              'OUTPUT_DIM':len(NER_to_Index),
              'N_LAYERS':1,
              'BIDIRECTIONAL':True,
              'DROPOUT':0.33,
              'state_dict': model.state_dict()}

torch.save(checkpoint, './checkpoint.pth')
```

In [35]:
```
checkpoint = torch.load('./checkpoint.pth')
```

In [34]:
```
with open('./best_predict_table.pickle','wb') as fw9:
    pickle.dump(best_predict_table, fw9)
```

In [ ]:
```

```