HW4_Report

Task1(results evaluated using comll03eval.txt:

- The precision on the dev data is 81.21%.
  The recall on the dev data is 76.64%.
  The F1-score on the dev data is 78.86.

```
/content/ython3 hw4_yuhengchen.py data/train perl conll03eval.txt < self_dev1.out
processed 51578 tokens with 5942 phrases; found: 5608 phrases; correct: 4554.
accuracy:  95.59%; precision:  81.21%; recall:  76.64%; FB1:  78.86
              LOC: precision:  85.91%; recall:  84.00%; FB1:  84.94  1796
             MISC: precision:  83.21%; recall:  75.27%; FB1:  79.04  834
              ORG: precision:  73.17%; recall:  67.11%; FB1:  70.01  1230
              PER: precision:  81.06%; recall:  76.93%; FB1:  78.94  1748
```

- The hyperparameters are `embedding_dim = 100, hidden_dim = 256, num_layers = 1, dropout = 0.33, output_dim = 128, lr = 0.1, batch_size = 20, epochs = 140`
- I set a threshold to place the unknown words in the <unk> token for vocabulary and create a word to index as dict_vocab(use the index of word for word embedding). Then I create dataloader to train the BiLSTM model.I used nn.Embedding to create embedding matrix. I used permute in forward function to change the shape of outputs of the model.

Task2(results evaluated using comll03eval.txt:

- The precision on the dev data is 87.05%.
  The recall on the dev data is 88.72%.
  The F1-score on the dev data is 87.88.

```
/content/gdrive/MyDrive/Colab_Notebooks/HW4# perl conll03eval.txt < self_dev2.out
processed 51578 tokens with 5942 phrases; found: 6056 phrases; correct: 5272.
accuracy:  97.49%; precision:  87.05%; recall:  88.72%; FB1:  87.88
              LOC: precision:  91.38%; recall:  92.87%; FB1:  92.12  1867
             MISC: precision:  78.48%; recall:  80.69%; FB1:  79.57  948
              ORG: precision:  80.21%; recall:  79.49%; FB1:  79.85  1329
              PER: precision:  91.84%; recall:  95.33%; FB1:  93.55  1912
```

- The hyperparameters are embedding_dim = 101, hidden_dim = 256, num_layers = 1, dropout = 0.33, output_dim = 128, lr = 0.15, batch_size = 25, epochs = 140
- I used glove word embedding. Instead of word indexes, I used word vectors. In order to deal with the uppercase and lowercase words, I first detected if this word is in lowercase or not. If the word has lowercase character, it is lowercase, I lowered the word to find the vector and append 1 to the array of the vector. Otherwise, I lowered the word to find the vector and append 0 to the array of the vector. For the unknown words, I used the zero vector with size of 101. I created dataloader to train the model.
- For the model, I didn't use nn.Embedding layer because I found it has higher accuracy.

HW4_YuhengChen

March 25, 2023

[1]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[2]:
```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
import numpy as np
import gzip
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
from torch.utils.data import TensorDataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
```

[115]:
```
input_f = "gdrive/MyDrive/Colab Notebooks/HW4/data/train"
input_v = "gdrive/MyDrive/Colab Notebooks/HW4/data/dev"
input_t = "gdrive/MyDrive/Colab Notebooks/HW4/data/test"
f = open(input_f, "r")
train_text = [line.strip() for line in f.readlines()]

f2 = open(input_v, "r")
val_text = [line.strip() for line in f2.readlines()]
```

[116]:
```
f3 = open(input_t, "r")
test_text = [line.strip() for line in f3.readlines()]
```

[117]:
```
len(test_text)
```

[117]:
```
50349
```

[118]:
```
split = [item.split(' ') for item in train_text]
## create a dictionary to count word occurences
dict1 = {}
```

```
for item in split:
    if len(item) > 1:
        key = item[1]
        if key not in dict1:
            dict1[key] = 1
        else:
            dict1[key] += 1
# print(dict1['-DOCSTART-'])
## sort dictionary in descending order
dict_sorted = {k: v for k, v in sorted(dict1.items(), key=lambda item: item[1],␣
 ↪reverse = True)}
## replace rare words whose occurence less than 3 with a special token '< unk␣
 ↪>'.
rare_words_occ = 0
rare_words = []
for k,v in dict_sorted.items():
    if v < 2:
        rare_words_occ += v
        rare_words.append([k,v])

## delete rare words and add unkown token to vocabulary and update vocabulary
for item in rare_words:
      dict_sorted.pop(item[0])
updict = {'<unk>': rare_words_occ}
dict_vocab = {**updict, **dict_sorted}

## create word2idx
i = 1
for key, value in dict_vocab.items():
  dict_vocab[key] = i
  i += 1
```

[ ]:

0.1 Task 1: Simple Bidirectional LSTM model

[119]:
```
def read_data(file_path):
    sentences = []
    tags = []
    with open(file_path, 'r') as f:
        sentence = []
        tag = []

        for line in f:
            if line.strip():
                # word, _, ner_tag = line.strip().split()
                index, word, ner_tag = line.strip().split()
```

```
                # word = word.lower()
                sentence.append(word)
                tag.append(ner_tag)
            else:
                sentences.append(sentence)
                tags.append(tag)
                sentence = []
                tag = []
        if sentence:
            sentences.append(sentence)
            tags.append(tag)
    return sentences, tags

def create_tag2id(tags):
    tag2id = {}
    for tag_list in tags:
        for tag in tag_list:
            if tag not in tag2id:
                tag2id[tag] = len(tag2id)
    return tag2id

emb_dim = 100
hidden_dim = 256
drop_out = 0.33
output_dim = 128
# vocab_size = len(word2id)
batch_size = 1
sentences, tags_all = read_data(input_f)
tags_idx = create_tag2id(tags_all)
# ## create dataset for training
# sentences, tags = read_data(input_f)
# word2id = create_vocab(sentences)
# vocab_size = len(word2id)
tag2id = create_tag2id(tags_all)
```

[9]:
```
tag2id
```

[9]:
```
{'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}
```

[120]:
```
sentences_dev, tags_dev = read_data(input_v)
```

[121]:
```
def read_test_data(file_path):
    sentences = []
    with open(file_path, 'r') as f:
        sentence = []

        for line in f:
            if line.strip():
                # word, _, ner_tag = line.strip().split()
                index, word = line.strip().split()
                # word = word.lower()
                sentence.append(word)

            else:
                sentences.append(sentence)

                sentence = []

        if sentence:
            sentences.append(sentence)

    return sentences
```

[122]:
```
test_sentence = read_test_data(input_t)
```

[123]:
```
def create_test_dataset(sen):
  text = []


  for sentence in sen:
    sentence_text = []
    for item in sentence:
      if item in dict_vocab:
        embedding = torch.tensor(dict_vocab[item])
      else:
        embedding = torch.tensor(dict_vocab['<unk>'])
      sentence_text.append(embedding)
    text.append(sentence_text)


  # print(text)
  text = [torch.tensor(i) for i in text]
  text = pad_sequence(text, batch_first = True, padding_value= 0)


  # create a TensorDataset
```

```
  dataset = TensorDataset(text)
  # create the train_dataloader
  batch_size = 20
  dataloader = DataLoader(dataset, batch_size=20, shuffle=True)
  # print(tags)
  return dataloader, text
```

[124]:
```
def create_dataset(sen, tag):
  text = []
  text_len = []
  tags = []

  for sentence in sen:
    sentence_text = []
    for item in sentence:
      if item in dict_vocab:
        embedding = torch.tensor(dict_vocab[item])
      else:
        embedding = torch.tensor(dict_vocab['<unk>'])
      sentence_text.append(embedding)
    text.append(sentence_text)
    text_len.append(len(sentence_text))

  # print(len(tag))
  for taglst in tag:
    # print(taglst)
    sentence_tag = []
    for item in taglst:
      # print(item)
      sentence_tag.append(torch.tensor(tag2id[item]))
    tags.append(sentence_tag)
  # print(text)
  text = [torch.tensor(i) for i in text]
  text = pad_sequence(text, batch_first = True, padding_value= 0)
  text_len = torch.tensor(text_len)
  tags = [torch.tensor(i) for i in tags]
  tags = pad_sequence(tags, batch_first = True, padding_value= 100)

  print(text.size(), tags.size())
  # print(text)



  # create a TensorDataset
  dataset = TensorDataset(text, tags)
  # create the train_dataloader
  batch_size = 20
```

```
  dataloader = DataLoader(dataset, batch_size=20, shuffle=True)
  # print(tags)
  return dataloader, text, tags
```

[125]:
```
train_dataloader, train_text, train_tags = create_dataset(sentences, tags_all)
dev_dataloader, dev_text, dev_tags = create_dataset(sentences_dev, tags_dev)
test_dataloader, test_text = create_test_dataset(test_sentence)
train_dataloader.batch_size
```

```
torch.Size([14987, 113]) torch.Size([14987, 113])
torch.Size([3466, 109]) torch.Size([3466, 109])
```

[125]: 20

[10]:
```
# text,tags = create_dataset(sentences, tags_all)
# dev_text, dev_tags = create_dataset(sentences_dev, tags_dev)
# train_d = Train(text, tags)
# dev_d = Train(dev_text, dev_tags)
```

[11]:
```
# len(text[0])
```

[126]:
```
from pandas.core import accessor
import torch
import torch.nn as nn
class NERModel(nn.Module):
    def __init__(self, num_embeddings, embedding_dim, hidden_dim, output_dim,␣
 ↪dropout, num_layers):
        super().__init__()
        self.embedding = nn.Embedding(num_embeddings, embedding_dim)
        self.bilstm = nn.LSTM(embedding_dim, hidden_dim, num_layers,
                              bidirectional=True, batch_first=True)
        self.dropout = nn.Dropout(dropout)
        # self.linear = nn.Linear(hidden_dim * 2, output_dim)
        # self.elu = nn.ELU()
        # self.classifier = nn.Linear(output_dim, 9)
        self.linear = nn.Linear(hidden_dim*2, hidden_dim)
        self.activation = nn.ELU()
        self.classifier = nn.Linear(hidden_dim, output_dim)
        self.cl2 = nn.Linear(output_dim,9)



    def forward(self, x):
        out = self.embedding(x)
        out, _ = self.bilstm(out)
        out = self.dropout(out)
        out = self.linear(out)
        out = self.activation(out)
```

```
        out = self.classifier(out)
        out = self.cl2(out)
        out = out.permute(0,2,1)


        return out




    # def forward(self, x):
    #     embedded = self.embedding(x)
    #     lstm_output, _ = self.bilstm(embedded)
    #     linear_output = self.linear(lstm_output)
    #     elu_output = self.elu(linear_output)
    #     output = self.classifier(elu_output)


        # embedded = self.embedding(x)
        # lstm_out, (hidden, cell) = self.bilstm(embedded)
        # hidden = torch.cat((hidden[-2, :, :], hidden[-1, :, :]), dim=1)
        # linear_out = self.linear(hidden)
        # activation_out = self.elu(linear_out)
        # output = self.classifier(activation_out)
        # return output
```

[21]:
```
# Instantiate the model
# Define the loss function and optimizer
from sklearn.metrics import f1_score
embedding_dim = 100
hidden_dim = 256

num_layers = 1
dropout = 0.33
output_dim = 128
lr = 0.1
batch_size = 20
epochs = 140
# CUDA_LAUNCH_BLOCKING = "1"
# torch.backends.cudnn.deterministic = True
# Specify the device (GPU if available, otherwise CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = NERModel(num_embeddings=14987, embedding_dim=embedding_dim,
                 hidden_dim=hidden_dim, output_dim=output_dim, dropout =␣
 ↪dropout,
                 num_layers = num_layers)
```

```
model = model.to(device)
loss_function = nn.CrossEntropyLoss(ignore_index = 100)
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

# Train the model
for epoch in range(epochs):
    epoch_loss = 0.0
    for batch_inputs, batch_labels in train_dataloader:
        # Send the batch to the device
        batch_inputs, batch_labels = batch_inputs.to(device), batch_labels.
↪to(device)
        optimizer.zero_grad()
        outputs = model(batch_inputs)
        loss = loss_function(outputs, batch_labels)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
        pred = torch.argmax(outputs,axis=1)


        # # print(batch_inputs.size(), batch_labels.size())
        # # print(batch_labels.tolist())
        # # print(batch_inputs.tolist())
        # optimizer.zero_grad()
        # batch_outputs = model(batch_inputs)

        # print(batch_outputs.size())

        # tansposed_output = torch.transpose(batch_outputs, 1, 2)
        # # print(tansposed_output.size())
        # # target = torch.randint(number_of_classes,␣
↪(batch_size,sequence_length)).long()
        # # loss = loss_function(batch_outputs, batch_labels)


        # # batch_outputs = batch_outputs.view(-1, 9)
        # # batch_labels = batch_labels.view(-1)
        # # loss = loss_function(batch_outputs, torch.max(batch_labels,1)[1])
        # loss = loss_function(tansposed_output, batch_labels)
        # # print(loss)
        # epoch_loss += loss.item()
        # loss.backward()
        # optimizer.step()
        # # batch_preds = torch.argmax(tansposed_output, axis=1)
        # # batch_f1 = f1_score(batch_labels.cpu(), batch_preds.cpu(),␣
↪average='micro')
        # # epoch_f1 += batch_f1
```

```
    epoch_loss /= len(train_dataloader)
    # epoch_f1 /= len(train_dataloader)
    print(f"Epoch {epoch+1}: Loss={epoch_loss:.4f}")
    # , F1 Score={epoch_f1:.4f}"
```

```
Epoch 1: Loss=0.7208
Epoch 2: Loss=0.6218
Epoch 3: Loss=0.5454
Epoch 4: Loss=0.4882
Epoch 5: Loss=0.4415
Epoch 6: Loss=0.4035
Epoch 7: Loss=0.3699
Epoch 8: Loss=0.3374
Epoch 9: Loss=0.3095
Epoch 10: Loss=0.2841
Epoch 11: Loss=0.2617
Epoch 12: Loss=0.2424
Epoch 13: Loss=0.2247
Epoch 14: Loss=0.2095
Epoch 15: Loss=0.1954
Epoch 16: Loss=0.1833
Epoch 17: Loss=0.1720
Epoch 18: Loss=0.1592
Epoch 19: Loss=0.1522
Epoch 20: Loss=0.1425
Epoch 21: Loss=0.1349
Epoch 22: Loss=0.1273
Epoch 23: Loss=0.1223
Epoch 24: Loss=0.1160
Epoch 25: Loss=0.1094
Epoch 26: Loss=0.1037
Epoch 27: Loss=0.0986
Epoch 28: Loss=0.0936
Epoch 29: Loss=0.0900
Epoch 30: Loss=0.0862
Epoch 31: Loss=0.0807
Epoch 32: Loss=0.0788
Epoch 33: Loss=0.0748
Epoch 34: Loss=0.0703
Epoch 35: Loss=0.0678
Epoch 36: Loss=0.0643
Epoch 37: Loss=0.0616
Epoch 38: Loss=0.0597
Epoch 39: Loss=0.0584
Epoch 40: Loss=0.0554
Epoch 41: Loss=0.0537
Epoch 42: Loss=0.0515
Epoch 43: Loss=0.0491
```

```
Epoch 44: Loss=0.0476
Epoch 45: Loss=0.0453
Epoch 46: Loss=0.0435
Epoch 47: Loss=0.0426
Epoch 48: Loss=0.0411
Epoch 49: Loss=0.0409
Epoch 50: Loss=0.0386
Epoch 51: Loss=0.0366
Epoch 52: Loss=0.0358
Epoch 53: Loss=0.0346
Epoch 54: Loss=0.0337
Epoch 55: Loss=0.0319
Epoch 56: Loss=0.0311
Epoch 57: Loss=0.0293
Epoch 58: Loss=0.0297
Epoch 59: Loss=0.0291
Epoch 60: Loss=0.0277
Epoch 61: Loss=0.0273
Epoch 62: Loss=0.0258
Epoch 63: Loss=0.0256
Epoch 64: Loss=0.0244
Epoch 65: Loss=0.0241
Epoch 66: Loss=0.0231
Epoch 67: Loss=0.0223
Epoch 68: Loss=0.0215
Epoch 69: Loss=0.0205
Epoch 70: Loss=0.0210
Epoch 71: Loss=0.0204
Epoch 72: Loss=0.0197
Epoch 73: Loss=0.0197
Epoch 74: Loss=0.0193
Epoch 75: Loss=0.0177
Epoch 76: Loss=0.0181
Epoch 77: Loss=0.0172
Epoch 78: Loss=0.0171
Epoch 79: Loss=0.0169
Epoch 80: Loss=0.0159
Epoch 81: Loss=0.0155
Epoch 82: Loss=0.0156
Epoch 83: Loss=0.0149
Epoch 84: Loss=0.0164
Epoch 85: Loss=0.0148
Epoch 86: Loss=0.0147
Epoch 87: Loss=0.0141
Epoch 88: Loss=0.0137
Epoch 89: Loss=0.0141
Epoch 90: Loss=0.0130
Epoch 91: Loss=0.0132
```

```
Epoch 92: Loss=0.0135
Epoch 93: Loss=0.0121
Epoch 94: Loss=0.0121
Epoch 95: Loss=0.0122
Epoch 96: Loss=0.0116
Epoch 97: Loss=0.0114
Epoch 98: Loss=0.0115
Epoch 99: Loss=0.0111
Epoch 100: Loss=0.0114
Epoch 101: Loss=0.0108
Epoch 102: Loss=0.0109
Epoch 103: Loss=0.0107
Epoch 104: Loss=0.0107
Epoch 105: Loss=0.0106
Epoch 106: Loss=0.0107
Epoch 107: Loss=0.0107
Epoch 108: Loss=0.0095
Epoch 109: Loss=0.0100
Epoch 110: Loss=0.0092
Epoch 111: Loss=0.0094
Epoch 112: Loss=0.0089
Epoch 113: Loss=0.0093
Epoch 114: Loss=0.0095
Epoch 115: Loss=0.0091
Epoch 116: Loss=0.0090
Epoch 117: Loss=0.0084
Epoch 118: Loss=0.0090
Epoch 119: Loss=0.0084
Epoch 120: Loss=0.0081
Epoch 121: Loss=0.0078
Epoch 122: Loss=0.0080
Epoch 123: Loss=0.0080
Epoch 124: Loss=0.0083
Epoch 125: Loss=0.0084
Epoch 126: Loss=0.0081
Epoch 127: Loss=0.0079
Epoch 128: Loss=0.0079
Epoch 129: Loss=0.0077
Epoch 130: Loss=0.0068
Epoch 131: Loss=0.0075
Epoch 132: Loss=0.0076
Epoch 133: Loss=0.0068
Epoch 134: Loss=0.0069
Epoch 135: Loss=0.0068
Epoch 136: Loss=0.0070
Epoch 137: Loss=0.0068
Epoch 138: Loss=0.0067
Epoch 139: Loss=0.0072
```

```
Epoch 140: Loss=0.0062
```

[22]:
```
## save model
save_path_blstm1 = "gdrive/MyDrive/Colab Notebooks/HW4/blstm1.pt"
torch.save(model, save_path_blstm1)

##load model
# blstm1_model = torch.load(save_path_blstm1)
```

[23]:
```
save_path_blstm1 = "gdrive/MyDrive/Colab Notebooks/HW4/blstm1.pt"
blstm1 = torch.load(save_path_blstm1)
```

[27]:
```
# y_pred = []
# y_true = []
# device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# blstm1.eval()
# with torch.no_grad():
#     for batch_inputs, batch_labels in dev_dataloader:
#         batch_inputs = batch_inputs.to(device)
#         batch_labels = batch_labels.to(device)

#         output = blstm1(batch_inputs)

#         y_pred.extend(output.tolist())
#         y_true.extend(batch_labels.tolist())
```

[127]:
```
## predictions for test data

y_pred_test = []
pop = []
final_predictions = []
ignore_idx = 100
# y_true = []
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
blstm1.eval()
test_dataset = TensorDataset(test_text)
test_d = DataLoader(test_dataset,shuffle=False)

with torch.no_grad():
    for batch_inputs in test_d:
        batch_inputs = batch_inputs[0].to(device)

        output = blstm1(batch_inputs)

        y_pred_test.append(torch.argmax(output, dim=1))

for i in test_text:
    pop.append(i[i!=ignore_idx].tolist())
```

```
# print(len(y_pred_test), len(pop))

for i in range(len(pop)):

    final_predictions+=y_pred_test[i][0][:len(pop[i])].tolist()
```

[128]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
def predictions(dev_x, dev_y, ignore_idx, blstm1):
  true_tag = []
  position = []
  predictions = []
  final_pred = []
  for i in dev_y:
    true_tag+=i[i!=ignore_idx].tolist()
    position.append(i[i!=ignore_idx].tolist())

  dev_dataset = TensorDataset(dev_x)
  dev_d = DataLoader(dev_dataset,shuffle=False)

  blstm1.eval()
  with torch.no_grad():
    for inputs in dev_d:
      i = inputs[0].to(device)

      outputs = blstm1(i)
      predictions.append(torch.argmax(outputs, dim=1))
  # print(len(predictions), len(position))
  for i in range(len(position)):

    final_pred+=predictions[i][0][:len(position[i])].tolist()
  return true_tag, final_pred
```

[129]:
```
from sklearn.metrics import f1_score
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
true_tag, pred = predictions(dev_text, dev_tags, 100, blstm1)
f1_score(true_tag, pred, average='macro')
```

[129]:
```
0.8303096813899516
```

[130]:
```
id2tag = {y: x for x, y in tag2id.items()}
id2tag
```

[130]:
```
{0: 'B-ORG',
 1: 'O',
 2: 'B-MISC',
 3: 'B-PER',
 4: 'I-PER',
```

```
 5: 'B-LOC',
 6: 'I-ORG',
 7: 'I-MISC',
 8: 'I-LOC'}
```

[105]:
```
a = ['1 2 3', 'e f d']
for i in range(len(a)):
  split_str = a[i].split(' ')
  # print(split_str)
  split_str[2] = 'opp'
  a[i] = ' '.join(split_str)
  # print(split_str)
a
```

```
['1', '2', '3']
['1', '2', 'opp']
['e', 'f', 'd']
['e', 'f', 'opp']
```

[105]:
```
['1 2 opp', 'e f opp']
```

[132]:
```
## add predictions to dev
f2 = open(input_v, "r")
val_text = [line.strip() for line in f2.readlines()]

j = 0
for i in range(len(val_text)):
  if val_text[i] == '':
    continue
  else:
    split_str = val_text[i].split(' ')
    # print(pred[j])
    split_str[2] = id2tag[pred[j]]
    val_text[i] = ' '.join(split_str)
    j += 1
    # item += '3'

f3 = open(input_t, "r")
test_text = [line.strip() for line in f3.readlines()]
## add predictions to test
j = 0
for i in range(len(test_text)):
  if test_text[i] == '':
    continue
  else:
    test_text[i] += ' '
    test_text[i] += id2tag[final_predictions[j]]
    j += 1
```

[134]:
```
## self-predicted file
f2 = open(input_v, "r")
val_text_self = [line.strip() for line in f2.readlines()]

j = 0
for i in range(len(val_text_self)):
  if val_text_self[i] == '':
    continue
  else:
    val_text_self[i] += ' '
    # print(pred[j])
    val_text_self[i] += id2tag[pred[j]]
    j += 1
    # item += '3'
```

[136]:
```
file_test_pred = open('dev1.out','w')
for item in val_text:
        file_test_pred.write(item+"\n")
file_test_pred.close()
```

[137]:
```
file_test_pred_self = open('self_dev1.out','w')
for item in val_text_self:
        file_test_pred_self.write(item+"\n")
file_test_pred_self.close()
```

[138]:
```
file_test_pred_test = open('test1.out','w')
for item in test_text:
        file_test_pred_test.write(item+"\n")
file_test_pred_test.close()
```

0.2 Task 2: Using GloVe word embeddings

[139]:
```
## building GLOVE word vocab
word_vectors = {}

with gzip.open('/content/gdrive/MyDrive/Colab Notebooks/HW4/glove.6B.100d.gz',
               'rt', encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]

        # idx_to_word[len(idx_to_word)] = word
        vec = np.asarray(values[1:], dtype='float32')
        word_vectors[word] = vec
```

[140]:
```
## function to detect lower case charater in a string
def detectlower(string):
```

```
  for char in string:
    if char.islower():
      return True
  return False
```

[67]:
```
sentences[0]
```

[67]:
```
['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.']
```

[141]:
```
def create_w2v(sentences, tagslist):
  text = []
  tags = []


  for sentence in sentences:
    sentence_text = []
    for item in sentence:
      if item.lower() in word_vectors:
        if detectlower(item):
          # value = word_vectors[item].append(1)
          item = item.lower()
          value = np.append(word_vectors[item], 1)
        else:
          item = item.lower()
          value = np.append(word_vectors[item], 0)
      else:
        value = np.zeros(((100,)))
        # value = np.random.normal(size=(100,))
        value = np.append(value, 1)
      sentence_text.append(value)
    text.append(sentence_text)

  for taglst in tagslist:
    sentence_tag = []
    for item in taglst:
      sentence_tag.append(torch.tensor(tag2id[item]))
    tags.append(sentence_tag)

# print(text)
  text = [torch.tensor(i) for i in text]
  text = pad_sequence(text, batch_first = True, padding_value= 0)
  tags = [torch.tensor(i) for i in tags]
  tags = pad_sequence(tags, batch_first = True, padding_value= 100)
  return text, tags
```

[142]:
```
def create_w2v_test(sentences):
  text = []
```

```
  for sentence in sentences:
    sentence_text = []
    for item in sentence:
      if item.lower() in word_vectors:
        if detectlower(item):
          # value = word_vectors[item].append(1)
          item = item.lower()
          value = np.append(word_vectors[item], 1)
        else:
          item = item.lower()
          value = np.append(word_vectors[item], 0)
      else:
        value = np.zeros(((100,)))
        # value = np.random.normal(size=(100,))
        value = np.append(value, 1)
      sentence_text.append(value)
    text.append(sentence_text)


# print(text)
  text = [torch.tensor(i) for i in text]
  text = pad_sequence(text, batch_first = True, padding_value= 0)

  return text
```

[143]:
```
train_text, train_tags = create_w2v(sentences, tags_all)
dev_text, dev_tags = create_w2v(sentences_dev, tags_dev)
test_text = create_w2v_test(test_sentence)
```

[15]:
```
train_text.size(), train_tags.size()
```

[15]:
```
(torch.Size([14987, 113, 101]), torch.Size([14987, 113]))
```

[72]:
```
test_text.size(), dev_tags.size()
```

[72]:
```
(torch.Size([3684, 124, 101]), torch.Size([3466, 109]))
```

[144]:
```
# create a TensorDataset
train_dataset = TensorDataset(train_text, train_tags)
dev_dataset = TensorDataset(dev_text, dev_tags)
# create the train_dataloader
train_dataloader2 = DataLoader(train_dataset, batch_size=20, shuffle=True)
dev_dataloader2 = DataLoader(dev_dataset, batch_size=20, shuffle=True)
```

[234]:
```
len(dict_vocab)
```

[234]: 11984

[145]:
```
class GloveModel(nn.Module):
    def __init__(self, num_embeddings, embedding_dim, hidden_dim, output_dim,␣
 ↪dropout, num_layers):

        super().__init__()
        # self.embedding = nn.Embedding(num_embeddings, embedding_dim)
        self.bilstm = nn.LSTM(embedding_dim, hidden_dim, num_layers,
                              bidirectional=True, batch_first=True)
        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(hidden_dim * 2, hidden_dim)
        self.elu = nn.ELU()
        # self.classifier = nn.Linear(hidden_dim, output_dim)
        self.classifier = nn.Linear(hidden_dim, output_dim)
        # self.cl2 = nn.Linear(output_dim,9)
        # self.linear = nn.Linear(hidden_dim*2, hidden_dim)
        # self.activation = nn.ELU()
        # self.classifier = nn.Linear(hidden_dim, output_dim)
        self.cl2 = nn.Linear(output_dim, 9)




    def forward(self, x):
        # out = self.embedding(x)
        out, _ = self.bilstm(x)
        out = self.dropout(out)
        out = self.linear(out)
        out = self.elu(out)
        out = self.classifier(out)
        out = self.cl2(out)
        out = out.permute(0,2,1)

        return out
```

[50]:
```
embedding_dim = 101
hidden_dim = 256

num_layers = 1
dropout = 0.33
output_dim = 128
lr = 0.15
batch_size = 25
epochs = 140
# CUDA_LAUNCH_BLOCKING = "1"
# torch.backends.cudnn.deterministic = True
```

```
# Specify the device (GPU if available, otherwise CPU)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = GloveModel(num_embeddings=14987, embedding_dim=embedding_dim,
                   hidden_dim=hidden_dim, output_dim=output_dim, dropout =␣
 ↪dropout,
                   num_layers = num_layers)


model = model.to(device)
loss_function = nn.CrossEntropyLoss(ignore_index = 100)
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

# Train the model
for epoch in range(epochs):
    epoch_loss = 0.0
    for batch_inputs, batch_labels in train_dataloader2:
        # Send the batch to the device
        batch_inputs, batch_labels = batch_inputs.to(device), batch_labels.
 ↪to(device)
        # print(batch_inputs.size(), batch_labels.size())

        optimizer.zero_grad()
        outputs = model(batch_inputs.float())

        # torch.Size([20, 113, 101]) torch.Size([20, 113])
        # torch.Size([20, 128, 113])
        # print(outputs.size())
        loss = loss_function(outputs, batch_labels)

        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
        pred = torch.argmax(outputs,axis=1)


        # # print(batch_inputs.size(), batch_labels.size())
        # # print(batch_labels.tolist())
        # # print(batch_inputs.tolist())
        # optimizer.zero_grad()
        # batch_outputs = model(batch_inputs)

        # print(batch_outputs.size())

        # tansposed_output = torch.transpose(batch_outputs, 1, 2)
        # # print(tansposed_output.size())
```

```
        # # target = torch.randint(number_of_classes,␣
↪(batch_size,sequence_length)).long()
        # # loss = loss_function(batch_outputs, batch_labels)


        # # batch_outputs = batch_outputs.view(-1, 9)
        # # batch_labels = batch_labels.view(-1)
        # # loss = loss_function(batch_outputs, torch.max(batch_labels,1)[1])
        # loss = loss_function(tansposed_output, batch_labels)
        # # print(loss)
        # epoch_loss += loss.item()
        # loss.backward()
        # optimizer.step()
        # # batch_preds = torch.argmax(tansposed_output, axis=1)
        # # batch_f1 = f1_score(batch_labels.cpu(), batch_preds.cpu(),␣
↪average='micro')
        # # epoch_f1 += batch_f1
    epoch_loss /= len(train_dataloader2)
    print(f"Epoch {epoch+1}: Loss={epoch_loss:.4f}")
```

```
Epoch 1: Loss=0.5039
Epoch 2: Loss=0.3156
Epoch 3: Loss=0.2453
Epoch 4: Loss=0.2041
Epoch 5: Loss=0.1790
Epoch 6: Loss=0.1645
Epoch 7: Loss=0.1533
Epoch 8: Loss=0.1457
Epoch 9: Loss=0.1369
Epoch 10: Loss=0.1315
Epoch 11: Loss=0.1272
Epoch 12: Loss=0.1217
Epoch 13: Loss=0.1183
Epoch 14: Loss=0.1139
Epoch 15: Loss=0.1127
Epoch 16: Loss=0.1078
Epoch 17: Loss=0.1057
Epoch 18: Loss=0.1021
Epoch 19: Loss=0.0998
Epoch 20: Loss=0.0980
Epoch 21: Loss=0.0949
Epoch 22: Loss=0.0934
Epoch 23: Loss=0.0904
Epoch 24: Loss=0.0885
Epoch 25: Loss=0.0872
Epoch 26: Loss=0.0858
Epoch 27: Loss=0.0824
```

```
Epoch 28: Loss=0.0809
Epoch 29: Loss=0.0788
Epoch 30: Loss=0.0773
Epoch 31: Loss=0.0770
Epoch 32: Loss=0.0737
Epoch 33: Loss=0.0718
Epoch 34: Loss=0.0716
Epoch 35: Loss=0.0697
Epoch 36: Loss=0.0677
Epoch 37: Loss=0.0665
Epoch 38: Loss=0.0655
Epoch 39: Loss=0.0637
Epoch 40: Loss=0.0622
Epoch 41: Loss=0.0610
Epoch 42: Loss=0.0603
Epoch 43: Loss=0.0572
Epoch 44: Loss=0.0573
Epoch 45: Loss=0.0557
Epoch 46: Loss=0.0545
Epoch 47: Loss=0.0538
Epoch 48: Loss=0.0529
Epoch 49: Loss=0.0506
Epoch 50: Loss=0.0493
Epoch 51: Loss=0.0489
Epoch 52: Loss=0.0470
Epoch 53: Loss=0.0465
Epoch 54: Loss=0.0447
Epoch 55: Loss=0.0442
Epoch 56: Loss=0.0426
Epoch 57: Loss=0.0407
Epoch 58: Loss=0.0411
Epoch 59: Loss=0.0402
Epoch 60: Loss=0.0391
Epoch 61: Loss=0.0372
Epoch 62: Loss=0.0373
Epoch 63: Loss=0.0368
Epoch 64: Loss=0.0350
Epoch 65: Loss=0.0345
Epoch 66: Loss=0.0333
Epoch 67: Loss=0.0318
Epoch 68: Loss=0.0322
Epoch 69: Loss=0.0300
Epoch 70: Loss=0.0291
Epoch 71: Loss=0.0295
Epoch 72: Loss=0.0282
Epoch 73: Loss=0.0270
Epoch 74: Loss=0.0268
Epoch 75: Loss=0.0259
```

```
Epoch 76: Loss=0.0254
Epoch 77: Loss=0.0240
Epoch 78: Loss=0.0239
Epoch 79: Loss=0.0234
Epoch 80: Loss=0.0233
Epoch 81: Loss=0.0219
Epoch 82: Loss=0.0204
Epoch 83: Loss=0.0213
Epoch 84: Loss=0.0197
Epoch 85: Loss=0.0195
Epoch 86: Loss=0.0191
Epoch 87: Loss=0.0187
Epoch 88: Loss=0.0184
Epoch 89: Loss=0.0172
Epoch 90: Loss=0.0171
Epoch 91: Loss=0.0167
Epoch 92: Loss=0.0166
Epoch 93: Loss=0.0162
Epoch 94: Loss=0.0152
Epoch 95: Loss=0.0143
Epoch 96: Loss=0.0139
Epoch 97: Loss=0.0134
Epoch 98: Loss=0.0141
Epoch 99: Loss=0.0142
Epoch 100: Loss=0.0125
Epoch 101: Loss=0.0134
Epoch 102: Loss=0.0126
Epoch 103: Loss=0.0210
Epoch 104: Loss=0.0122
Epoch 105: Loss=0.0117
Epoch 106: Loss=0.0109
Epoch 107: Loss=0.0108
Epoch 108: Loss=0.0107
Epoch 109: Loss=0.0107
Epoch 110: Loss=0.0101
Epoch 111: Loss=0.0097
Epoch 112: Loss=0.0099
Epoch 113: Loss=0.0096
Epoch 114: Loss=0.0099
Epoch 115: Loss=0.0092
Epoch 116: Loss=0.0088
Epoch 117: Loss=0.0092
Epoch 118: Loss=0.0084
Epoch 119: Loss=0.0080
Epoch 120: Loss=0.0087
Epoch 121: Loss=0.0077
Epoch 122: Loss=0.0078
Epoch 123: Loss=0.0084
```

```
Epoch 124: Loss=0.0078
Epoch 125: Loss=0.0074
Epoch 126: Loss=0.0072
Epoch 127: Loss=0.0077
Epoch 128: Loss=0.0069
Epoch 129: Loss=0.0067
Epoch 130: Loss=0.0061
Epoch 131: Loss=0.0066
Epoch 132: Loss=0.0076
Epoch 133: Loss=0.0068
Epoch 134: Loss=0.0060
Epoch 135: Loss=0.0064
Epoch 136: Loss=0.0059
Epoch 137: Loss=0.0059
Epoch 138: Loss=0.0062
Epoch 139: Loss=0.0061
Epoch 140: Loss=0.0057
```

[54]:
```
## save model
save_path_blstm2 = "gdrive/MyDrive/Colab Notebooks/HW4/blstm2.pt"
torch.save(model, save_path_blstm2)

# #load model
# blstm2_model = torch.load(save_path_blstm2)
```

[146]:
```
#load model
save_path_blstm2 = "gdrive/MyDrive/Colab Notebooks/HW4/blstm2.pt"
blstm2_model = torch.load(save_path_blstm2)
```

[147]:
```
def predictions(dev_x, dev_y, ignore_idx, blstm1):
  true_tag = []
  position = []
  predictions = []
  final_pred = []
  for i in dev_y:
    true_tag+=i[i!=ignore_idx].tolist()
    position.append(i[i!=ignore_idx].tolist())

  dev_dataset = TensorDataset(dev_x)
  dev_d = DataLoader(dev_dataset,shuffle=False)

  blstm1.eval()
  with torch.no_grad():
    for inputs in dev_d:
      i = inputs[0].to(device).float()

      outputs = blstm1(i)
```

```
      predictions.append(torch.argmax(outputs, dim=1))

  for i in range(len(position)):

    final_pred+=predictions[i][0][:len(position[i])].tolist()
  return true_tag, final_pred
```

[148]:
```
## predictions for test data

y_pred_test = []
pop = []
final_predictions = []
ignore_idx = 100
# y_true = []
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
blstm2_model.eval()
test_dataset = TensorDataset(test_text)
test_d = DataLoader(test_dataset,shuffle=False)

with torch.no_grad():
    for batch_inputs in test_d:
        batch_inputs = batch_inputs[0].to(device).float()

        output = blstm2_model(batch_inputs)

        y_pred_test.append(torch.argmax(output, dim=1))

for i in test_text:
    pop.append(i[i!=ignore_idx].tolist())
# print(len(y_pred_test), len(pop))

for i in range(len(pop)):

    final_predictions+=y_pred_test[i][0][:len(pop[i])].tolist()
```

[149]:
```
from sklearn.metrics import f1_score
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
true_tag2, pred2 = predictions(dev_text, dev_tags, 100, blstm2_model)
f1_score(true_tag2, pred2, average='macro')
```

[149]:
```
0.8725888568896255
```

[150]:
```
f2 = open(input_v, "r")
val_text = [line.strip() for line in f2.readlines()]

j = 0
for i in range(len(val_text)):
```

```
  if val_text[i] == '':
    continue
  else:
    split_str = val_text[i].split(' ')
    split_str[2] = id2tag[pred2[j]]
    val_text[i] = ' '.join(split_str)
    j += 1
    # item += '3'

f3 = open(input_t, "r")
test_text = [line.strip() for line in f3.readlines()]
## add predictions to test
j = 0
for i in range(len(test_text)):
  if test_text[i] == '':
    continue
  else:
    test_text[i] += ' '
    test_text[i] += id2tag[final_predictions[j]]
    j += 1
```

[151]:
```
## self-predicted file
f2 = open(input_v, "r")
val_text_self = [line.strip() for line in f2.readlines()]

j = 0
for i in range(len(val_text_self)):
  if val_text_self[i] == '':
    continue
  else:
    val_text_self[i] += ' '
    # print(pred[j])
    val_text_self[i] += id2tag[pred2[j]]
    j += 1
    # item += '3'
```

[152]:
```
file_test_pred = open('dev2.out','w')
for item in val_text:
        file_test_pred.write(item+"\n")
file_test_pred.close()

file_test_pred_self = open('self_dev2.out','w')
for item in val_text_self:
        file_test_pred_self.write(item+"\n")
file_test_pred_self.close()

file_test_pred_test = open('test2.out','w')
```

```
for item in test_text:
        file_test_pred_test.write(item+"\n")
file_test_pred_test.close()
```