CSCI544: Homework Assignment No 4
Nehal Muthukumar
1677301672

TASK - 1: Simple Bidirectional LSTM model

Training Dataset Preparation:

Our approach to preparing the training dataset involves several steps:
1) Word2idx dictionary creation: We start by parsing the training file and creating a dictionary to map words to their respective indices.
2) Label2idx dictionary creation: We also create a separate dictionary to map NER labels to their respective indices.
3) Encoding of Sentences: Each sentence is then encoded into a list of word indices using the word2idx dictionary.
4) Encoding of Labels: Similarly, the NER tags for each sentence are encoded using the label2idx dictionary.
5) Special Words in word2idx: We incorporate three special words in the word2idx dictionary, namely <PAD>, <UNK>, and <UNK_CAPITAL>. The <PAD> word is for padding, <UNK> is for unknown words that start with a lowercase letter, and <UNK_CAPITAL> is for unknown words that start with a capital letter.
6) Handling of Unknown Words: If a word is not present in the word2idx dictionary, we check if it starts with a capital letter or not and map it to the corresponding tag(<UNK_CAPITAL> or <UNK>).
7) Word Frequency Tracking: We keep track of the frequency of words in the training data. If the frequency is 1, we map the word to the <UNK>/<UNK_CAPITAL> tag in the word2idx dictionary based on whether the word starts with a capital letter.
8) Padding of Sentences and Tags: We pad both the sentence list and tag list with the maximum sentence length to make them uniform for batching.
We return three items from the custom dataset, namely the encoded sentences list, the encoded tags list corresponding to each sentence, and the number of words in each sentence, which is used for pad_pack_sequence to speed up training.

Dev and Test Data Preparation:

To prepare the dev and test datasets, we follow a similar approach as the training data but using the word2idx and label2idx dictionaries created from the training data.

1) Encoding of Sentences and Labels: We encode each sentence in the dev and test datasets into a list of word indices using the word2idx dictionary. Similarly, the NER tags for each sentence are encoded using the label2idx dictionary.
2) Handling of Unknown Words: We also map unknown words to their corresponding tags based on whether they start with a capital letter or not, using the same approach as in the training data.

By using the same word2idx and label2idx dictionaries as in the training data, we ensure consistency across all datasets.

Model:

Our NER model was implemented in PyTorch and followed the architecture as provided in Task 1.
To optimize our model, we experimented with different hyperparameters and found the following values to work best:

- Batch Size: 8
- Learning Rate: 0.005
- Epochs: 40
- Step Size: 10
- Gamma: 1
- Momentum: 0.95

We used the Cross-entropy loss function and the Stochastic Gradient Descent (SGD) optimizer. We also initialized the weights in the CrossEntropy loss with the values [1.5, 0.7, 1.6, 2.5, 1.7, 1.5, 1.7, 1.5, 1.5] to balance the class distribution.
We incorporated a scheduler, but we found that a constant learning rate of 0.005 yielded the best F1 score.

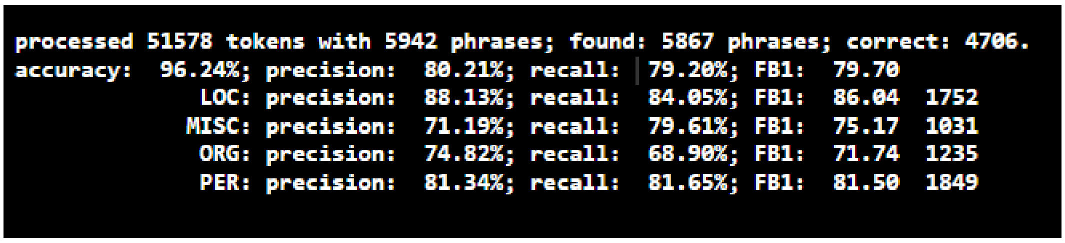
To further optimize the training process, we used the pack_padded_sequence and pad_packed_sequence functions during the forward() pass. This helped us to speed up training by neglecting the padded word indices. We passed the number of words in the original sentence to pack_padded_sequence to do this. After the Bi-LSTM layer, we again padded the output to make it uniform in length.

Model Training:

During training, we were able to achieve a validation loss of 0.2236 and a training loss of 0.0266.

Results:

After training, we tested the model on the dev data by running the Perl script with dev data predictions. The output we obtained from this script was as follows:



```
processed 51578 tokens with 5942 phrases; found: 5867 phrases; correct: 4706.
accuracy:  96.24%; precision:  80.21%; recall:  79.20%; FB1:  79.70
              LOC: precision:  88.13%; recall:  84.05%; FB1:  86.04  1752
             MISC: precision:  71.19%; recall:  79.61%; FB1:  75.17  1031
              ORG: precision:  74.82%; recall:  68.90%; FB1:  71.74  1235
              PER: precision:  81.34%; recall:  81.65%; FB1:  81.50  1849
```

Precision: 80.21%
Recall: 79.20%
F1-score: 79.70%

TASK - 2: Using GloVe word embeddings

Preparing Glove Embedding:

To utilize pre-trained word embeddings, we used the 'glove.6B.100d.txt' file and read it to create an embedding dictionary. We then created an embedding matrix using this dictionary with the addition of three special tags - <pad>, <unk>, and <unk_capital>.

The embedding vector for <pad> is set to all zeros. The vector for <unk> is calculated as the mean of all the other vectors in the embedding dictionary. For <unk_capital>, we take half of the <unk> vector.
All other available words in the embedding dictionary take vectors to form the embedding matrix.

We also created the word2idx dictionary from the vocabulary of the embedding dictionary.

Train Dataset Preparation:

In Task 2, we have the word2idx dictionary beforehand, which we created from Glove. Additionally, we use boolean masking to deal with word capitalization.

To prepare the training dataset, we follow these steps:

1) We create the label2idx dictionary to map labels to an index.
2) We encode each sentence into a list of word indices using the word2idx dictionary.
3) We similarly encode the NER tags using the label2idx dictionary.
4) For each word in a sentence, we check if the word is Capitalized and add a boolean mask value of 1; if it does, otherwise, 0. We keep track of this boolean mask for each sentence.
5) If a word is not present in the vocabulary, we map it to the corresponding unknown tag in word2idx based on whether it has a capital letter or not.
6) Finally, we pad the sentence list, tag list, and the boolean mask with the maximum sentence length to make them uniform for batching.

The custom dataset returns 4 items:

a) The encoded sentences list.
b) The encoded tags list corresponding to each sentence.
c) The number of words in each sentence (to be used for pad_pack_sequence to speed up the training).
d) The boolean mask.

Dev and Test Data preparation:

We follow a similar approach to the training data to prepare the dev and test datasets.
1) Encoding of Sentences and Labels: We encode each sentence in the dev and test datasets into a list of word indices using the word2idx dictionary. Similarly, the NER tags for each sentence are encoded using the label2idx dictionary.
2) Handling of Unknown Words: We also map unknown words to their corresponding tags based on whether they start with a capital letter or not, using the same approach as in the training data.
3) Boolean mask: Based on whether a word is capitalized or not, we create the boolean mask for each sentence in the dev and test.

Model:

We pass the pre-trained Glove embedding matrix to the model's embedding layers. After the embedding is done, we stack the words with the corresponding boolean masks. This boolean mask is added as an extra dimension to the word embedding.

The hyperparameters used in the model are as follows:

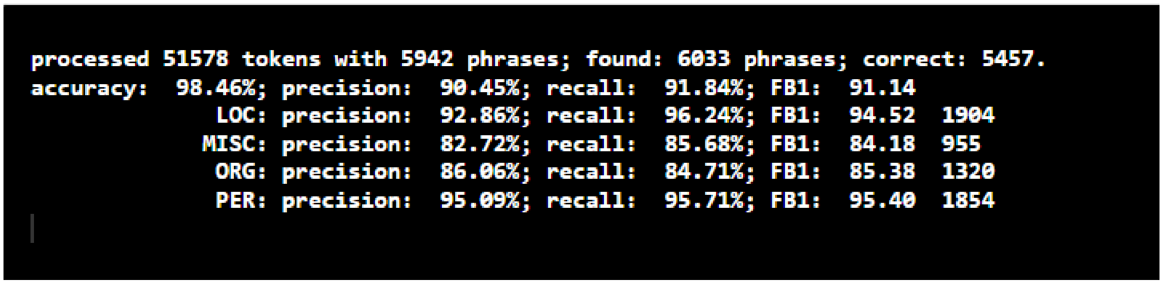
- Batch size: 32
- Learning rate: 0.2
- Epochs: 30
- Step size: 20
- Gamma: 1

We used a scheduler, but the model with a constant learning rate of 0.2 gave the best F1 score. Additionally, we used a momentum of 0.9 in the SGD optimizer. In the cross-entropy loss, we used initial weights that were calculated as follows:
weights = [(3.5 - frequency/sum_frequency) for frequency in tag_frequencies]

Model Training and Result:

During training, we reached a validation loss of 0.0650 and a training loss of 0.0047.

After training, we tested the model on the dev data by running the Perl script with dev data predictions. The output we obtained from this script was as follows:



```
processed 51578 tokens with 5942 phrases; found: 6033 phrases; correct: 5457.
accuracy:  98.46%; precision:  90.45%; recall:  91.84%; FB1:  91.14
              LOC: precision:  92.86%; recall:  96.24%; FB1:  94.52  1904
             MISC: precision:  82.72%; recall:  85.68%; FB1:  84.18  955
              ORG: precision:  86.06%; recall:  84.71%; FB1:  85.38  1320
              PER: precision:  95.09%; recall:  95.71%; FB1:  95.40  1854
```

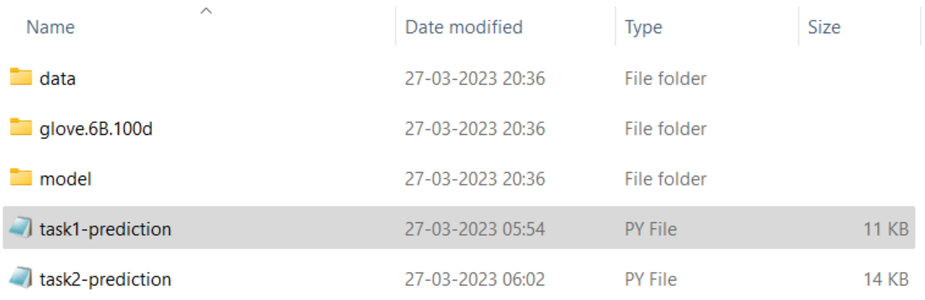
- **Precision:** 90.45%
- **Recall:** 91.84%
- **F1-score:** 91.14%

To Run the Code:

We have separate scripts for TASK 1 and TASK 2:

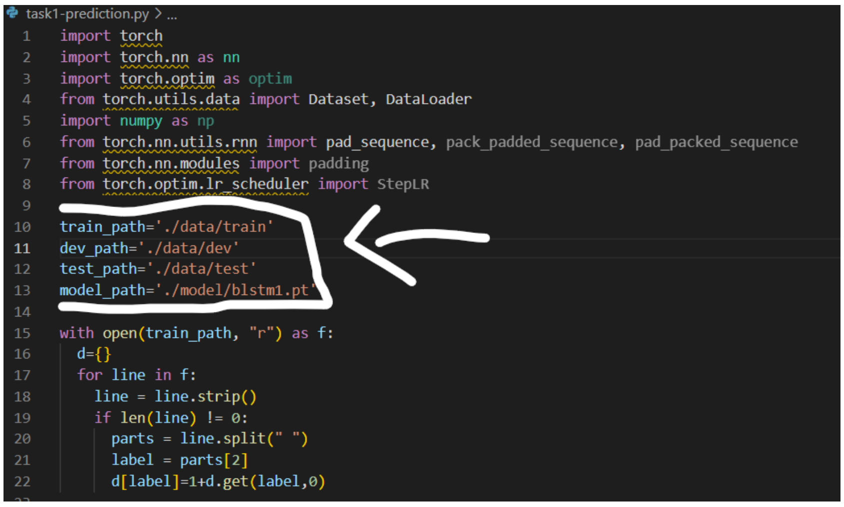
TASK 1:

1) **'task1-prediction.py'**
1.1) For prediction on dev and test data, use the script 'task1-prediction.py.'
1.2) Place the folders Data, Glove, and Model in the same directory where the script is present



Name	Date modified	Type	Size
data	27-03-2023 20:36	File folder	
glove.6B.100d	27-03-2023 20:36	File folder	
model	27-03-2023 20:36	File folder	
task1-prediction	27-03-2023 05:54	PY File	11 KB
task2-prediction	27-03-2023 06:02	PY File	14 KB

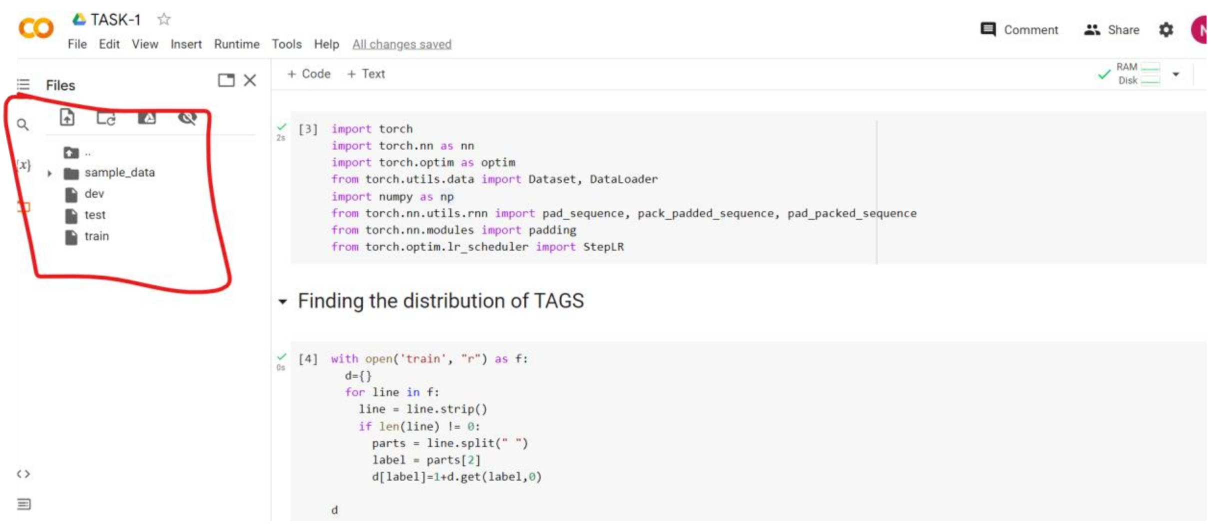
(OR) You can also directly give the filename in the code 'task1-prediction.py' in the first few lines.



```
task1-prediction.py > ...
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence, pad_packed_sequence
from torch.nn.modules import padding
from torch.optim.lr_scheduler import StepLR

train_path='./data/train'
dev_path='./data/dev'
test_path='./data/test'
model_path='./model/blstm1.pt'

with open(train_path, "r") as f:
  d={}
  for line in f:
    line = line.strip()
    if len(line) != 0:
      parts = line.split(" ")
      label = parts[2]
      d[label]=1+d.get(label,0)
```

1.3) The 'model' folder contains the model 'blstm1.pt' for TASK 1
1.4) Simply run the file using the command "**python task1-prediction.py**."
1.5) The script will create 'test1.out' and 'dev1.out' files in the same directory.

2) ‘TASK1.ipynb’
2.1) 'TASK1.ipynb' contains all the code from data preparation, model, training, and prediction on dev and test
2.2) Upload the file to Google Drive and open it on Colab
2.3) Upload the files 'train,' 'dev,' and 'test' in Colab



```
[3] import torch
    import torch.nn as nn
    import torch.optim as optim
    from torch.utils.data import Dataset, DataLoader
    import numpy as np
    from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence, pad_packed_sequence
    from torch.nn.modules import padding
    from torch.optim.lr_scheduler import StepLR
```

Finding the distribution of TAGS

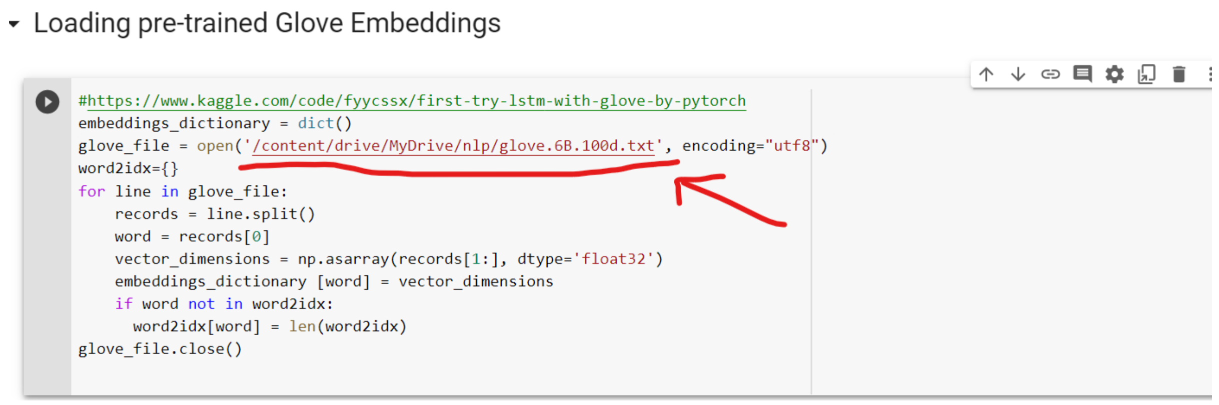
```
[4] with open('train', "r") as f:
      d={}
      for line in f:
        line = line.strip()
        if len(line) != 0:
          parts = line.split(" ")
          label = parts[2]
          d[label]=1+d.get(label,0)

    d
```

2.4) The Notebook is documented, so run the cells one by one to see all results

TASK 2:

1) 'task2-prediction.py'
1.1) For prediction on dev and test data, use the script 'task2-prediction.py.'
1.2) Place the folders Data, Glove, and Model in the same directory where the script is present (OR) You can also directly give the filename in the code 'task2-prediction.py' in the first few lines
1.3) The 'model' folder contains the model 'blstm2.pt' for TASK 2
1.4) Simply run the file using the command "**python task2-prediction.py**."
1.5) The script will create 'test2.out' and 'dev2.out' files in the same directory

2) 'TASK2.ipynb'
2.1) 'TASK2.ipynb' contains all the code from data preparation, model, training, and prediction on dev and test
2.2) Upload the file to Google Drive and open it on Colab
2.3) Upload the files 'train', 'dev', and 'test' in Colab
2.4) The GloVe path should be given in the Notebook. We can upload it and use the path



```
#https://www.kaggle.com/code/fyycssx/first-try-lstm-with-glove-by-pytorch
embeddings_dictionary = dict()
glove_file = open('/content/drive/MyDrive/nlp/glove.6B.100d.txt', encoding="utf8")
word2idx={}
for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = np.asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
    if word not in word2idx:
      word2idx[word] = len(word2idx)
glove_file.close()
```

2.5) The Notebook is documented, so run the cells one by one to see all results