```
!git clone https://github.com/tmikolov/word2vec.git
```

```
Cloning into 'word2vec'...
remote: Enumerating objects: 148, done.
remote: Total 148 (delta 0), reused 0 (delta 0), pack-reused 148
Receiving objects: 100% (148/148), 119.14 KiB | 2.29 MiB/s, done.
Resolving deltas: 100% (86/86), done.
```

Clone word2vec code from Git and execute make file

```
%cd /content/word2vec
```

```
/content/word2vec
```

```
!make
```

Word embeddings snippet for 'of'

of -0.565323 0.236813 0.272062 -1.426221 0.428859 -1.386884 -0.589239 2.778557 -1.440811 0.791160 2.015024 0.394671 1.053778 3.489750

```
!wget -c http://nlp.cs.rpi.edu/course/spring19/enwiki.tar.gz
```

```
--2019-02-21 00:24:40--  http://nlp.cs.rpi.edu/course/spring19/enwiki.tar.gz
Resolving nlp.cs.rpi.edu (nlp.cs.rpi.edu)... 128.113.126.107
Connecting to nlp.cs.rpi.edu (nlp.cs.rpi.edu)|128.113.126.107|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 515364205 (491M) [application/x-gzip]
Saving to: ‘enwiki.tar.gz’

enwiki.tar.gz       100%[===================>] 491.49M  26.0MB/s    in 25s

2019-02-21 00:25:05 (19.3 MB/s) - ‘enwiki.tar.gz’ saved [515364205/515364205]
```

```
!tar -xvzf enwiki.tar.gz
```

```
enwiki.sample.txt
```

```
!./word2vec -train enwiki.sample.txt -output emboutput_1.txt -size 50 -window 5 -negative 1(
```

```
Starting training using file enwiki.sample.txt
Vocab size: 460155
Words in train file: 247257644
Alpha: 0.004003  Progress: 91.99%  Words/thread/sec: 235.24k  ^C
```

Word2vec is trained to implement wiki english corpus on for 50 vector dimension

```
!git clone https://github.com/glample/tagger.git
```

```
Cloning into 'tagger'...
remote: Enumerating objects: 61, done.
remote: Total 61 (delta 0), reused 0 (delta 0), pack-reused 61
Unpacking objects: 100% (61/61), done.
```

```
!python trainreduce.py
```

```
!wget -c http://nlp.cs.rpi.edu/course/spring19/name_tagging.tar.gz
```

```
!tar -xvzf name_tagging.tar.gz
```

The test file is used to get tokens without BIO tagging for usage in LSTM tagger

```
filename = open('/content/eng.test.clean.bio')
output = open('/content/testing.txt','w')
Final_list=[]
a=''
for i in filename:
  if i != '\n':
    #print i
    file = i.strip()
    file=file.split(' ')
    for j in range (0,len(file)):
      if j==0:
        a=a+file[j]+' '
  else:
    a=a.strip()
    #Final_list.append(a)
    output.write(a)
    print(a)
    a=''
    a='\n'
    output.write(a)
output.close()
filename.close()
```

Limit the train size to 10000 for LSTM tagger as epochs are fewer and that may compromise neural network iterations and weight assignment to each tag

```
filename = open('eng.train.clean.bio')
output = open('eng.train.clean.bio.txt','w')
Final_list=[]
a=''
for i in filename:
  if i == '\n':
    a=a.strip()
    c = c+1
    #Final_list.append(a)
    output.write(a)
    print(a)
    a=''
   if c == 10000:
    break
```

```
%cd tagger
```

```
/content/tagger
```

```
!cat /content/tagger/emboutput_1.txt
```

```
from google.colab import files
files.upload()
```

Choose Files emboutput_1.txt

- **emboutput_1.txt**(text/plain) - 222518365 bytes, last modified: 2/20/2019 - 100% done

```
Saving emboutput_1.txt to emboutput_1.txt
```

```
!./train.py --train /content/eng.train.clean.bio.txt --dev /content/eng.dev.clean.bio --test
```

```
                  FAC: precision:  56.03%; recall:  19.95%; FB1:  29.42  141
                  GPE: precision:  85.35%; recall:  74.31%; FB1:  79.45  1454
                  LOC: precision:  48.44%; recall:  24.31%; FB1:  32.38  128
                  ORG: precision:  66.15%; recall:  36.68%; FB1:  47.20  910
                  PER: precision:  86.63%; recall:  46.97%; FB1:  60.91  920
         ID      NE  Total      O  S-GPE  S-PER  S-ORG  B-PER  E-PER  B-ORG  E-ORG  I-ORG  I-PE
          0       O  79418  78947     39     34    116     23     15     33     39     56     1
          1   S-GPE   1439    254   1054      8     49      3      3     18      4      1
          2   S-PER   1303    808      7    455     11      3     13      1      1      1
          3   S-ORG   1318    804     27     10    432      5      2     10     15      0
          4   B-PER    394     27      2      3      0    348      1      4      0      0
          5   E-PER    394     28      0      4      0      1    347      0      4      0
          6   B-ORG    323     44     12      4      8      9      0    182      1     18
          7   E-ORG    323     72      3      0      8      0      6      1    191     10
          8   I-ORG    402     76      0      0      0      1      4      7      5    272
          9   I-PER    128     16      0      0      0      2      2      0      0      1    1€
         10   B-GPE    231      2      7      0      1      3      0     16      0      0
         11   E-GPE    231      5      6      0      0      0      5      0      8      7
         12   S-LOC    190    142     11      0      1      0      0      1      0      1
         13   I-GPE    169      5      0      0      0      0      0      0      0     21
         14   B-LOC     65      6      0      0      0      0      0      0      0      0
```

```
!./tagger.py --model /content/tagger/models/tag_scheme=iobes,lower=False,zeros=False,char_di
```

```
Loading model...
Compiling...
WARNING (theano.tensor.blas): We did not find a dynamic library in the library_dir of
WARNING (theano.tensor.blas): We did not find a dynamic library in the library_dir of
Tagging...
---- 1 lines tagged in 43.6801s ----

8050, cost average: 0.908462
```

I have implemented dataframes to store tags, calculate tags based on 'O' and otherwise checks for true positive , false positive and false negatives.

Calculating precision and recall is actually quite easy. Imagine there are 100 positive cases among 10,000 cases. You want to predict which ones are positive, and you pick 200 to have a better chance of catching many of the 100 positive cases. You record the IDs of your predictions, and when you get the actual results you sum up how many times you were right or wrong. There are four ways of being right or wrong:

TN / True Negative: case was negative and predicted negative TP / True Positive: case was positive and predicted positive FN / False Negative: case was positive but predicted negative FP / False Positive: case was negative but predicted positive

```
8700, cost average: 1.350124
```

```
import pandas as pd
list1 =[]
list2=[]
set_s = set()
set_b = set()

with open('C:/Users/Yaminie/Downloads/yamini_testing_tagged.txt', 'r', encoding = "utf8") as
    with open('C:/Users/Yaminie/Downloads/name_tagging/eng.test.clean.bio', 'r', encoding =
      for line in file1:
          for word in line.split():
              #print(word)
              #print("-------------------------------")
              word = word.split("__")
              list1.append(word)
```

```
                set_s.add(word[1])
        print(set_s)
        df1 = pd.DataFrame(list1,columns =['Word','Tag'])
        print(df1)
        for l in file2:
            l=l.strip()
            l = l.split(" ")
            #print(l)
            if(l[0] != ''):
                list2.append(l)
                set_b.add(l[1])
        df2 = pd.DataFrame(list2,columns =['Word','Tag'])
        print(df2)
        print(set_b)

      tag1 = df1['Tag']
      tag2 = df2['Tag']
      true_positive = 0
      false_negative = 0
      false_positive = 0
      for i in range (0, len(tag1)):
            if tag1.loc[i] == tag2.loc[i] and tag1.loc[i]!='O':
                true_positive +=1
            elif tag1.loc[i]!='O' and tag2.loc[i] =='O':
                false_positive +=1
            elif tag1.loc[i] == 'O' and tag2.loc[i] != 'O':
                false_negative +=1
            elif tag1.loc[i]!='O' and tag2.loc[i] != 'O':
                false_negative +=1
                false_positive +=1

      precision = true_positive/(true_positive+false_positive)
      recall = true_positive/ (true_positive+false_negative)
      fscore = 2*(recall * precision) / (recall + precision)
      print(precision*100)
      print(recall*100)
      print(fscore)
```

Upon executing script for precision recall , following were the results for English corpus:

1. Precision : 79.89977728285078
2. Recall : 56.84669219595933
3. F-score : 0.6643005940899622

Performance and Error Analysis

• Worked: Successfully able to download and implement word2vec, used word vector dimension to 50. It was convenient to use word2vec library for vector implementation which equate to one hot encoding. Furthermore, LSTM tagger was easy access and due to large size of english corpus, it was set to 10000 using a python script.

• Improvements: BIO format chunks were not properly tagged, i.e. chunking in the form that 'Amazon Prime' is not regarded as two separate and taken as one. Observed that new lines caused confusion in tagging -- my tags were not implemeted properly without strip function.

• Training data used: 10000 of wiki dataset and complete dataset was used for English

• Epochs ran: 16 for English, 20 for Spanish

Last epoch metrics:

accuracy: 97.53%; precision: 81.12%; recall: 76.84%; FB1: 78.92 FAC: precision: 66.37%; recall: 69.44%; FB1: 67.87 113 GPE: precision: 88.53%; recall: 89.62%; FB1: 89.07 2205 LOC: precision: 53.60%; recall: 45.27%; FB1: 49.08 250 ORG: precision: 68.47%; recall: 58.35%; FB1: 63.01 1367 PER: precision: 87.69%; recall: 84.82%; FB1: 86.23 1446

Error Analysis:

1. The size of corpus should be identified to implement it within runtime.
2. Tags were not easily identified without strip function.
3. Further scope is chunking in a manner that recognition can be improved for Inside or Beginning.
4. If one word is identified incorrectly in a sentence the others are as well.
5. Test file using tagger was causing differences.