CS563: Natural Language Processing

Indian Institute of Technology Patna

Assignment 2

Named Entity Recognition

Group Members:

- Muhammed Sinan C K (1901CS38)
- Vaishakh Sreekanth Menon (1901CS68)
- Varsha Tumburu (1901CS69)

Aim

Implementing HMM on Named Entity Recognition(NER) task by using the Viterbi algorithm. We attempt to BIO tag the test dataset provided based on our trained HMM model and present scores of accuracy, recall and F1 for evaluation.

Dataset

Understanding dataset...

The two main tasks of the Named Entity Recognition are namely: capturing the *boundaries of named entities* in a text and *classifying the named entity types*. To demonstrate the HMM, we will illustrate a simple NLP scenario for boundary segmentation. There are several formats for classifying NER boundaries, as shown below:

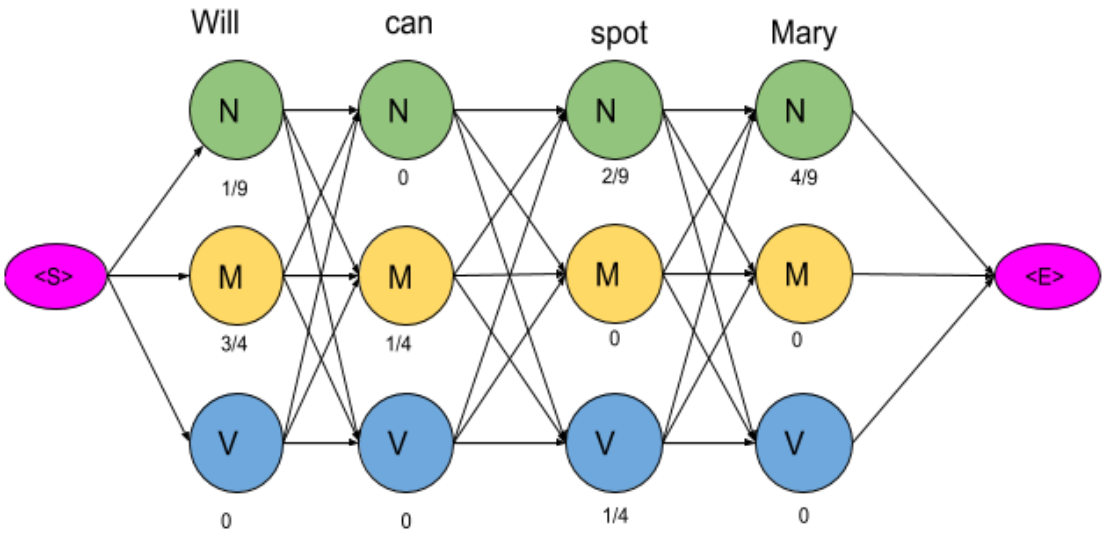
'B' : Beginning of named entity
'I' : Inside of named entity
'O' : Outside of named entity

In our scenario below, will proceed to apply the 'BIO' labels. Note that the 'BIO' labels are considered the hidden states, whereas the vocabulary tokens are the observations.

```
679   RT  O
680   @NPR    O
681   :   O
682   Song    O
683   of  O
684   the O
685   Day O
686   :   O
687   Blonde  B
688   Redhead I
689   :   O
690   A   B
691   Different   I
692   Kind    I
693   Of  I
694   Ache    I
695   http://n.pr/9O8yXE  O
696
```

The dataset beside is a part of the training data provided. It shows how words are classified under B, I, and O. Proper nouns are usually considered B and then continue with I, and then end with O.

Viterbi algorithm



The above figure represents the entire graph structure showing all transition probabilities between tags, emission probabilities for words and initial probabilities (pi). Multiplying the 3 for a word will give the probability of having that tag mapped with the word. To find these values by brute force and then compare to find the highest will be of high time complexity. Because if there are k POS tags and n words, then time complexity will be $o(n^k)$ which is exponential.

This is where viterbi algorithm comes into play. It uses dynamic programming which reduces the time complexity to $O(k^2 n)$.

In this approach, we create a 2D matrix of size nxk called dp where n is the length of the sentence and k is the number of tags. Here dp[index][tag] denotes the best probability for mapping the word (sentence[index]) to the tag.

After calculating all probabilities of 2D matrix dp , we can backtrack from end of the sentence to beginning of the sentence to find the best tag sequence of a given sentence.

Pseudo code

```
Iterate through all prev_tag in index-1:
    dp[index][tag]= max(dp[index-1][prev_tag] x
transition_probability[prev_tag to tag] x
emission_probability(current word | tag), dp[index][tag])
```

Unigram vs Bigram HMM Model Assumption

First Order HMM (Bigram model) is based on the assumption that the tag of a word depends on the tag of the previous word. So transition probabilities in this scenario connect two tags.

On the other hand, the unigram model does not require the knowledge of the previous tag to calculate the probability of the current tag. So, transition probabilities here are basically the initial probabilities of tags (i.e, π).

Similarly an n-gram model depends on the sequence of n previous tags. In the case of tagging words, the bigram model usually works well for POS tagging, since we can usually predict that an adverb comes near a verb, adjective near a noun, etc.

Thus, the only difference of implementing these models is the step of calculating the transition probability. Emission and initial probabilities remain the same.

Implementation

We use 5 fold cross validation to train these HMM models and then note down their accuracy, precision, recall and F1 scores. We can notice that the bigram model is always slightly better than our unigram.

Emission probabilities for one train test split:

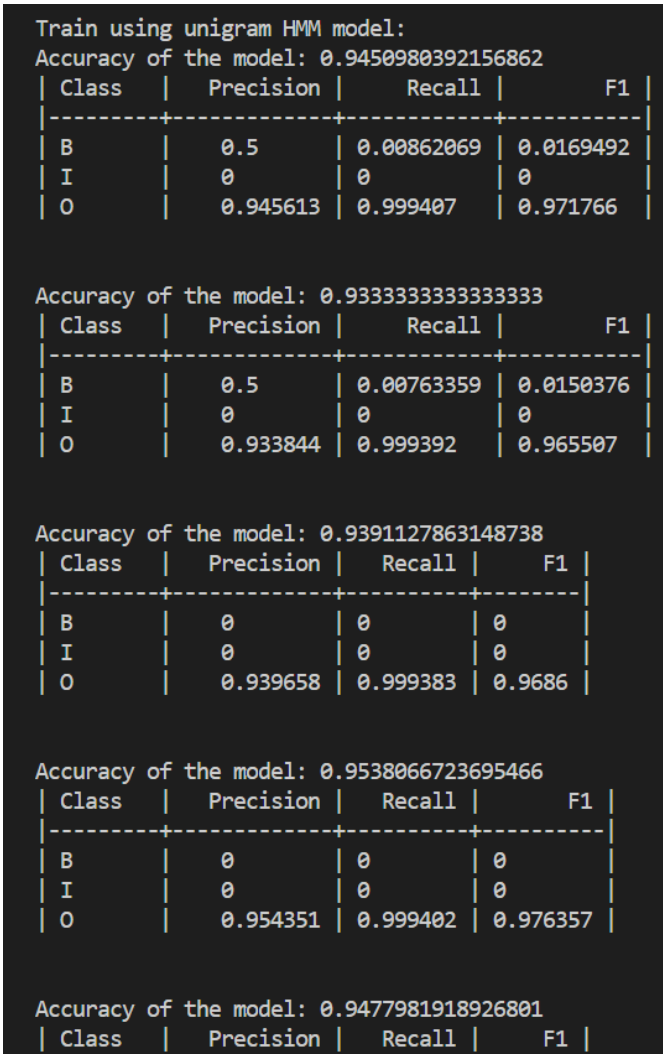
	B	I	O
zac	0.002146	NaN	NaN
tumblr	0.002146	NaN	NaN
nijverdal	0.002146	NaN	NaN
the	0.027897	0.013423	0.021030
wow	0.002146	NaN	0.000075
...	...	...	...
tested	NaN	NaN	0.000075
positive	NaN	NaN	0.000075
cocaine	NaN	NaN	0.000075
sources	NaN	NaN	0.000075
http://bi	NaN	NaN	0.000075

Similarly, other probabilities have also been calculated for different folds of the dataset. Let us check a prediction on a random sentence, picking up words from the dataset. The below sentence is well predicted using any model.

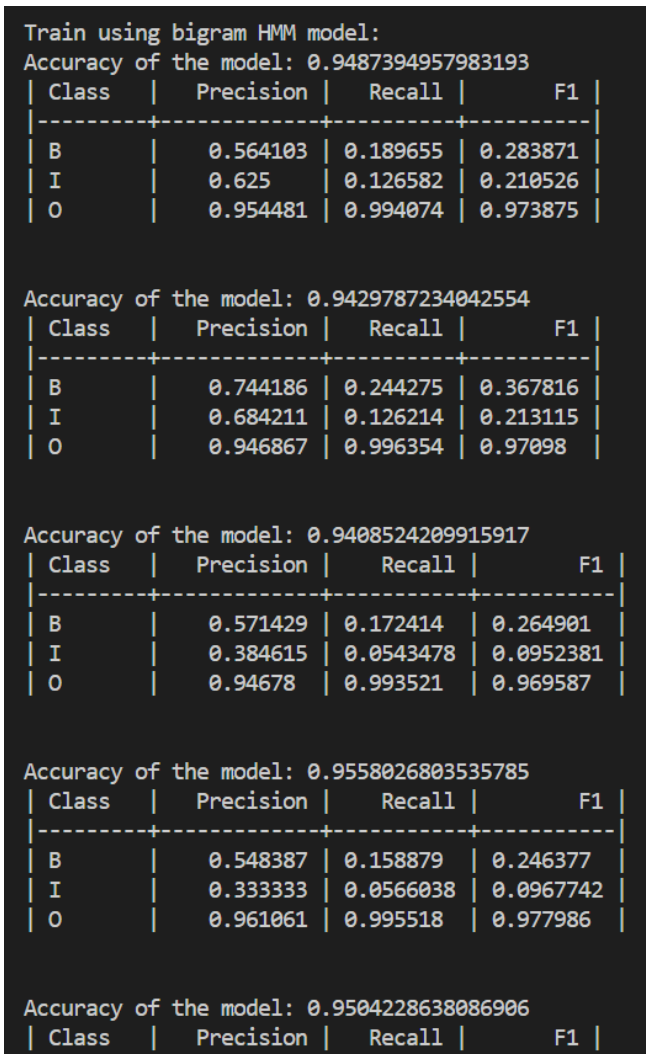
1	justin	B
2	bieber	I
3	is	O
4	touring	O
5	with	O
6	michael	B
7	in	O
8	ohio	B

Usually bigram model should perform *much* better than the unigram (only 0.4% improvement here). However, this time we can attribute these results to the nature of dataset. BIO tagging involves many O's and barely any B's and I's whenever a named entity is found. Since the vast majority contain O's, whether a tag depends on the previous one or not does not make much of a difference here. Even if most are tagged Os according to pi, we will still get a high probability as shown below.

```
Train using unigram HMM model:
unigram hmm average accuracy =  0.943829804625224
Train using bigram HMM model:
bigram hmm average accuracy =  0.9477592368712872

---Test sentence predicted!---
```



```
Train using unigram HMM model:
Accuracy of the model: 0.9450980392156862
| Class   |   Precision |     Recall |        F1 |
|---------+-------------+------------+-----------|
| B       |    0.5      | 0.00862069 | 0.0169492 |
| I       |    0        | 0          | 0         |
| O       |    0.945613 | 0.999407   | 0.971766  |


Accuracy of the model: 0.9333333333333333
| Class   |   Precision |     Recall |        F1 |
|---------+-------------+------------+-----------|
| B       |    0.5      | 0.00763359 | 0.0150376 |
| I       |    0        | 0          | 0         |
| O       |    0.933844 | 0.999392   | 0.965507  |


Accuracy of the model: 0.9391127863148738
| Class   |   Precision |   Recall |     F1 |
|---------+-------------+----------+--------|
| B       |    0        | 0        | 0      |
| I       |    0        | 0        | 0      |
| O       |    0.939658 | 0.999383 | 0.9686 |


Accuracy of the model: 0.9538066723695466
| Class   |   Precision |   Recall |       F1 |
|---------+-------------+----------+----------|
| B       |    0        | 0        | 0        |
| I       |    0        | 0        | 0        |
| O       |    0.954351 | 0.999402 | 0.976357 |


Accuracy of the model: 0.9477981918926801
| Class   |   Precision |   Recall |     F1 |
```



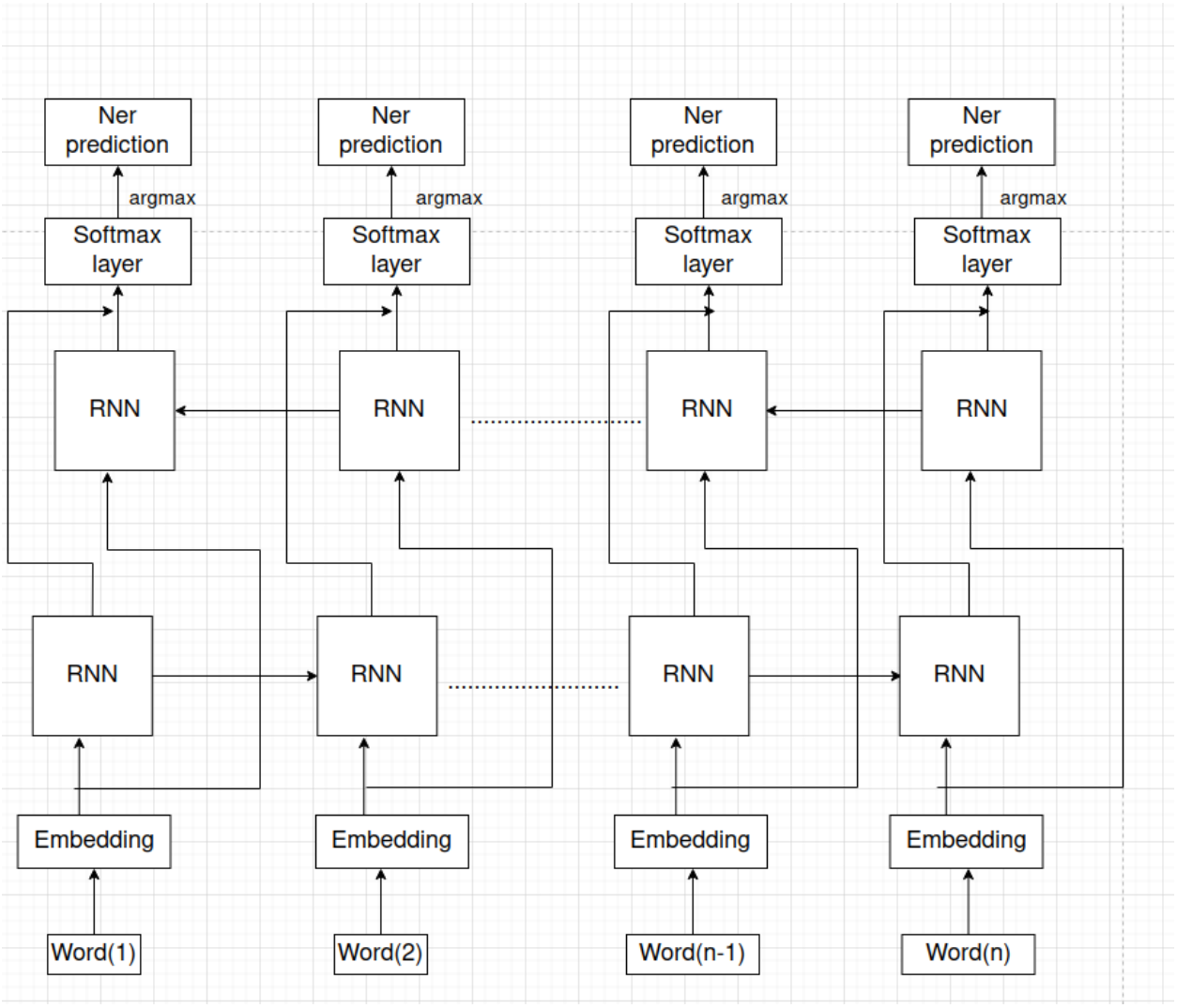
```
Train using bigram HMM model:
Accuracy of the model: 0.9487394957983193
| Class   |   Precision |   Recall |       F1 |
|---------+-------------+----------+----------|
| B       |    0.564103 | 0.189655 | 0.283871 |
| I       |    0.625    | 0.126582 | 0.210526 |
| O       |    0.954481 | 0.994074 | 0.973875 |


Accuracy of the model: 0.9429787234042554
| Class   |   Precision |   Recall |       F1 |
|---------+-------------+----------+----------|
| B       |    0.744186 | 0.244275 | 0.367816 |
| I       |    0.684211 | 0.126214 | 0.213115 |
| O       |    0.946867 | 0.996354 | 0.97098  |


Accuracy of the model: 0.9408524209915917
| Class   |   Precision |    Recall |        F1 |
|---------+-------------+-----------+-----------|
| B       |    0.571429 | 0.172414  | 0.264901  |
| I       |    0.384615 | 0.0543478 | 0.0952381 |
| O       |    0.94678  | 0.993521  | 0.969587  |


Accuracy of the model: 0.9558026803535785
| Class   |   Precision |    Recall |        F1 |
|---------+-------------+-----------+-----------|
| B       |    0.548387 | 0.158879  | 0.246377  |
| I       |    0.333333 | 0.0566038 | 0.0967742 |
| O       |    0.961061 | 0.995518  | 0.977986  |


Accuracy of the model: 0.9504228638086906
| Class   |   Precision |   Recall |       F1 |
```

The scores shown above are the metrics for each fold training and testing on the given dataset. Notice how unigram has most zeroes denoting there were no predicted values of the respective tag and yet, the accuracy is quite high overall.

Bidirectional RNN Architecture

A bidirectional recurrent neural network (BiRNN) is a type of artificial neural network that processes sequential data in both forward and backward directions. BiRNNs have two hidden layers: one that processes the input sequence in the forward direction and another that processes it in the backward direction. The outputs of these two layers are then combined to produce a final output sequence. This approach allows the network to take into account not only past

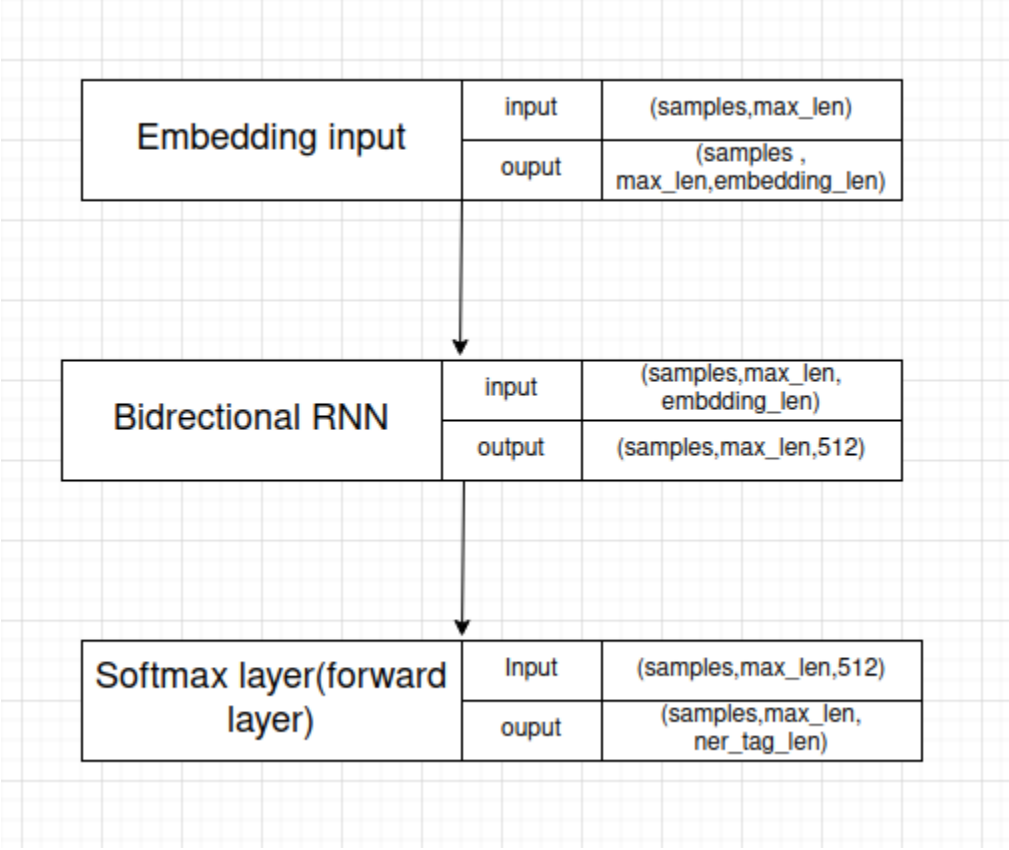
information but also future information when making predictions. BiRNNs are particularly useful for tasks that require understanding the context of a sequence, such as speech recognition and natural language processing. They have shown significant improvements in accuracy compared to traditional RNNs and have become a popular choice for many sequence modeling tasks.



For our ner prediction we will pass each word of the sentence to the word to the word embedding which will give a vector representation of a word, then we will pass these words to the forward RNN sequentially. And in each time stamp when passing a word RNN will produce output and hidden state, where hidden state will pass to the previous hidden state to the next word . This way will

take the output from the RNN going backward also . Then we will combine both backward RNN output and forward RNN output . Then we will pass combined output to the feedforward network to change the output length to tag length . Then final output will be passed to the softmax layer such that we can get the probabilities of each named entity for the current word. After that we will take argmax (taking highest probability) and according to the index in the vector we will find which ner current word belongs to . While training RNN architecture will learn to predict the NER and in testing RNN will be able to predict the NER for a given sentence using trained weights. We can use different types of RNN such as GRU and LSTM. Those are 2 variants of RNN which have memory to remember previous states and forget unwanted things.

Possible dimensions for this architecture are:



Samples- number of samples
Max_len- maximum sentence length
embedding len - Length of embedding

Ner_tag_len - total number of ner tags
Here for word embedding we can use embedding such as word2vec . And each layer of RNN (forward and backward) will produce 256 length dimensions for each word, and that will be concatenated to get 512 . Then this will be passed to the feedforward network and softmax layer to get probabilities of ner prediction. And after that we will take argmax to get the ner predicted output. Here in feed forward we can use one or two hidden layers accordingly.