Named Entity Recognition: A NLP Implementation

Cecilia Aponte (1822225)
Sapienza Università di Roma
Natural Language Processing

Abstract

A Named Entity Recognition (NER) task is implemented to understand words as a location (LOC), an organization (ORG), a person (PER), or other (O). The neural network architecture has a pre-trained Sense Embedding layer, two Bi-LSTM layers with dropout, and Adam optimizer. The model resulted in a F1 score of 0.806 for all classes.

1 Introduction

Language for an adult that understands the dialect is simple and effortless. However in order for machines to understand this, Natural Language Processing is necessary to be able to read, understand, and derive meaning to language. One way to derive meaning is by processing text using Named Entity Recognition where the machine can recognize certain words as a location (LOC), an organization (ORG), a person (PER), or other (O).

A NER neural network algorithm is used with a Bidirectional Long short Term Memory (Bi-LSTM) architecture and an embedding layer previously trained using Sense Tags. The Bi-LSTM uses past and future labels in a sequence to identify and classify named entities in text that can be later used for tasks such as recommendation systems.

2 Pre-processing

The text used for training contained 100k sentences, while the validation and testing contained 14k each. Each sentence is previously tokenized and tagged with their named entity. The data is all left as is (it was also tested with a) all lowercase and b) only first word in sentence lowered) and later processed to set them up in the structure needed for the model.

As seen in Figure 1, the steps to structure the data shown is as follows:

1. (Optional - Figure 2) Each token is replaced by its word sense ID as referenced to its BabelNet ID, since embeddings tokens are as: $lemma\_bnID$.

2. Divide sentences in equal amounts of tokens based on window size (and shifted, if needed). For remaining tokens, an empty pad is added.

3. Build the word/label vocabularies by a) counting each token in the dataset, b) adding all tokens (and index) that have a minimum frequency into a dictionary, and c) adding all special tokens such as $< pad >$ and $< unk >$.

4. Encode each token to their numerical value given by the vocabulary index. For tokens: None and not-in-vocabulary, the index of $< pad >$ and $< unk >$ are added, respectively.

5. Setup the data into batches; those samples are processed into the model and then the model's internal parameters are updated.

6. Create the embeddings by a) loading the saved Sense word2vec vectors, b) creating a tensor of size (vocab length, embed dim) with random values, and c) update with the saved vector the indices of the embeddings for the token that are included in the vocabulary.

3 Implementation Approach

The pre-processed data is inputted into the model, which contains a trained Sense Embedding layer, two Bi-LSTM layers with dropout, and a linear fully-connected layer for classification. A general example is shown in Figure 3.

The trained Sense embedding layer was trained using word2vec and sense-annotated dataset with characteristics in Figure 4. It provides a multi-dimensional vector space where words with similar

meanings are closer together (Figure 5), giving greater context to tokens for NER.

The two Bi-LSTM layers provide temporal context by keeping past and future tokens into the long and short term memory, as seen in Figure 6. Dropout is added to regularize the model from overfitting by excluding some 'neurons' from being activated. This is fully connected in a linear layer as the output of the model.

Cross Entropy Loss is used to provide the measure of entropy loss of all classes and the softmax output as a probability measure. This softmax probability helps the system choose the NER class with the maximum value. The model is optimized using the Adam Optimizer, since it provides a per-parameter learning rate to help with NER sparse gradients that could cause vanishing gradients and an inability to tune its weights and provide an accurate system; it's also based on the gradients changing values to help with noisy data.

4 Experimental Setup and Results

The model was trained using Google Colab's GPU and experimented with various different parameters and data processing, summarized in Figure 7 and 8-14. The main model improvements are (as shown highlighted in Figure 15):

a) Sense Embedding Bi-LSTM layer enabled more context for classification, however it did not add much improvement probably because it did not provide differences for grouped named entities (e.g. 'Sara A' and 'University of New York').

b) That was attempted to be solved by leaving capitalized words as is. This was the highest improvement of F1 by 0.1, as shown in Figure 16.

c) To potentially solve classification issues with LOC and ORG, lowercase words were Sense tagged with their most common synset.

d) Figure 21 revealed that most wrong labels were chosen as "O" and concluded that the unbalanced dataset with more 'O' labels samples was affecting the results. To balance each class and calculate loss proportionally, weights (Figure 18) were added. The results for the experiments show a decrease in F1 as seen in Figure 17 and 20. However, Figure 22 shows an improvement with mistakes decrease in 'O' labels from 55% to 15% and increase in F1 for 'ORG' from 54% to 68%.

The chosen model is (b), due to constraints with nltk.wordnet library and testing as instructed and (d)'s decrease in performance. The model's results in Figure 19 shows $F1score = 0.806$; 'ORG' has the lowest F1 due to its complexity to learn since it has words that on their own would be labeled otherwise (e.g. Expected: 'University of New York' $\rightarrow$ all 'ORG'. Predicted : 'University' $\rightarrow$ 'ORG', 'of' $\rightarrow$ 'O', 'New York' $\rightarrow$ both 'LOC').

5 Analysis and Future Improvements

The chosen model with F1 score of 0.806, has been structured and tested significantly. However the inability for some of the experiments to improve the model as expected confirms the need to add robustness to the model.

Since most of the problems are between 'LOC' and 'ORG' given their similar structure, extra features can be added to highlight more those differences and learn them more precisely. These features could be added after the embedding layer is built at the end of the vector for each word. These features would include: capitalization of 1) first letter 2) all letters in word, capitalization of the word(s) to the 3) left and 4) right, 5) location in sentence (start, mid, end) 6) word containing a mix of letters and digits or 7) all digits. These features have the potential to add more description to the differences with mainly 'LOC', 'PER', and 'ORG'. For example, feature (2) would help ORG since many of these are all capitalized (e.g. UN, IBM) and is information lost during processing.

If adding the features in this location is not beneficial, a CFR Model can be added after the Bi-LSTM layer to account for those information from the adjacent labels. This would mainly solve (similar to (3) above) the issue between 'LOC' and 'ORG'; for example, "University of New York" and "New York" should be tagged differently due to the neighboring words. A Bi-CRF as expained in (3) would also take into account previous words, like "University of" in this example.

Otherwise, N-grams can be used to provide structure mainly for 'ORG', since they include N-words usually grouped together to form a single entity.

6 Conclusion

This paper proposes for NER classification a Bi-LSTM model with Sense embeddings. Even though the the performance does not reach competitive models, further improvements are promising such as the addition of features for each token either to the input or after the Bi-LSTM layer through the CFR Model.

7 Supplemental Material

This section includes all images and plots referenced above, as instructed.

Step 1: Divide data by window

Sentence (sample #)	Raw data (T# = token # in sentence)	Changes: (window size / shift)	Updated data divided by window size/shift
1	[T1, T2, T3, T4, T5, T6]	4 / 4	[T1, T2, T3, T4], [T5, T6, None, None]
2	[T1, T2, T3, T4, T5]	4 / 3	[T1, T2, T3, T4], [T4, T5, None, None]

Step 2: Build vocabularies (token : index)

```
vocabulary = dictionary {  '<pad>':      0,
                           '<unk>':      1,
                           'Burgdorf':   2,
                           'had':        3,
                           'brought':    4,
                           'a':          5,
                           ...           ,
                           'Snowman':    100841,
                           'Arminians':  100842  }

label vocabulary = dictionary {  '<pad>':  0,
                                 'PER':    1,
                                 'O':      2,
                                 'ORG':    3,
                                 'LOC':    4  }
```

Step 3: Encode data

Type	Data (Divided with window size / shift: 4 / 4)	Encoded data to corresponding index
Input	["The", "weather", "was", "amazing"] , ["today", ".", None, None] , ["Barack", "Obama", "was", "elected"], [".", None , None, None]	[307, 62, 10, 255] , [26099, 12, 0, 0] , [17281, 17282, 86, 4304], [12, 0 , 0, 0]
Output	["O", "O", "O", "O"] , ["O", "O", None, None] , ["PER", "PER", "O", "O"] , ["O", None, None, None]	[2, 2, 2, 2] , [2, 2, 0, 0] , [1, 1, 2, 2], [2, 0 , 0, 0]

Step 4: Load data in batches

Batch # (size = 2)	Input	Output
1	[[307, 62, 10, 255] , [26099, 12, 0, 0]]	[[2, 2, 2, 2] , [2, 2, 0, 0]]
2	[[12, 0 , 0, 0] , [17281, 17282, 86, 4304]]	[[2, 0 , 0, 0] , [1, 1, 2, 2]]

Step 6: Build pre-trained Sense embeddings

```
tensor ( [
[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],   <unk>
[ 0.0000,  0.0000, -0.0000,  ...,  0.0000, -0.0000,  0.0000],   <pad>
[ 0.9787,  0.6246, -0.7219,  ..., -0.4724,  0.0810, -0.8786],   token 1
    ⋮        ⋮        ⋮              ⋮        ⋮        ⋮
[-0.1693, -1.3692,  2.0089,  ..., -2.3439, -0.7039, -1.4282],   token N-2
[ 0.1748, -1.1250, -2.0812,  ...,  0.8125,  0.4747, -0.0035],   token N-1
[ 0.0707,  2.5915, -1.1100,  ...,  2.0221, -0.9284, -0.7874] ] )  token N
```

Figure 1: Diagrams show a summary of the pre-processing steps.

Step 1: Tag tokens with its sense (Optional)

Previous	['The', 'Croats', 'settled', 'there', 'in', 'the', 'early', '7th', 'century', '.']
Rules for tagging	- Tokens that are not capitalized and not the first word in the sentence are tagged (to leave its sense for NER tags that are dependent to capitalization) - Tokens that have at least one synset
Synset Selection	'settled' → POS: VBD / verb / v lemma: 'settle' synsets: [Synset('settle.v.01'), Synset('decide.v.02'), ... Synset('settle.v.21'), Synset('fall.v.32')] Chosen synset: a) most common: Synset('settle.v.01') b) word sense disambiguated given sentence and list of synsets above: Synset('settle.v.11') c) word sense disambiguated given sentence: Synset('settle.v.11') Wordnet ID: 'wn:01988458v' Babelnet ID: 'bn:00093535v' new token: 'settle_bn:00093535v'
Tagged Tokens with BabelNet IDs	['The', 'Croats', 'settled_bn:00093535v', 'there_bn:00114427r', 'in', 'the', 'early_bn:00101935a', '7th_bn:00096074a', 'century_bn:00017234n', '.']

Step 2: Divide data by window

Sentence (sample #)	Raw data (T# = token # in sentence)	Changes: (window size / shift)	Updated data divided by window size/shift
1	[T1, T2, T3, T4, T5, T6]	4 / 4	[T1, T2, T3, T4], [T5, T6, None, None]
2	[T1, T2, T3, T4, T5]	4 / 3	[T1, T2, T3, T4], [T4, T5, None, None]

Step 3: Build vocabularies (token : index)

```
vocabulary = dictionary {  '<pad>':                  0,
                           '<unk>':                  1,
                           'Burgdorf':               2,
                           'had_bn:00089240v':       3,
                           'brought_bn:00084058v':   4,
                           'a':                      5,
                           ...                       ,
                           'Willesee':               58312,
                           'Badon'                   58313  }

label vocabulary = dictionary {  '<pad>':  0,
                                 'PER':    1,
                                 'O':      2,
                                 'ORG':    3,
                                 'LOC':    4  }
```

Step 4: Encode data

Type	Data (Divided with window size / shift: 4 / 4)	Encoded data to corresponding index
Input	["The", "weather_bn:00006808n", "was_bn:00083181v", "amazing_bn:00096925a"] , ["today", ".", None, None] , ["Barack", "Obama", "was_bn:00083181v", "elected_bn:00087517v"], [".", None , None, None]	[307, 6812, 86, 255] , [42344, 12, 0, 0] , [17281, 17282, 86, 4263], [12, 0 , 0, 0]
Output	["O", "O", "O", "O"] , ["O", "O", None, None] , ["PER", "PER", "O", "O"] , ["O", None, None, None]	[2, 2, 2, 2] , [2, 2, 0, 0] , [1, 1, 2, 2], [2, 0 , 0, 0]

Step 5: Load data in batches

Batch # (size = 2)	Input	Output
1	[[307, 6812, 86, 255] , [42344, 12, 0, 0]]	[[2, 2, 2, 2] , [2, 2, 0, 0]]
2	[[17281, 17282, 86, 4263], [12, 0 , 0, 0]]	[[2, 0 , 0, 0] , [1, 1, 2, 2]]

Figure 2: Diagrams show a summary of the pre-processing steps, which includes the optional tagging of senses to certain words. This was done by retrieving its Sense tag and a) the most common or b) word sense disambiguated (WSD) synset, if available. The wordnet offset was extracted and referenced to its BabelNet ID, since embeddings tokens are as: $lemma\_bnID$. The last step for building the pre-trained Sense embeddings is the same as Figure 1.

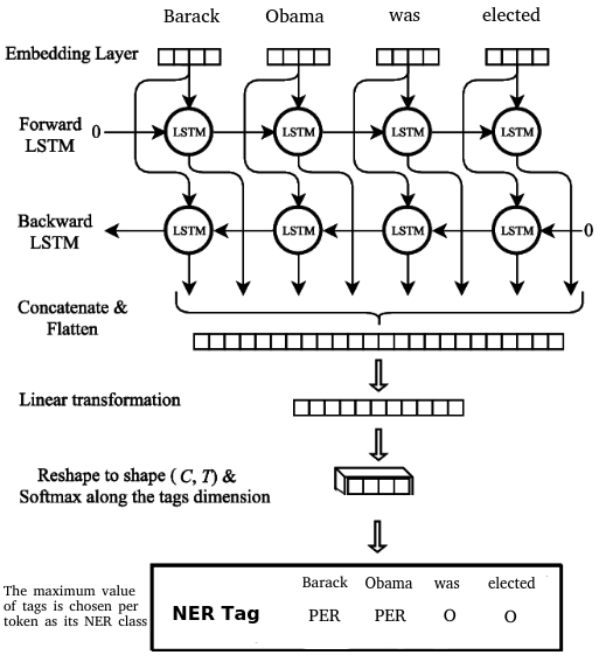

Figure 3: The diagram shows an example of the Bi-LSTM model for NER tagging. The modified diagram from (1) shows the input of tokens into an embedding layer, which has been loaded with pre-trained Sense embeddings. This is passed to the Bi-LSTM which has two stacked layers each with a forward and backward element. Dropout is added to regularize the model from overfitting and improve performance; where some 'neurons' of the Bi-LSTM are excluded from being activated and weight updated during training. A fully-connected linear layer outputs values and with Cross Entropy Loss calculates softmax for a probabilistic distribution. The highest probability of a class becomes the output prediction by the model.

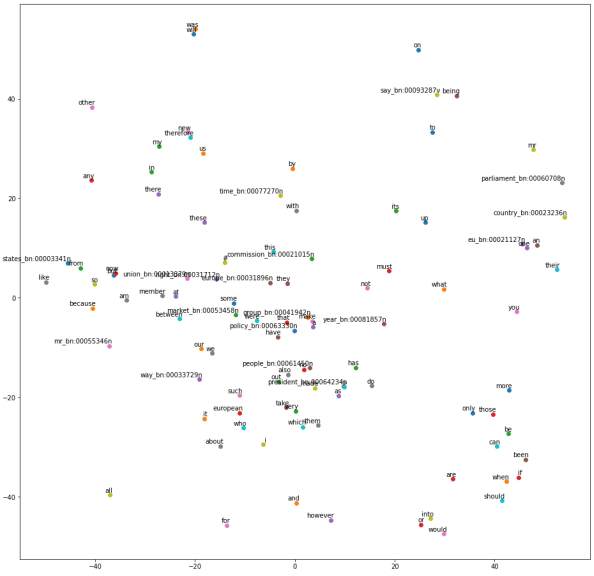

Figure 5: Diagram shows a 2D vector space of 100 words from the Sense embeddings. It was transformed from a 400 to 2 dimension vector using only 2,000 words in the vocabulary to reduce computation complexity. Words such as 'parliment', 'country', and 'eu' are found in close proximity showing correctly their similarity.

Sense Embedding Characteristics:

Type	Parameter
Dataset and Model	EuroSense High Precision dataset in English Gensim word2vec model
Tokens removed	Punctuation (except - '), replace digits with #, extra blank spaces, lowercase all words, annotations of lemmas (Sense tags) that occur < 5 times.
Training Parameters	'embedding_size': 400, 'epochs': 30, 'min_count': 3, 'negative': 30, 'window': 5
Cosine-Spearman Correlation	0.3558

Figure 4: Diagrams show a summary of the characteristics of the Sense embeddings for pre-processing, training, and accuracy.

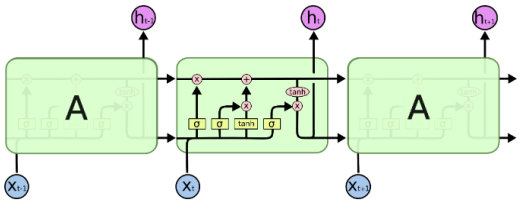

Figure 6: The diagram from (2), shows a more detailed structure of an LSTM model. The model uses some activation functions to regulate the short and long term memory (used to keep information about the most recent and oldest knowledge, each providing necessary information of a sequence. At each time step, the forward method model inputs the token and uses the past output and knowledge, then makes a prediction, and outputs its information to the next time. The backward step updates the neuron weights (only those possible) with its learned values of the future elements.

NER Model Characteristics:

Parameter	Explanation
Epochs = 20	More training, overfits the model; less doesn't reach optimal accuracy. See Figure 8.
Dropout = 0.5	More provides less structure in model (underfits); otherwise overfits to training. See Figure 9.
Vocab Freq = 1	Probably, removing more tokens also removes important context to classify better. See Figure 10.
Window size = 100 Window shift = 100	Smaller window size decreases performance, probably due to less tokens available to provide context. A different shift decreased performance. See Figure 11.
Hidden dim = 64	Increasing the hidden dimensions in the Bi-LSTM provides a more detailed model. Since the increase in performance is not significant, it is left to the second best performance. See Figure 12.
Bi-LSTM = True LSTM layers = 2	The Bi-LSTM improved the model from ~0.69 to ~0.72 (comparing results from test dataset), this improvement shows that future information (backward phase) is also important in the sequence to give context to tokens. Even though Figure 13 shows that the best amount of LSTM layers is one, this results to a overfitted model since dropout is not available. Therefore layer is increased to 2 and the dropout increased to compensate to a potential overfitted model.
Loss: Cross Entropy Optimizer: Adam	The loss function used is necessary for multiple classes. The Adam optimizer as seen in Figure 14 has a higher performance, although AdaGrad is close, since Adam includes AdaGrad and RMSProp. However since the datasets are not noisy, the RMSProp adds only a small improvement that is regardless useful.

Figure 7: The diagram shows the final parameters used to train the model and an explanation for their decision alongside Figures 8-14 that provide the justification.

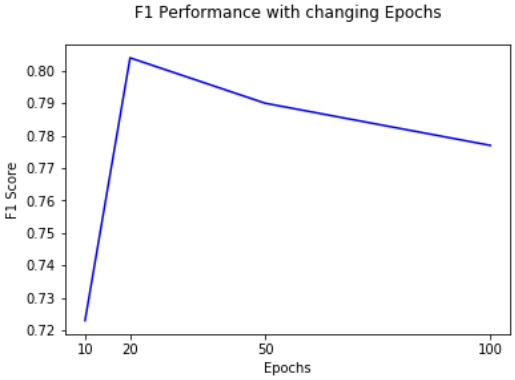

Figure 8: The diagram shows the changing F1 score with changing epochs. This shows that given the data and model, the best performance is with epoch = 20.

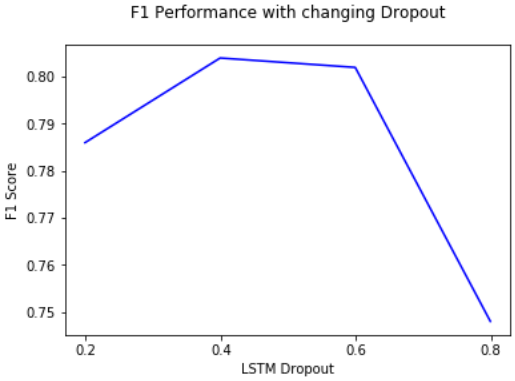

Figure 9: The diagram shows the changing F1 score with changing LSTM Dropout. This shows that given the data and model, the best performance is with dropout between 0.4 and 0.6, depending on the other final parameters.

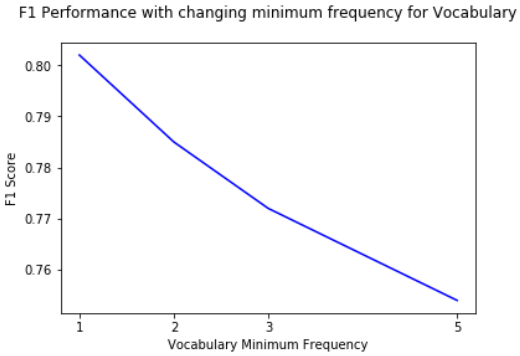

Figure 10: The diagram shows the changing F1 score with changing minimum frequency for vocabulary. This shows that given the data and model, the best performance is with $vocab\_freq = 1$.

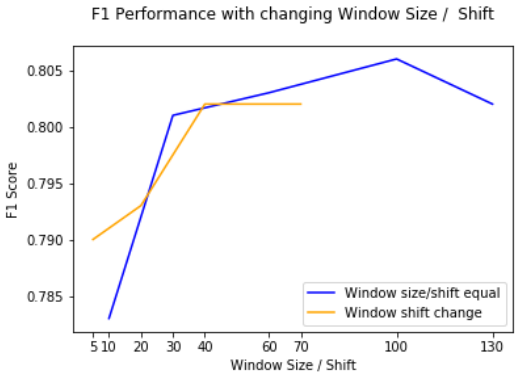

Figure 11: The diagram shows the changing F1 score with changing window size and shift. This shows that given the data and model, the best performance is with when both window size and shift are the same and equal to 100.

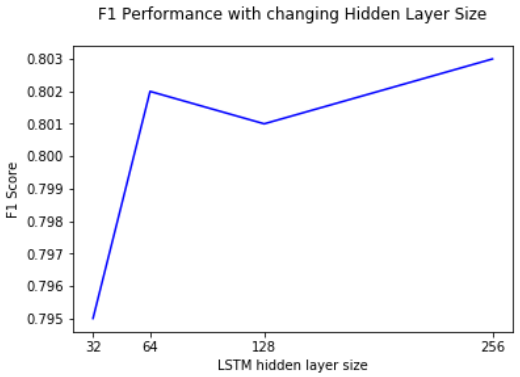

Figure 12: The diagram shows the changing F1 score with changing LSTM hidden dimension. This shows that given the data and model, hidden = 64 is chosen since the performance does not vary significantly with bigger dimension.

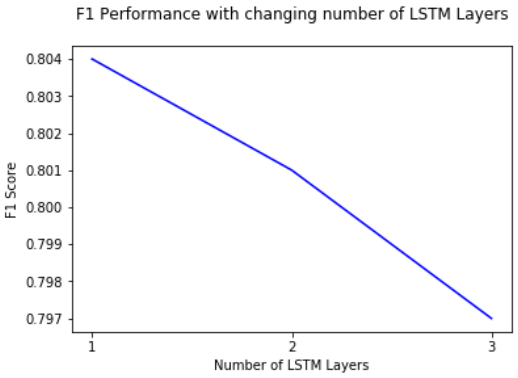

Figure 13: The diagram shows the changing F1 score with changing number of LSTM layers. This shows that given the data and model, the best performance is with LSTM_layer = 2, since one layer cannot have dropout and without this the model overfits.

NER Model Experiments:

Experiment	Explanation	F1 score
SensEmbed Bi-LSTM	Sense embedding is added to improve the understanding of words in its sentences' context to make NER classifications. This is the second largest model improvement as seen in Figure 15	0.715
SensEmbed Part-Cap Bi-LSTM	Leave all words that are capitalized as is, if they are not the first word in the sentence. The largest model improvement (Figure 15)	0.801
SensEmbed All-Cap Bi-LSTM	Improvement is seen when all tokens are left as is, without removing any capitalization	0.806
SensEmbed All-Sensetag Bi-LSTM	Words were tagged with its sense BabelNet ID (if available) from 'word' to 'lemma_bnID'. F1 score decreased, potentially due to labeling LOC instead of ORG (e.g. 'New York' vs 'New York University' gave same sense for 'New York')	0.799
SensEmbed Part-Sensetag-synCom Bi-LSTM	Given the above possible explanation, only lowercase words were tagged with their sense chosen from the most common synset	0.807
SensEmbed Part-Sensetag-synWSD Bi-LSTM	Those Sense tagged are chosen from the list of synets that are disambiguated given its sentence (WSD), since it was thought the chosen synset should be based on its sentence context	0.813
SensEmbed Part-Senstag-allWSD Bi-LSTM	Those Sense tagged are chosen from all possible disambiguated options given its sentence (WSD)	0.813
SensEmbed Part-Senstag-synCom Loss-weight Bi-LSTM	Analysis of the model's confusion matrix (CM) in Figure 21 shows that most incorrect labels were predicted as 'O', the class with most samples. To balance the dataset, weights in Figure 19 were added to the loss calculation. Although the F1 score is worse, the CM in Figure 22 shows improved balance; yet LOC/ORG remain mislabeling which is reasonable given their very similar structure	0.731

Figure 15: The table shows some of the main experiments done for the model details such as adding Sense embeddings, leaving capitalizations as is, tagging words with their Senses given its most common or WSD synset. Explanation of the reasoning behind each chosen experiment is given. Those experiments highlighted in green are the main model changes, and the orange is the final chosen model.

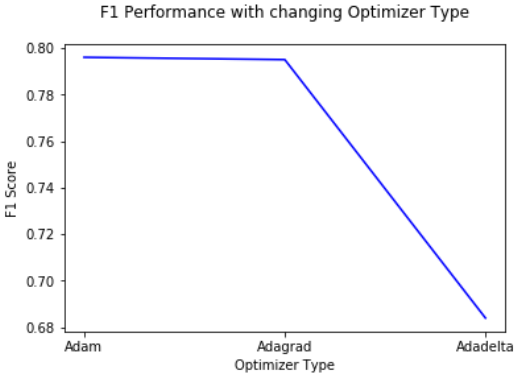

Figure 14: The diagram shows the changing F1 score with changing optimizer type. This shows that given the data and model, the best performance is with the Adam Optimizer

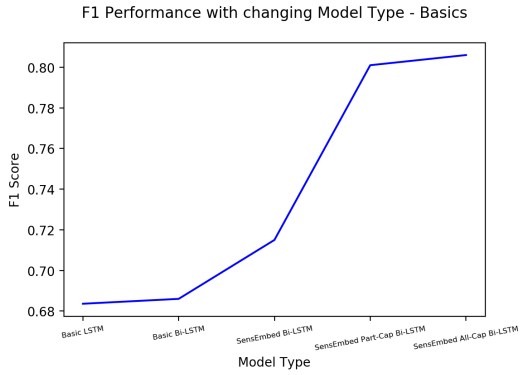

Figure 16: The diagram shows the changing F1 score with changing model type. This shows that as experiments with different models are made, there is an increase in performance. The biggest improvement in the sequence of models is when the data is not altered to remove capitalization. The second best improvement is when Sense embeddings are added to the model.

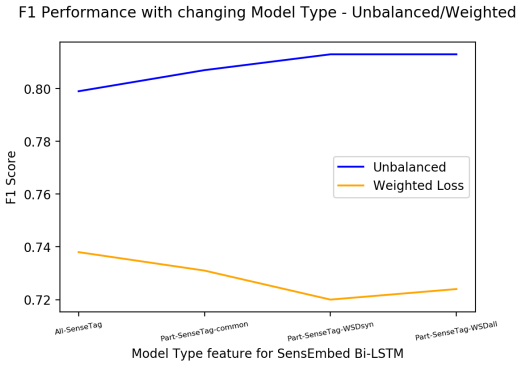

Figure 17: The diagram shows the changing F1 score with changing model type with an established Sense embedding layer Bi-LSTM base for all. Blue represents the results from the dataset which is unbalanced with more 'O' samples, while orange show the results when weights are added to the loss to balance the training. The results show that adding the weights reduce the performance of the model and the best performance is the unbalanced model with partial Sense tagging and WSD chosen synsets.

Performance Type	Train	Validation	Test
Precision:	0.876	0.825	0.834
Recall:	0.847	0.782	0.788
F1:	0.858	0.799	0.806
F1 per Class			
O:	0.991	0.985	0.985
PER:	0.919	0.824	0.831
LOC:	0.822	0.755	0.771
ORG:	0.700	0.631	0.636

Figure 19: The table shows the performances for the chosen NER model. The final F1 score for the test dataset is 0.806, which is close to the best value at 1 where both recall and precision are perfect. Recall is the main issue with 0.788 meaning there are lower percentage of correct relevant results. The class that has the worst F1 score is 'ORG', as expected since this class can contain words that usually on their own would be labelled as any other class, making it the hardest to learn (e.g. Expected: 'University of New York' $\rightarrow$ all 'ORG'. Predicted : 'University' $\rightarrow$ 'ORG', 'of' $\rightarrow$ 'O', 'New York' $\rightarrow$ both 'LOC').

NER Model Loss Weights:

Labels	Amount of Labels in Training Dataset	Calculation	Ratio
<pad>	0	O/<pad>	0
PER	100,409	O/PER	21.7
O	2,177,423	O/O	1
ORG	61,988	O/ORG	35.1
LOC	84,937	O/LOC	25.6

Figure 18: The diagram shows the weights added to the model loss, so that it can take a ratio for each label. This balances the dataset that has disproportionate amount of samples for one class compared to others.

Performance Type	Train	Validation	Test
Precision:	0.748	0.659	0.666
Recall:	0.904	0.824	0.832
F1:	0.810	0.723	0.731
F1 per Class			
O:	0.977	0.969	0.969
PER:	0.890	0.767	0.778
LOC:	0.782	0.672	0.684
ORG:	0.593	0.483	0.493

Figure 20: The table shows the performances for the weighted NER model. The final F1 score for the test dataset is 0.731, which is lower than the chosen model's. Unlike the chosen model's results, Recall is not the issue but rather Precision which means that this weighted model is not precise to choose the correct labels. The class that has the worst F1 score is still 'ORG'.

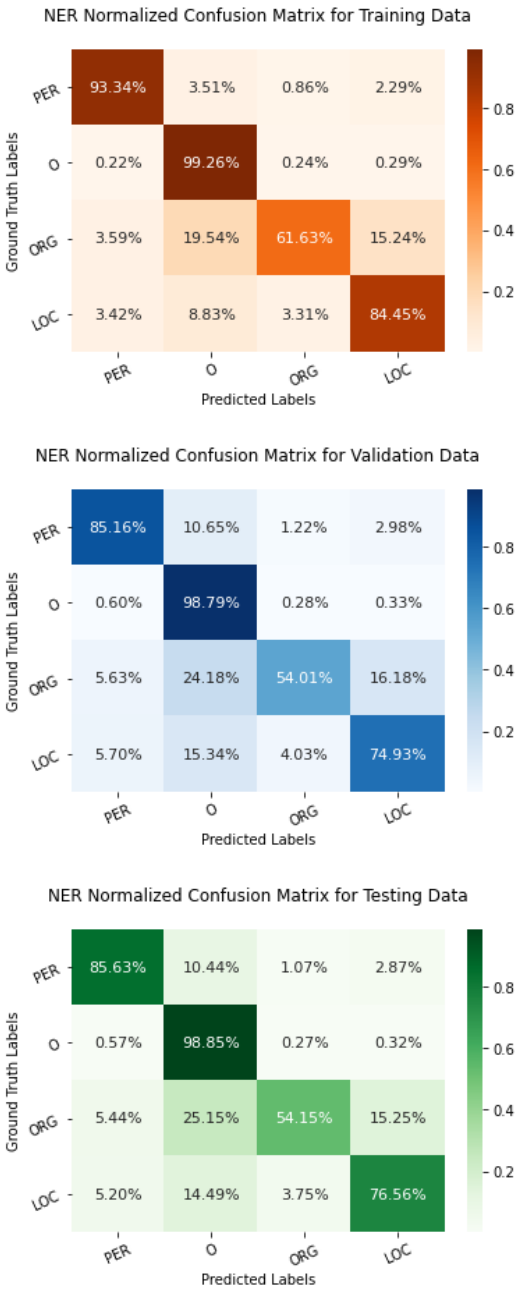

Figure 21: The diagram shows the confusion matrix as a heat map for the chosen model for all datasets (top to bottom: train, dev, test). Labels are colored darker to lighter with higher percentage of correct labeling and decreases as percentages decrease (e.g. 'O' is darkest with 99% and 'ORG' lighter with 54% for True Labels in the test data; the goal is for this main diagonal of True Labels to be as closest to 100% meaning classes labeled correctly). Most errors (55%) are labeled incorrectly as 'O'; the second highest error is mislabeling 'ORG' as 'LOC' with 15% (False 'ORG') all describing the test data. Similar behaviors are seen with all datasets.

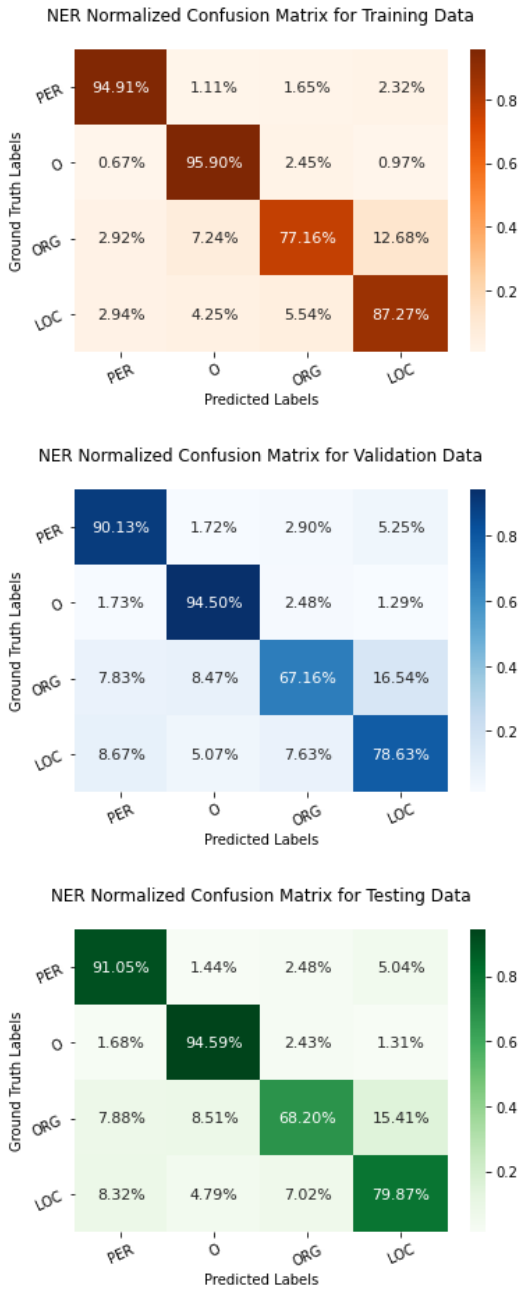

Figure 22: The diagram shows the confusion matrix as a heat map for the weighted model for all datasets. Weights were added to the loss in order to balance the datasets that had more 'O' samples than the rest. The high amount of errors seen previously for incorrect 'O' labeling is solved. However, the error of labeling incorrectly 'LOC' instead of 'ORG' (and vice-versa to a lesser extent) still occurs. Similar behaviors are seen with all datasets.

8 Citations

[1] Cornegruta, Savelie Bakewell, Robert Withey, Samuel Montana, Giovanni. (2016). Modelling Radiological Language with Bidirectional Long Short-Term Memory Networks.

17-27. 10.18653/v1/W16-6103.

[2] Olah, C. (2015, August 27). Understanding LSTM Networks. colah's blog. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3] Panchendrarajan, Rrubaa Amaresan, Aravindh. (2019). Bidirectional LSTM-CRF for Named Entity Recognition.