PROPOSING AND IMPLEMENTING A MODEL/FRAMEWORK FOR NAMED ENTITY RECOGNITION (NER)

Hari Nath Bingi
harinath0906@gmail.com

Table of Contents

Data Pre-Processing

Notwithstanding the assignment brief (optional pre-processing), we wanted to apply common best practices to text pre-processing, as this will generally help improve our model performance (HaCohen-Kerner et al, 2020).

Case Folding

First step was to normalise each word in the string with respect to *case*. This would help improve *term frequency* and *document frequency* efforts as without normalising for case the same word could be counted multiple times.

The dataset was already supplied with strings normalised to lowercase.

Manage Punctuation

We used the *re* package so that we could replace different punctuations (of multiple occurrences) with a single symbol using *regular expressions*.

Input Embedding

We explored the effectiveness of the following three features to determine their usefulness in helping our Named Entity Recognition (NER) model:

- TF-IDF.
- POS Tagging.
- Word Length.
- Character Embedding.

In this section we will describe the features and the suitability or otherwise of the features in the models' performance shall be judged in the following sections.

Word Length

During tokenization from the first assignment, we noticed some very long words and will determine if word length can be a useful feature to improve our final model.

TF-IDF

Term Frequency – Inverse Document Frequency is used to score each term with its corresponding frequency in a document corpus. We modelled our code on that introduced to us in Lab01.

For each document in the corpus, we calculated the frequency of each term. We then offset this with the overall frequency of the term in the entire corpus – the theory being that we want to score words that are frequent within a document, but that if the same word appears many times in the entire corpus it should be scored lower (t = term, d = document, D = corpus):

$$\mathrm{tfidf}(\mathrm{t}, \mathrm{d}, \mathrm{D}) = \mathrm{tf}(\mathrm{t}, \mathrm{d}) \times \mathrm{idf}(\mathrm{t}, \mathrm{D})$$

POS Tagging

Part-Of-speech Tagging is used to classify each word by its function – the way it works in the sentence. It is important to note that a word can be tagged differently according to the way it is used in a sentence.

There are various means by which to achieve this:

- Syntactic Word Classification – words belonging to Open (nouns, adjectives, adverbs, verbs) and Closed (determiners, conjunctions, pronouns) classes.
- Modern English Tag Sets – such as those available from Penn Treebank, Brown Corpus, and C7.
- Hidden Markov Model (HMM) – use HMM-based probability to determine the best tag to assign to a word.
- Viterbi Algorithm – builds on the HMM to calculate probabilities among all tag sequences, not just a single tag sequence of HMM. Viterbi Algorithm enables an effective way to perform this.

We used *PerceptronTagger* from *nltk.* It has good reviews (Honnibal, 2013), is highly accurate, and very stable.

Each of the three datasets (train, validation, test) was passed through the *PerceptronTagger* and each word in the corresponding datasets indexed and tagged with a POS tag as a feature for consideration as part of *Input Embeddings* model.

Character Embedding

Character Embedding is useful to successfully predict words which are out of vocabulary.

We created a dictionary of valid characters from our word list and assigned a one-hot vector for each character.

This vector was then used as an input feature along with other Input Embeddings to assist in our NER model.

Model

We chose to use a pre-trained word-embedding model from Gensim for two reasons:

1. Gensim has a much larger corpus of words upon which it is trained, compared to our corpus.
2. Based on our experience in the Labs, the time taken to train our own model would make it impractical for us to perform rigorous testing of different model designs and input parameters.

We considered six different Gensim datasets:

- *glove-twitter-25.*
- *glove-twitter-50.*
- *glove-wiki-gigaword-100.*
- *glove-wiki-gigaword-200.*
- *glove-wiki-gigaword-300.*
- *word2vec-google-news-300.*

Among these datasets, we rejected *word2vec-google-news-300* as it had the largest amount of missing words when compared to our corpus:

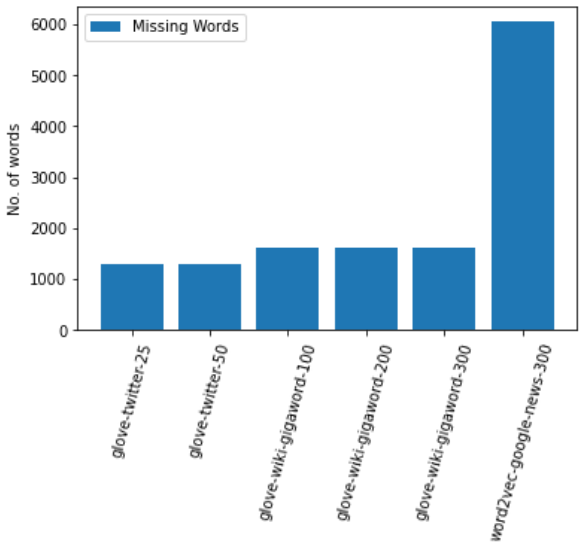

Figure 1 - Comparison of Missing Word Count for each Gensim dataset

We used *glove-wiki-gigaword-100* for the following reasons:

1. The frequency of missing words was quite low, and
2. The size of the embedding (100) was relatively high.

From various lab exercises throughout the Semester, we learned that a higher dimension for input embeddings would normally give a better efficacy.

To reduce missingness from the datasets, we considered the common base form of words by using *lemmatisation* and *stemming* packages from *nltk*:

- *WordNetLemmatizer.*
- *PorterStemmer.*
- *SnowballStemmer.*

NER Model

As suggested in the assignment brief, we started with the Bi-LSTM CRF model as our Baseline.

We designed our model to save the weights only if there was an improvement in performance (F1 Score), here is a sample output from our best performing model:

```
Info : model f1-score Improved from -inf to 0.94560614
Epoch:1, Training loss: 8292.86, val loss: 1110.14, train f1: 0.9518, val f1: 0.9456,
time: 256.99s
Info : model f1-score Improved from 0.94560614 to 0.95434092
Epoch:2, Training loss: 4047.80, val loss: 923.60, train f1: 0.9668, val f1: 0.9543,
time: 253.38s
Info : model f1-score Improved from 0.95434092 to 0.95698782
Epoch:3, Training loss: 2903.96, val loss: 881.96, train f1: 0.9737, val f1: 0.9570,
time: 253.51s
Info : model f1-score Improved from 0.95698782 to 0.96228163
Epoch:4, Training loss: 2157.68, val loss: 847.79, train f1: 0.9797, val f1: 0.9623,
time: 252.46s
Info : model f1-score Improved from 0.96228163 to 0.96360508
Epoch:5, Training loss: 1617.19, val loss: 872.26, train f1: 0.9833, val f1: 0.9636,
time: 253.81s
Info : model f1-score Improved from 0.96360508 to 0.96598729
Epoch:6, Training loss: 1253.45, val loss: 847.28, train f1: 0.9862, val f1: 0.9660,
time: 252.93s
Info : model f1-score Improved from 0.96598729 to 0.96757544
Epoch:7, Training loss: 986.05, val loss: 883.39, train f1: 0.9874, val f1: 0.9676,
time: 253.64s
Info : model f1-score Improved from 0.96757544 to 0.96929592
Epoch:8, Training loss: 775.29, val loss: 875.54, train f1: 0.9894, val f1: 0.9693,
time: 252.42s
Epoch:9, Training loss: 606.81, val loss: 900.15, train f1: 0.9927, val f1: 0.9688,
time: 254.83s
Info : model f1-score Improved from 0.96929592 to 0.97181048
Epoch:10, Training loss: 536.75, val loss: 947.48, train f1: 0.9932, val f1: 0.9718,
time: 258.76s
Info : model f1-score Improved from 0.97181048 to 0.97220752
Epoch:11, Training loss: 414.76, val loss: 942.77, train f1: 0.9964, val f1: 0.9722,
time: 254.84s
Epoch:12, Training loss: 328.93, val loss: 963.59, train f1: 0.9969, val f1: 0.9715,
time: 253.55s
Info : model f1-score Improved from 0.97220752 to 0.97300159
Epoch:13, Training loss: 281.67, val loss: 965.57, train f1: 0.9978, val f1: 0.9730,
time: 253.49s
Info : model f1-score Improved from 0.97300159 to 0.97485442
Epoch:14, Training loss: 226.36, val loss: 963.28, train f1: 0.9983, val f1: 0.9749,
time: 253.61s
Epoch:15, Training loss: 213.22, val loss: 1041.04, train f1: 0.9981, val f1: 0.9718,
time: 251.83s
```

Layers

We tested three different layer designs in pursuit of our best performing NER model:

1. Single layer Bi-LSTM.
2. Double layer Bi-LSTM.
3. Triple layer Bi-LSTM.
4. Single layer Bi-GRU.

Graves et al (2013), introduced the concept of layering as a way of adding levels of abstraction of inputs over time.

Layer Performance

As can be seen by the below Validation graphs, more layers in our Bi-LSTM model did not improve performance on our Validation dataset:

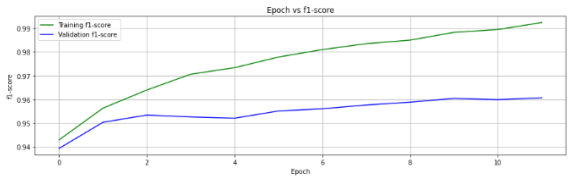

Figure 2 - Epoch vs F1 Score for Bi-LSTM Single Layer

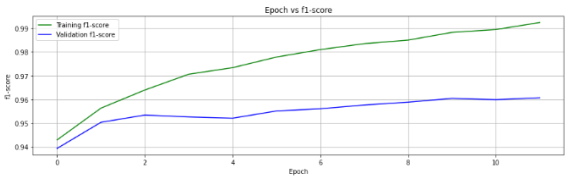

Figure 3 - Epoch vs F1 Score for Bi-LSTM Double Layer

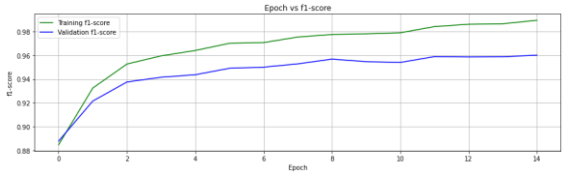

Figure 4 - Epoch vs F1 Score for Bi-LSTM Triple Layer

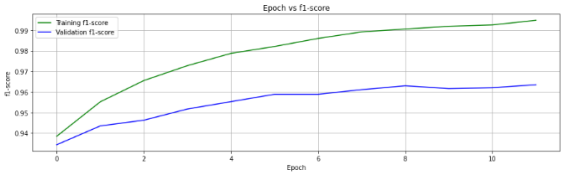

Figure 5 - Epoch vs F1 Score for Bi-GRU Single Layer

Noting the triple layer Bi-LSTM model had a greater consistency between Training and Validation datasets. Bi-GRU had a higher performing F1 Score against the Validation dataset, however, was ultimately beaten by Bi-LSTM single layer once other parameters were included (Attention, Input Embedding features, etc).

Attention

We use Attention in our LSTM models to enable a direct connection at the input to Bi-LSTM and input to the Feed Forward network to focus on a part of the Input Sequence (Content-Based, Dot-Product, Scaled Dot-Product).

Input Embeddings Attention

In pursuit of a more effective NER model, we applied Attention to the Input Embeddings of each word.

The below are the results of applying the three different Attention approaches:

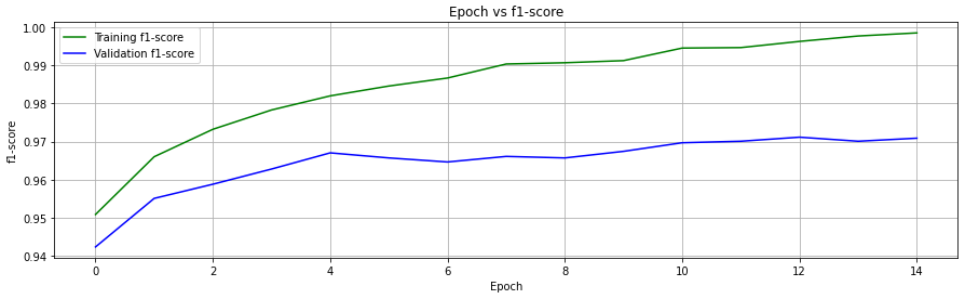

Figure 6 - Epoch vs F1 Score for Dot-Product Attention on Input Embeddings

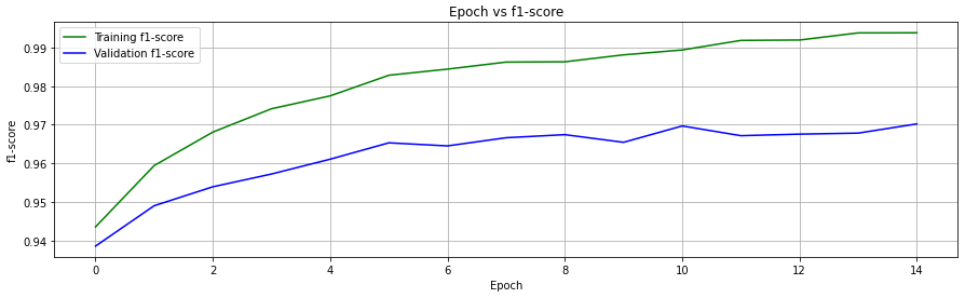

Figure 7 – Epoch vs F1 Score for Scaled Dot-Product Attention on Input Embeddings

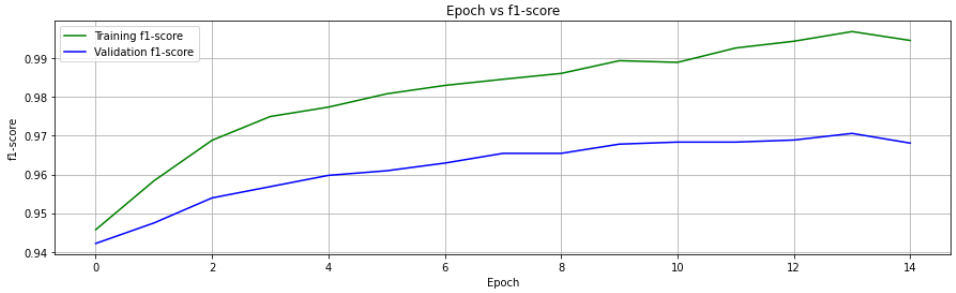

Figure 8 - Epoch vs F1 Score for Scaled Content-Based Attention on Input Embeddings

The *Dot-Product* Attention setting performed the best and became part of our model for final performance tuning.

LSTM Hidden State Attention

We also applied Attention to the Hidden States of each word from Bi-LSTM:

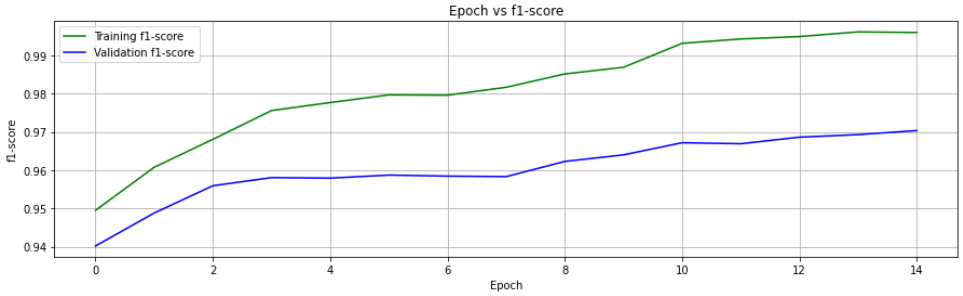

Figure 9 - Epoch vs F1 Score for Dot-Product Attention on LSTM Outputs

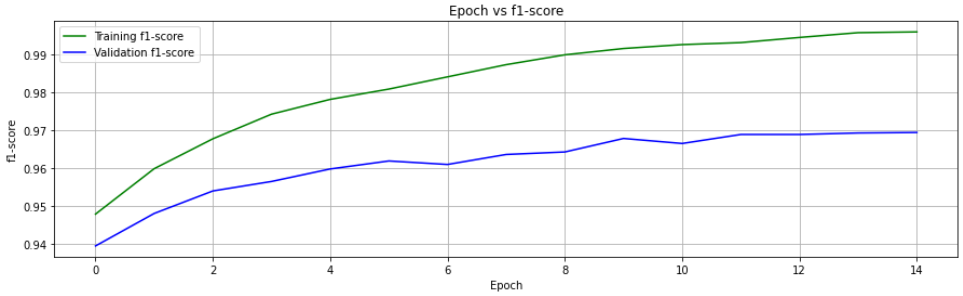

Figure 10 - Epoch vs F1 Score for Scaled Dot-Product Attention on LSTM Outputs

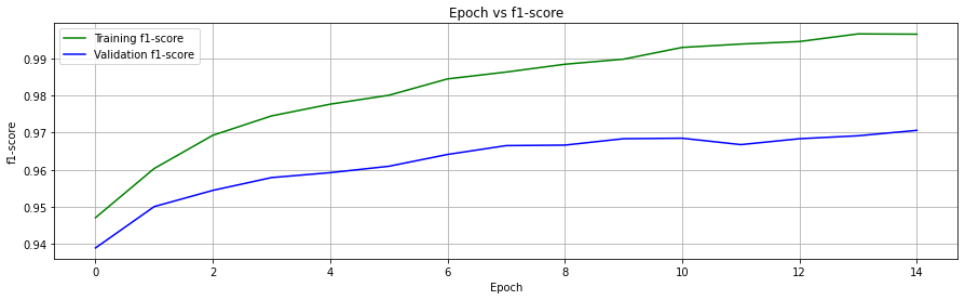

Figure 11 - Epoch vs F1 Score for Content-Based Attention on LSTM Outputs

The *Scaled Dot-Product* Attention setting exhibited the optimum consistency and so was included as part of our final NER model performance tuning.

Evaluation

Setup

We took an iterative approach to evaluation, making incremental adjustments to Input Embeddings and model design.

As per the assignment specification, the supplied Validation dataset was used for performance assessment. We chose to use the Harmonic Mean of the *Precision* and *Recall* values from the models (F1 score) as the basis for performance.

There are four areas we wanted to focus on in searching for the optimum NER model:

1. Input Embeddings: the use of various input features (TF-IDF, POS tagging, word length, character embedding).
2. Model type: the type of model used to predict NER (LSTM, Bi-LSTM, GRU).
3. Model design: the number of layers used in the model (single, double, and triple).
4. Attention: we compared content-based, dot-product, scaled dot-product in the two areas:
    a. Between input embedding of the words, and
    b. Between the Bi-LSTM hidden states of the words

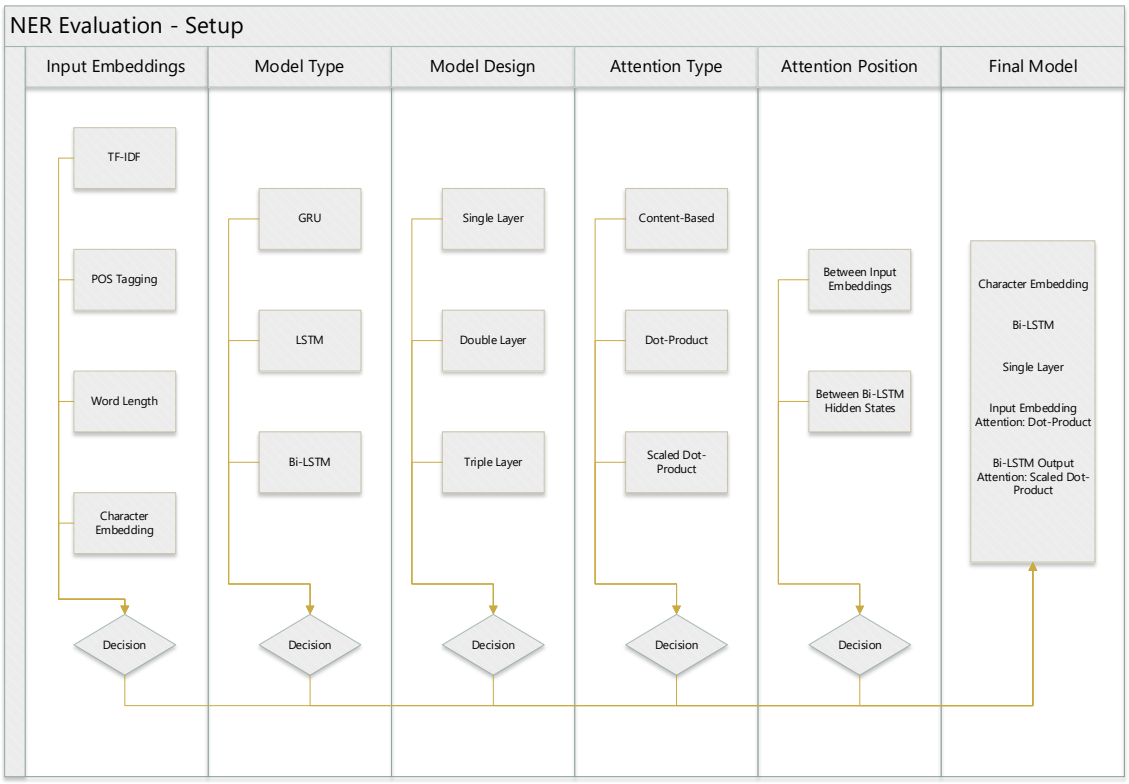

Figure 12 - Decision Process for NER Model Evaluation

Input Parameters

The following input parameters were used at various stages through model evaluation:

- Epochs: 15.
- Hidden dimension: 50.
- Word2Vec pre-trained model dataset: glove-wiki-gigaword-100.
- Character embedding hidden layers: 50.
- Character embedding number of model layers: 1.

The complete list of Parameters (including Boolean-based parameters to enable efficient coding) are included in the Appendix.

We used *Stochastic Gradient Descent* (SGD) with a learning rate of 0.01. To avoid over-fitting, we used a weight-decay value of `1e-4`. This technique provides a solution to L2 regularization (constraints when the weights are large) by adding the sum of squares of all weight coefficients to the cost (Subramanian, 2018).

Results

We performed twenty-one (21) different tests in our pursuit of the best performing NER model. The following Ablation Studies summarise the various iterations and results of these tests for each of Input Embeddings, Attention, and Layers.

It is noted that the Base Model for each study is a Bi-LSTM Word2Vec pre-trained model with one layer, without additional Input Embeddings or Attention.

Performance Comparison

The best performance came from using Character Embedding with Word2Vec as input features. Various iterations of model design were compared by experimenting with different Attention settings in the Input Embeddings, as well as the output NER model.

Finally, we compared GRU to Bi-LSTM for the NER model – with Bi-LSTM reaching the best F1 score:

Model	NER Model	F1 Score
Base model	Bi-LSTM	0.9606
Input Embedding - Dot Product **LSTM output - Scaled Dot Product**	**Bi-LSTM**	**0.9748**

Figure 13 - Performance Comparison for different NER models and input features

For a full list of all experiments conducted as part of the evaluation process please refer to the *Ablation* table in the Appendix.

Ablation Study - Embeddings

Our best performing Named Entity Recognition (NER) model was based on Bi-LSTM (Long Short-Term Memory). Here is the F1 score (Harmonic Mean) for each of the various Input Embeddings:

Model	F1 Score
Base model	0.9606
+ Word Length	0.9556
+ TF-IDF	0.9642
+ POS Tagging	0.9225
+ Character Embedding LSTM	0.9688
+ Character Embedding Bi-LSTM	**0.9719**
+ Character Embedding Bi-LSTM 2 layers	0.9712

Figure 14 - Ablation Study for Baseline Performance + various Input Embeddings

The best performance came from using Character Embeddings with Word2Vec and a single layer Bi-LSTM model (F1 score of 0.9719). It is noted that the F1 Score was based on the Validation dataset as per the assignment brief.

Ablation Study – Attention

Our best performing NER model was based on *Scaled Dot-Product* Attention. Here is the F1 score (Harmonic Mean) for each of the various attention settings:

Model	Strategy	F1 Score
Base model		0.9606
+ Attention: Input Embeddings of words	Dot-Product	0.9626
+ Attention: Input Embeddings of words	Scaled Dot-Product	0.9636
+ Attention: Input Embeddings of words	Content-Based	0.9645
+ Attention: Hidden States of words	Dot-Product	0.9711
+ Attention: Hidden States of words	Scaled-Dot Product	0.9700
+ Attention: Hidden States of words	Content-Based	0.9706
+ Attention: Input Embeddings & Hidden States of words	**Dot-Product (Input Embeddings) Scaled Dot-Product (Hidden States)**	**0.9748**

Figure 15 - Ablation Study for Baseline Performance + various Attention settings

Ablation Study – Layers

Our best performing NER model was based on the *Gated Recurrent Unit (GRU)* model with an F1 score of 0.9634. Here is the F1 score (Harmonic Mean) for each of the various attention settings:

Model	F1 Score
Base model	0.9606
+ Bi-LSTM - 2 layers	0.9575
+ Bi-LSTM - 3 layers	0.9600
+ GRU - 1 layer	**0.9634**

Figure 16 - Ablation Study for Vaseline Performance + various Layer values

However, when it came time to also consider the optimal settings for Input Embeddings and Attention, the Base Model (Bi-LSTM with single layer) was the best performing as per the Performance Comparison.

References

Graves, A., Fernández, S., & Schmidhuber, J. 2007. 'Multi-dimensional Recurrent Neural Networks', pp. 549-558. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

HaCohen-Kerner, Y., Miller, D., & Yigal, Y. 2020. 'The influence of pre-processing on text classification using a bag-of-words representation'. PLoS ONE 15(5): e0232525.

Honnibal, M. 2013. 'A Good Part-of-Speech Tagger in about 200Lines of Python'. Explosion, <https://explosion.ai/blog/part-of-speech-pos-tagger-in-python>, viewed 2 June2020.

Manning, C., Raghavan, P., Schütze, H. 2008. 'Introduction to Information Retrieval'. Cambridge University Press.

Subramanian, V. 2018. 'Deep Learning with PyTorch – A practical approach to building neural network models using PyTorch'. Ch. 4, pp. 75. Packt Publishing, Birmingham, United Kingdom.

Appendices

Code for Input Embeddings Attention

```
#Self attention among the entire input embeddings
if USE_DOTPRODUCT_ATTN4 == True:
        attn_weights = F.softmax(torch.bmm(embeds.transpose(0, 1),embeds.transpose(0, 1).trans
        pose(1, 2)),dim=-1)
        attn_output = torch.bmm(attn_weights,embeds.transpose(0, 1))
        embeds = torch.cat((embeds,attn_output.transpose(0,1)), 2)
elif USE_SCALED_DOTPRODUCT_ATTN4 == True:
        attn_weights = F.softmax(torch.bmm(embeds.transpose(0, 1),embeds.transpose(0, 1).trans
        pose(1, 2))/math.sqrt(self.lstm_in),dim=-1)
        attn_output = torch.bmm(attn_weights,embeds.transpose(0, 1))
        embeds = torch.cat((embeds,attn_output.transpose(0,1)), 2)
elif USE_CONTENTBASE_ATTN4 == True:
        cos = nn.CosineSimilarity(dim=2, eps=1e-6)
        cosinesim = []
        for x in range(len(embeds)):
                tmp= embeds[x].repeat(len(embeds),1)
                output = cos(tmp.unsqueeze(1),embeds)
                output = F.softmax(output,dim=0).transpose(0,1)
                cosinesim.append(output[0])
        cosinesim = torch.stack(tuple(cosinesim))
        attn_weights  = torch.unsqueeze(cosinesim,0)
        attn_output = torch.bmm(attn_weights,embeds.transpose(0, 1))
        embeds = torch.cat((embeds,attn_output.transpose(0,1)), 2)
```

Code for LSTM Hidden State Attention

```
#Self Attention between LSTM outputs
if USE_DOTPRODUCT_SELFATTN == True:
#attn_weights = F.softmax(torch.bmm(lstm_out.transpose(1, 2),lstm_out),dim=-1)
#attn_output = torch.bmm(lstm_out,attn_weights.transpose(1, 2))  #Transpose can be removed and
tried
#concat_output = torch.cat((attn_output, lstm_out), 1)
#concat output = concat output.view(len(sentence), self.linear in dim)
#lstm_feats = self.hidden2tag(concat_output)
        attn_weights = F.softmax(torch.bmm(lstm_out.transpose(0, 1),lstm_out.transpose(0, 1).t
        ranspose(1, 2)),dim=-1)
        attn_output = torch.bmm(attn_weights,lstm_out.transpose(0, 1))
        concat_output = torch.cat((attn_output.transpose(0,1), lstm_out), 1)

        concat_output = concat_output.view(len(sentence), self.linear_in_dim)
        lstm_feats = self.hidden2tag(concat_output)
elif USE_SCALED_DOTPRODUCT_SELFATTN == True:
        attn_weights = F.softmax(torch.bmm(lstm_out.transpose(0, 1),lstm_out.transpose(0, 1).t
        ranspose(1, 2))/math.sqrt(self.hidden_dim // 2),dim=-1)
        attn_output = torch.bmm(attn_weights,lstm_out.transpose(0, 1))
        concat_output = torch.cat((attn_output.transpose(0,1), lstm_out), 1)
        concat_output = concat_output.view(len(sentence), self.linear_in_dim)
        lstm_feats = self.hidden2tag(concat_output)
elif USE_CONTENTBASE_SELFATTN == True:
        cos = nn.CosineSimilarity(dim=2, eps=1e-6)
        cosinesim = []
        print(lstm_out.shape)
        for x in range(len(lstm_out)):
            tmp= lstm_out[x].repeat(len(lstm_out),1)
            print(tmp.shape)
            output = cos(tmp.unsqueeze(1),lstm_out)
            output = F.softmax(output,dim=0).transpose(0,1)
            cosinesim.append(output[0])
        cosinesim = torch.stack(tuple(cosinesim))
        attn_weights  = torch.unsqueeze(cosinesim,0)
        print(attn_weights.shape)
        print(lstm_out.shape)
        print(lstm_out.shape)
        attn_output = torch.bmm(attn_weights,lstm_out.transpose(0, 1))
        concat_output = torch.cat((attn_output.transpose(0,1), lstm_out), 1)
        concat_output = concat_output.view(len(sentence), self.linear_in_dim)
        lstm_feats = self.hidden2tag(concat_output)
#attn_weights = F.softmax(cos(lstm_out.transpose(1, 2),lstm_out),dim=-1)
else:
        lstm_out = lstm_out.view(len(sentence), self.linear_in_dim)
        lstm_feats = self.hidden2tag(lstm_out)  #Pass through Linear layer after LSTM
return lstm_feat
```

Code for Input Parameters

```
#Hyperparameters for training
EPOCHS = 15
HIDDEN_DIM = 50

#Dataset to be used for word2vec embeddings
GENISM_DATASET = "glove-wiki-gigaword-100"

#Input Features to be used - as many as required can be set to true. Word2Vec will be default
USETFIDF = False
USE_POSTAG =False
USE_WORDLENGTH = False
USE_CHARBASEDWORD2VEC = True

#Inputs for Character LSTM Model. Only applicable if USE_CHARBASEDWORD2VEC = True
CHAR_HIDDEN_DIM = 50 #Number of neurons in hidden layers
CHAR_BIDIRECTIONAL = True #True for Bi-LSTM else LSTM
CHAR_NUM_LAYERS = 1 #Number of LSTM layers

#The below are for applying self attention to BI-
LSTM / BIGRU of the model before linear layer. A maximum of one of two should be True.
USE_DOTPRODUCT_SELFATTN = False
USE_SCALED_DOTPRODUCT_SELFATTN = True
USE_CONTENTBASE_SELFATTN = False

#The below are for applying self attention between input embeddings.
#A maximum of one of two should be True.
USE_DOTPRODUCT_ATTN4 =True
USE_SCALED_DOTPRODUCT_ATTN4 = False
USE_CONTENTBASE_ATTN4 = False

#Inputs for the Model Build
NUM_LAYERS = 1 #1 for normal Bi-LSTM / BiGRU else stacked Bi-LSTM, stacked Bi-GRU
USE_GRU = False #If True Bi-GRU will be used, else Bi-LSTM
```

Ablations

Default Input Features	Additional Input Features	Attention (Layer and Type)	NER Model	Layers in NER	Kaggle F1	Validation F1	Action Taken	Additional Remarks	Notebook index
Default Test									
Word2Vec			BILSTM	1	0.95	0.9606		This can be treated as baseline while showing input features evaluation.	Test 4
Tried various attention at output of LSTMs									
Word2Vec		LSTM output - Dot Product	BILSTM	1	0.948	0.9626	No much difference hence did not consider.		Test 1
Word2Vec		LSTM output - Scaled Dot Product	BILSTM	1	0.949	0.9636	No much difference hence did not consider.		Test 19
Word2Vec		LSTM output - Content Base	BILSTM	1	0.9492	0.9645	No much difference hence did not consider.		Test 2
Tried various input embedding									
Word2Vec	Word Length		BILSTM	1	0.944	0.9556	No much difference hence did not consider.		Test 3
Word2Vec	TFIDF		BILSTM	1	0.949	0.9642	No much difference hence did not consider.		Test 5
Word2Vec	POSTAG		BILSTM	1	0.902	0.9225	The result indicated it is a very bad option. Hence did not consider.		Test 6
Word2Vec	Character Embedding LSTM		BILSTM	1	0.9522	0.9688	Significant Improvement and hence consider to fine tune this		Test 9
Word2Vec	Character Embedding Bi-LSTM		BILSTM	1	0.9534	0.9719	Significant Improvement after fine tuning and hence consider to use this with various types of attention.	This can be treated as baseline while showing attention layer, attention type, model type.	Test 10
Word2Vec	Character Embedding Bi-LSTM 2 layers		BILSTM	1	0.953	0.9712	Another attempt to fine tune did not yield any improvement and hence did not consider.		Test 18
Tried different types of layer for NER model									
Word2Vec			BILSTM	2	0.943	0.9575	The result indicated it is a very bad option. Hence did not consider.		Test 7
Word2Vec			BILSTM	3	0.949	0.96	The result did not improve. Hence did not consider.		Test 8
Word2Vec			BiGRU	1	0.949	0.9634	The result did not improve. Hence did not consider, but used to it test at the end with best attention and input strategies		Test 11
Tried different attention types at different places with best input features									
Word2Vec	Character Embedding Bi-LSTM	Input Embedding - Dot Product	BILSTM	1	0.954	0.9711	Slight improvement and was willing to try with different scenarios		Test 14
Word2Vec	Character Embedding Bi-LSTM	Input Embedding - Scaled Dot	BILSTM	1	0.952	0.97	No much difference hence did not consider.		Test 12
Word2Vec	Character Embedding Bi-LSTM	Input Embedding - Content Base	BILSTM	1	0.952	0.9706	No much difference hence did not consider.		Test 13
Word2Vec	Character Embedding Bi-LSTM	LSTM output - Dot Product	BILSTM	1	0.9534	0.9703	Slight improvement and was willing to try with different scenarios		Test 15
Word2Vec	Character Embedding Bi-LSTM	LSTM output - Scaled Dot Product	BILSTM	1	0.954	0.9694	No much difference hence did not consider.		Test 16
Word2Vec	Character Embedding Bi-LSTM	LSTM output - Content Base	BILSTM	1	0.952	0.9706	No much difference hence did not consider.		Test 17
Tried all best things we achieved until now									
Word2Vec	Character Embedding Bi-LSTM	Input Embedding - Dot Product LSTM output - Scaled Dot Product	BILSTM	1	0.9546	0.9748			Default
Word2Vec	Character Embedding Bi-LSTM	Input Embedding - Dot Product LSTM output - Scaled Dot Product	BiGRU	1	0.9515	0.9726			Test 20