Named Entity Recognition using Attention Mechanism

Lavanshu Agrawal[1] and Shabari Gadewar[2]

[1] University of Sydney, NSW 2006, Australia
lagr7305@uni.sydney.edu.au
[2] University of Sydney, NSW 2006, Australia
sgad8451@uni.sydney.edu.au

1 Data preprocessing

The data given was a clean data with no missing values or infinite values. It was already in lower case. The train and validation data consisted of punctuations too with their Named Entity Recognition (NER) tags, hence there wasn't a requirement of the stopwords-removal.

2 Input Embedding

An embedding is a low-dimensional space which is used to represent the high-dimensional vectors. [1] It is a mathematical representation of the text data in a vector space which also allows us to obtain semantic and syntactic meaning from the text, thus facilitating the identification of linguistic regularities. [2]

For this study, we used three features: *word embeddings (w2v), Part of Speech (PoS) tags, and word length* for creating the input embedding to be fed to the model. While developing an NER model for a language, an explicit knowledge of linguistics of the language plays a very important role in deciding the features of the model, hence, even the simple features like capitalization and length could be very powerful in recognizing the distinct semantic information between the words that otherwise seem to be similar.

Word embedding is a mathematical representation of words that replaces the otherwise one-hot encoded vectors of all words to a lower dimensional space capable of capturing the semantic information of words. This makes it very powerful to perform the Natural Language Processing (NLP) tasks.

We also used the **PoS tags** in combination with the w2v because of their ability to highlight the dependence between different words of a sentence. This would allow the model to gain additional contextual information for a given sentence, thus exhibit a better performance at classifying the NER tags. For example, if the PoS tag for a word is 'noun', then there is a high probability that the NER tag for the given word will be a 'PER' or 'ORG' tag. And our hypothesis is that it will help in helping the model to gain a better performance.

We further also added the **word length** as an additional feature. This would allow the feature matrix to gain additional information about the patterns that could be highlighted by some words.

We combined these three features to form the input embedding. The size of the input embedding matrix is (13972,63). For the word embedding *glove-twitter-25* pre-trained w2v is used which maps all the unique words to a feature space with the dimension size of 25. To form the w2v, all the data available in training, validation and test dataset were included so that the out-of-vocabulary words (which might come in test inputs) could be handled. The PoS tags were generated using the NLTK library applied over the list of unique words and it mapped the words under 37 different tags. These tags (unique tags) were thereafter enumerated and one-hot encoded before concatenating them with the w2v matrix. We also added a word length vector which added 1 further dimension to the input embedding matrix (*merged_matrix*). Hence, this resulted in an input embedding matrix of size (*unique_words*, *w2v+PoS+word_length* dimensions = **13972, 63**).

3 NER model

For this study, we built the following models:

1. Bi-LSTM CRF architecture. (Model with better prediction for competition leaderboard)

2. GRU Encoder-decoder with attention mechanism. (Final model for the scope of this project)

The advantage of the Bi-LSTM CRF architecture was that the CRF used in addition to Bi-LSTM allowed the model to take care of contextual information while predicting the NER tags. However, the computation time of the model was high and hence, the GRU model was decided as the final model for its benefits in computational time due to the lesser parameters involved. To address the lack of CRF, an encoder-decoder mechanism was used. Further details are listed below in the experimental setup (section 4.1).

The principles of the various algorithms involved in our modeling phase are as follows:

LSTM and Bi-LSTM

The Bi-LSTM model is an RNN model that is used to perform the natural language processing tasks due to their ability to learn the contextual dependencies. [3] The LSTM layers adopt a gating mechanism which also helps them to mitigate the vanishing gradient problem. The working of an LSTM layer is as follows:

1. **Forget gate:** We pass the information from the input and the previous hidden state through a sigmoid function (with the addition of bias). The idea behind this step is that the values closer to 0 will diminish (or be forgotten) and the values closer to 1 will be kept in the model. The resultant output is then pointwise multiplied with the previous cell state.
2. **Input gate:** The input and previous hidden state are passed through a sigmoid function (as in forget gate) and also separately through a tangent function (to regulate the output values). The output received from the sigmoid and tangent functions is thereafter pointwise multiplied and their resultant is then pointwise added to the cell state to yield the current cell state.
3. **Output gate:** The output gate yields the hidden state for the next timestep. The input and previous hidden state are again passed through a sigmoid function and pointwise multiplied with the resultant of the current state passed through a tangent function. This gives the hidden state of the current timestep.

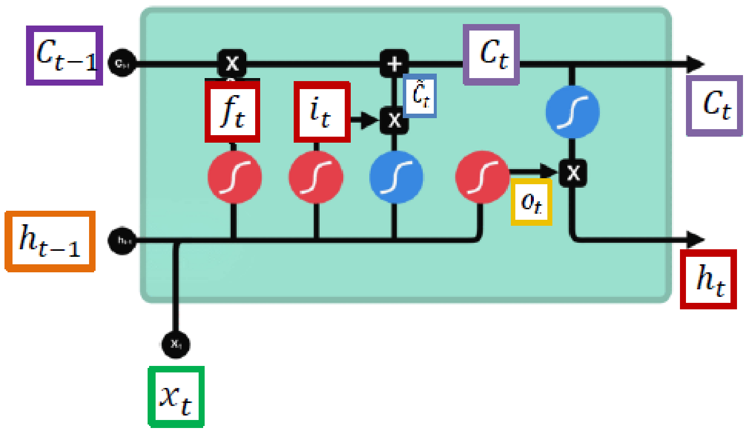

Figure 1: Components of an LSTM architecture

The Bi-LSTM model makes use of two LSTM layers working in left-to-right and right-to-left directions respectively. This enables the model to learn from the past information and future information, thus reducing the learning time for the model.

The dimensions of the input passed to the first LSTM layer is (sequence_length, embedding_size). The Forward LSTM is half the dimensions of the hidden_size specified and Backward LSTM is half of hidden size so totally the output received has number_features = hidden_size. This is due to the fact that the outputs from both the hidden layers are concatenated to result in the matrix of shape (sequence_length, hidden_size). Thereafter, the result was flattened with the use of a linear layer to result in the output_size = tagset_size. The below figure highlights the architecture of the Bi-LSTM model.

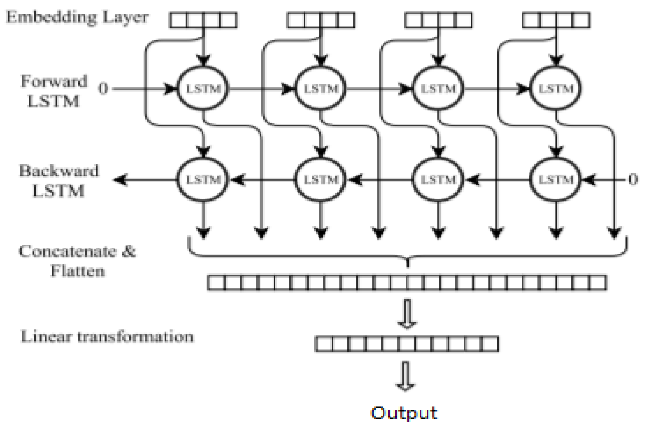

Figure 2: Working of Bi-LSTM using forward and backward LSTM layers

CRF

In addition to Bi-LSTM, we also made use of the Conditional Random Field (or CRF). It calculates the inter-dependency between the terms from the complete corpus of the text data. The idea behind CRF is to calculate the transition probabilities between the tags. Among the various methods available to do so (such as Hidden Markov Model [HMM], Maximum-Entropy Markov Model [MEMM] and CRF), the CRF was preferred due to the benefits it offers as compared to the other two models such as:

- being able to accommodate any textual information by not having a strict independence assumption as in case of HMM.
- computing the conditional probability of global optimal output nodes and therefore avoiding the biased labels problem as in case of HEMM.

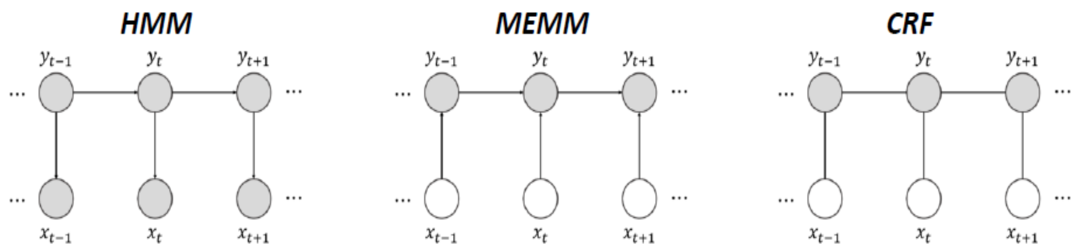

Figure 3: A view of inter-dependencies in HMM, MEMM and CRF

Since the CRF uses the global contextual information, thus it also faces a slower computation time as compared to the other two models. However, in our case, we had a small corpus, hence we used the CRF model which trained in feasible times (less than 10 minutes per epoch).

To generate the emission and transition probabilities using the CRF model, we used the **Viterbi algorithm**. The intuition behind the Viterbi algorithm is to use dynamic programming to reduce the number of computations by storing the calculations which are going to get repeated. Here we use stored values instead of calculating them again unlike brute force algorithm which calculates each path every time.

GRU

GRU with encoder-decoder mechanism: Since the output of decoder at each timestep is fed as input for the next time step, this would allow the GRU model to gain some contextual information which was formerly provided with CRF in case of Bi-LSTM. Hence, for the given NER problem, we implemented the model containing GRUs with an encoder-decoder architecture which is otherwise generally used for the question answering problem. Below figure highlights the architecture of GRU.

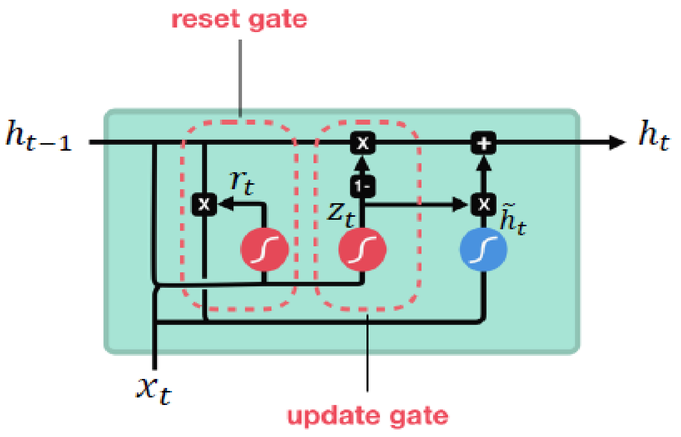

Figure 4: Components of a GRU architecture

The GRU provides an advantage over LSTM by reducing the number of training parameters and thus provides a better computational time. It also transfers the information using only hidden state and doesn't use the cell state as in the case of LSTM. [6] The GRU employs an update gate which works quite the same to the logic of the forget gate and input gate. It undertakes the input vector and the previous hidden state to generate the new hidden state by making values close to zero to forget them and close to one to keep them. GRU also employs a reset gate, which is simply used to forget some of the past information, for instance, a value of zero or close to zero, would allow the model to forget the past information. [7]

Attention mechanism

The attention mechanism was developed to facilitate a model to be able to remember long sequences of information and thus address the vanishing gradient problem. The attention establishes a shortcut connection between the context vector and the entire source input. [5] In an **encoder-decoder** architecture, the attention score at each timestep is generated by taking the corresponding dot products of the decoder hidden state with each encoder hidden state. These attention scores are thereafter passed through a softmax layer to generate the attention probabilities distribution. This attention distribution is thereafter used to generate a weighted sum of the encoder hidden states to calculate the resultant attention output. Finally, this attention output is concatenated with the decoder hidden state to predict the output from the decoder at the given timestep. [7] There are various attention scores which can be used other than the dot product [8] such as the content-based attention [9], additive attention [10] and scaled dot-product [11]. In this study, we used this architecture of attention mechanism and studied the performance over **dot-product** and **scaled dot-product** of the attention scores. The scaled dot-product scales the dot-product attention score by a factor of $1/\sqrt{n}$ which mitigates the problem of small gradients exhibited by the softmax function in cases of large input. However, in the current situation, since the input size is not large, we opted for dot-product attention scores since they converged faster than the latter, hence offering an advantage of computational time.

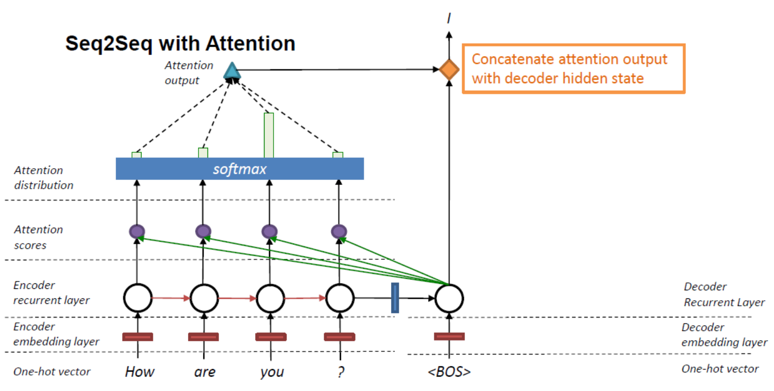

Figure 5: Attention mechanism using encoder-decoder architecture

Note: As an alternative approach to implement attention, we could also have adopted the **self-attention** wherein the different positions of a single sentence are related together to compute a representation. This enables to learn the correlations existing between the words at current timestep and those at the previous timesteps. For this, the hidden states corresponding to each word from the forward layer and backward layer of Bi-LSTM could be used to compute the attention scores.

4 Evaluation

4.1 Evaluation setup

Two different architectures were built for the study:

1. Bi-LSTM CRF.
2. GRU encoder-decoder with attention mechanism.

For Bi-LSTM CRF, we first built the model using only w2v as input embedding. The main aim of this model was to compete in the Kaggle leaderboard competition [12]. The optimal number of epochs were noted and thereafter followed in the later experiments. The Input embeddings were created using entire data available to us so that the problem of out of vocabulary word doesn't arise later. We tested the model with different combinations of features as input embedding while keeping the w2v constant in all different arrangements because of the w2v feature's high ability to provide different representations for each word. We thereafter experimented the combination of all the features with different numbers of Bi-LSTM layers in the model.

For the GRU encoder-decoder model with attention mechanism, the main aim was to build an architecture as per the scope of this study. The model was initially built using w2v as input vector and tested with different attention scores. The scaled dot product attention score was thereafter experimented with different combinations of input embeddings while keeping w2v as constant in all the combinations. The best performing input embedding along with the scaled dot product attention scores was then experimented with different numbers of layers of GRU in the encoder-decoder architecture.

Further, in the latter model, the decoder's architecture is built (for language modelling) such that the currently predicted output at a particular timestep goes as input for the next timestep and it stops when it sees the <End of Sentence (EOS)> tag. Due to this, the model predicted a varying number of output tags for a given input sequence. To overcome this limitation, we trained the model to a high number of epochs as well as we made the length of predicted tags equal to the length of the words passed as input. For this, we made the length of the input sequence as baseline, and when the model generated less number of tags, we imputed 'O' tag for the remaining places (because the 'O' tag is contained by over 80% of the dataset); and when the model generated more number of tags, we dropped the tags for the extra places.

Our input embedding size is (13972,63) where 13972 is the total number of unique words in our corpus (train data+test data + validation data) and 63 is embedding vector size.
While the number of epochs were 45, we tried out multiple epochs to find the optimal number of epochs which came out to be 45. The optimizer used was Stochastic gradient descent (SGD), Learning rate was 0.01 and Hidden dimensionality was 50.

4.2 Evaluation result

The following table compares the performances that were obtained under the different model settings tested as a part of this study:

Table 1: Performance Comparison

Model	**Features**	**Layers**	**Attention**	**Val Accuracy**
Bi-LSTM CRF	w2v	1	Not applied	0.9521

Bi-LSTM CRF	w2v+PoS	1	Not applied	0.9583
Bi-LSTM CRF	w2v+PoS+Length	1	Not applied	0.9477
Bi-LSTM CRF	**w2v+PoS+Length**	**2**	**Not applied**	**0.9637**
Bi-LSTM CRF	w2v+PoS+Length	3	Not applied	0.9571
GRU encoder-decoder	w2v	1	Dot product	0.8711
GRU encoder-decoder	w2v	1	Scaled dot product	0.8862
GRU encoder-decoder	w2v+PoS	1	Scaled dot product	0.8826
GRU encoder-decoder	w2v+PoS+Length	1	Scaled dot product	0.8731
GRU encoder-decoder (40000 epochs)	w2v+PoS+Length	1	Scaled dot product	0.8942
GRU encoder-decoder (40000 epochs)	**w2v+PoS+Length**	**2**	**Scaled dot product**	**0.9224**
GRU encoder-decoder (40000 epochs)	w2v+PoS+Length	3	Scaled dot product	0.9179

The following table lists the different combinations of input embeddings experimented upon with both of the models:

Table 2: Ablation Study - different embedding model

Model	**Embedding**	**Epochs**	**Val accuracy**
Bi-LSTM+CRF	Word2Vec	70	0.9521
	Word2Vec+PoS	45	0.9583
	Word2Vec+Pos+Length	45	0.9477
GRU encoder-decoder with scale dot-product attention	Word2vec	48000	0.8862
	w2v+POS	48000	0.8826
	Word2vec+POS+Wordlength	48000	0.8731

The following table lists the different layers experimented upon with both of the models:

Table 3: Ablation Study - different attention strategy

Model	**Embedding**	**Layers**	**Val accuracy**
Bi-LSTM+CRF (45	W2v+PoS+Length	1	0.9477

epochs)	W2v+PoS+Length	2	0.9637
	W2v+PoS+Length	3	0.9571
GRU encoder-decoder with scale dot-product attention (40000 epochs)	W2v+PoS	1	0.8942
	W2v+PoS	2	0.9224
	W2v+PoS	3	0.9179

The following table lists the different attention scores experimented upon with GRU encoder-decoder:

Table 4: Ablation Study - different layer strategy

Model	**Embedding**	**Attention score**	**Val accuracy**
GRU encoder-decoder (48000 epochs)	W2v	Dot product	0.8711
	W2v	Scaled Dot product	0.8862

Discussion of the results

Out of the two models, Bi-LSTM CRF gave better results and was used to enter the Kaggle leaderboard competition. The settings under which the best results were obtained were: input embeddings as a combination of w2v+PoS+word length and 2 layers of the model. This model didn’t make use of the attention mechanism. The model seemed to provide such a good prediction because of the (global) contextual information gained by the CRF and due to it using the Viterbi algorithm.

After adding POS tags in our input embedding as one more feature, f1 score was improved in Kaggle leaderboard. This was an expected behavior since PoS tags reflect some contextual information between words which would also help to better predict the NER tags. We further added the word length feature and observed that there was a small decrease in its performance as compared to the previous arrangement. However, since the performance on the leaderboard didn’t affect much, we kept the three features as input embedding to further experiment with different numbers of layers which finally gave us our best performance with 2 layers.

The GRU encoder-decoder architecture was used to fully achieve the scope of the study. The highlight of this model was its usage of attention mechanism. We tried the attention score calculation (*dot product* and *scaled dot product*) among which the model achieved better performance with scaled dot product (however, it was also observed that the learning rate was slow). We thereafter tested the model with different combinations of features for the input embedding and found the model to be yielding a significant performance score with w2v and PoS features as input embedding. Different numbers of layers were thereafter tested, from which the model yielded the best performance with 2 layers. Hence, the best performance of the model was achieved using w2v+PoS as input feature, 2 layers of LSTM and using scaled dot product to calculate the attention score.

References

[1] Koehrsen, W. 2020. *Neural Network Embeddings Explained*. [online] Medium. Available at: <https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526> [Accessed 6 June 2020].

[2] Mikolov, T., Yih, W.T. & Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies* (pp. 746-751).

[3] Singh, D. (May, 2020). ‘*Why is RNN used for machine translation?*’ [online] Quora. Available at: <https://www.quora.com/Why-is-RNN-used-for-machine-

translation#:~:text=They%20are%20good%20at%20learning,is%20shared%20between%20all%20timesteps.> [Accessed 8 June 2020].

[4] Panchendrarajan, R. & Amaresan, A. (2020). Bidirectional LSTM-CRF for Named Entity Recognition. In *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*.

[5] Weng, L., 2020. *Attention? Attention!*. [online] Lil'Log. Available at: <https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#whats-wrong-with-seq2seq-model> [Accessed 6 June 2020].

[6] Phi, M. (2020). ‘*Illustrated Guide to LSTM's and GRU's: A step by step explanation’*. [online] Youtube. Available at: <https://www.youtube.com/watch?v=8HyCNIVRbSU> [Accessed 6 June 2020].

[7] Han, C. S. (2020). *COMP5046: Natural Language Processing. Week 3: lecture 4. Word Classification and Machine Learning 2.* PowerPoint slides. Available at: <https://canvas.sydney.edu.au/courses/21510/files/9882544/download?wrap=1>

[8] Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

[9] Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[10] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[11] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

[12] Kaggle.com. 2020. COMP5046-2020S1-A2 | Kaggle. [online] Available at: <https://www.kaggle.com/c/2020-comp5046-a2> [Accessed 8 June 2020].