Université de Genève

Méthodes empiriques en traitement du langage (METL)

34C2046

Sequence-to-sequence workflows

authors:

Azeem Arshad

Emails:

muhammad.arshad.2@etu.unige.ch

Github project repo:

https://github.com/azeemarshad97/ner_seq2seq_project.git

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Département d'informatique

Contents

1 Timeline

Table 1: Project Timeline

4/05	▪ Project debut: deciding the prospects of each proposed project. ▪ Final choice: *Named Entity Recognition with CoNLL2003*.
11/05	▪ studying the CoNLL-2003 dataset and the pre-neural baseline tool *crfsuite*. ▪ Implementation of the latter and the neural baseline with Keras..
18/05	▪ Proposing improvements of the neural baseline by modifying its hyperparameters: adding extra layers.
25/05	▪ Abandoning the idea of only adding a crf layer, and instead going for a BiLSTM model (shows more potential). ▪ Adding character level information to it, and also adding a crf layer to it too..
1/06	▪ Implementation of the improvements mentioned above. ▪ cleaning the code and plotting the figures and results. ▪ starting to write the report..
8/06	▪ Final code touches. ▪ finishing up the report. ▪ cleaning the project overall..

2 Introduction

Named Entity Recognition (NER) is an NLP task that involves the identification and classification of named entities within textual data. This final project focuses on exploring the sequence-to-sequence processing workflow, with a particular emphasis on the transition from statistical to neural solutions. This project aims to gain a comprehensive understanding of one of the key datasets i.e. CoNLL-2003, tools, and steps involved in sequence-to-sequence NER processing.

Sequence-to-sequence processing, in the context of NER, refers to the transformation of an input sequence of words into an output sequence of corresponding entity labels. The objective is to accurately identify and classify named entities, such as persons, organizations, locations, and others, within the text. This project delves into the complexity of the sequence-to-sequence processing workflow for NER tasks. We will be using statistical and neural tools for this project.

The specific problem addressed in this project is the analysis and evaluation of the sequence-to-sequence NER workflow, focusing on the CoNLL2003 dataset. Our primary objective is to establish baseline performance using both pre-neural and neural tools and compare their effectiveness. Additionally, the project aims to propose improvements that would enhance the accuracy and efficiency of NER systems.

The motivation driving this project stems from the increasing demand for robust and accurate named entity recognition systems. NER plays a critical role in various NLP applications, including information retrieval, question answering, and knowledge extraction. By improving the effectiveness of NER systems, we can enhance information extraction, facilitate data analysis, and enable more sophisticated language understanding.

NER poses several challenges that make the development and improvement of NER models difficult. Firstly, data collection and annotation for NER tasks are expensive as entity words are relatively sparse, requiring a large number of annotated samples with varied distributions and sentence structures. Additionally, disagreements among subject matter experts regarding the correct ground truth label add to the complexity. Secondly, NER tasks involve a wide range of error types, including incorrect span boundaries, label predictions, and span hallucination, requiring thorough error analysis to identify the root causes. Furthermore, the lack of a standardized metric for benchmarking NER systems adds complexity in evaluating their performance. Semantic overlaps across classes also contribute to inconsistent annotations and challenges in defining class hierarchies. Lastly, the complexity of NER systems require more sophisticated models and longer convergence times, resulting in increased computional power cost. We thus need to address these challenges in order to improve the accuracy and efficiency of our NER models.[1]

To achieve the objectives of this project, a systematic exploration of the sequence-to-sequence NER workflow will be conducted. This includes tasks such as data acquisition and preprocessing, establishing baseline performance with a CRF model and neural tools using a simple transformer with keras. We will be analyzing system outputs, proposing improvements, and evaluating against the established statistical and neural baselines. The project report will comprehensively document the undertaken steps and findings, contributing to the broader understanding of sequence-to-sequence NER processing.

3 State of the art

Numerous attempts have been made in the past to tackle the problem of Named Entity Recognition (NER) across a variety of datasets and languages. This task is considered a crucial step in the field of information extraction and has been studied widely [2, 3].

One of the studies in NER was the CoNLL-2003 shared task [4], which employed a variety of statistical machine learning methods, such as Maximum Entropy models and Conditional Random Fields (CRFs) [5] [6]. One of our baseline model will be based on this same CRF model, with the help of the *sklearn* library, *sklearn.crfsuite*.

In recent years, the emergence of deep learning models has brought about a paradigm shift in the field. Recurrent Neural Networks (RNNs), and in particular, Long Short-Term Memory (LSTM) models, have been successfully applied to the problem [7]. Although these concepts have been invented since the 80s and end 90s respectively, the computational power we have today have enabled these methods to grow very fast today.

More recently, transformer-based models, like BERT (Bidirectional Encoder Representations from Transformers), have been used for NER tasks [**?**]. These models are pre-trained on very large corpus, before being fine-tuned for specific tasks in NLP, such as NER. The state-of-the-art performance by these pre-trained models across many language related domains with numerous data-sets, leads them to be most used models today. Additionally, the website *Huggingface* makes it much easier to access these pre-trained models.

However, many models besides these big pre-trained models, have not been thoroughly tested and analysed with the CoNLL-2003 dataset. There have been models that test BiLSTM models with different datasets, other than the CoNLL-2003. In addition, the methods and models described above have not been combined and tested extensively. For example, a model that combines the strengths of LSTM and BERT could potentially perform better on the CoNLL-2003 dataset than any individual model. For example, [8] performs NER with BiLSTM-CRF with the CoNLL-2003 dataset, and uses the *GloVe* model to embed. By using DistilBert[9], we will analyse to what extent the results can improve. We will compare to what extent a BiLSTM model with character level information can improve the results, before adding an extra CRF layer to it, and analysing the results.

The gap in the current research landscape that our project aims to fill lies in exploring the effectiveness of these existing models on the CoNLL-2003 dataset and in combining different model architectures. By customizing, configuring, and experimenting with these models, we aim to observe the different results specific models can produce on the CoNLL-2003 data-set for the task of Named Entity Recognition.

4 My approach

The primary goal of this project is to perform Named Entity Recognition (NER) on the CoNLL-2003 dataset and to compare the results with a pre-neural model (specifically, the Conditional Random Field (CRF) model) and a baseline neural model (a simple transformer).

Firstly, we propose to enhance the performance of the baseline neural model through hyperparameter tuning, before slowly incrementing the model complexity. Following that, I will implement a Bi-directional Long Short-Term Memory (BiLSTM) model with character-level information for further improvement. In order , to boost the performance, I will also implement another BiLSTM model, similar to the first one, but with an extra CRF layer.

This plan of action was taken after a certain amount of research done about NER with the CoNLL-2003 data-set.. For instance, the work of [8] served as a reference point and indluenced my approach significantly. This research outlines the application of a BiLSTM-CRF model on the same dataset.

In terms of evaluation, I will rely on Python scripts widely available and accepted in the field for calculating precision, recall, and F1 score for NER tasks. These metrics will serve as key indicators of the effectiveness of our proposed improvements. The F1 score, in particular, being the harmonic mean of precision and recall, will be our primary measure of performance improvement. Although the main goal is to study different models, Rather than trying to achieve the best performances, we will still study how one model could be better than the other.

Regarding the tools, we will evaluate using libraries from the Python ecosystem such as TensorFlow and Keras, as well as scikit-learn.

5 Data and methods

5.1 Dataset

The dataset used for this project is the CoNLL-2003 dataset. It is a well-known and widely used data-set, facilitating different models comparisons.[4]

This dataset is a collection of news wire articles from the Reuters RCV1 corpus tagged with four types of named entities: Person (PER), Organization (ORG), Location (LOC), and Miscellaneous (MISC). It consists of data in two languages: English and German, though in this project, we will focus on the English language data. The dataset is divided into training, development, and testing subsets.

5.2 Data Organization

For the baseline CRF model, the data was downloaded from a Kaggle dataset repository, as the original source did not readily provide the data.

5.3 Evaluation

We primarily use the most adequate metrics to evaluate performance with respect to NER: Precision, Recall and F1-score.

Precision (P) is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall (R), on the other hand, is the ratio of correctly predicted positive observations to all observations in actual class. The F1 Score is the weighted average of Precision and Recall. It tries to find the balance between precision and recall.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = 2 \cdot \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

where

- TP: True Positive
- FP: False Positive
- FN: False Negative

5.4 CRF Model

A Conditional Random Field (CRF) is a type of statistical modeling method often used in pattern recognition and machine learning, where they are used for structured prediction. Unlike other machine learning models, CRFs take context into account. Hence, they are particularly useful for tasks where predictions are interdependent or where context can be important.

A key aspect of CRFs is that they formulate the prediction task as a conditional probability estimation. Given a sequence of input features, a CRF estimates the probability of a sequence of output labels, conditioned on the input features. The model parameters are learned from the training data to maximize this conditional probability.

The prediction in a CRF is made based on the state sequence that has the highest conditional probability given the input sequence. This is typically computed using efficient dynamic programming algorithms like the Viterbi algorithm.

In summary, CRFs are powerful models for tasks that involve making predictions over sequences or other structured sets of inputs, particularly when context is important.

5.4.1 Data Pre-processing

The data preprocessing is divided into a few steps. Firstly, data is read from a text file in a specific format where sentences are separated by blank lines and each line is a word with its features. A sentence is then converted into a DataFrame, where each word and its associated features form a row in the DataFrame. This process is repeated for all sentences in the training and testing datasets.

In the next step, all the sentences are transformed into feature sets using the `sent2features` function. Features are extracted using the `word2features` function. For each word, a set of features is extracted, including the word itself, the last two and three characters of the word, its Part-Of-Speech tag, whether the word is uppercase, title case, digit, and the POS tags and casing information for the previous and next words if available.

Finally, all labels are extracted from the sentences. The label of each word indicates whether the word is part of a named entity and its type (Location, Person, Organization, etc.) and its position in the entity (Beginning or Inside).

5.4.2 Model description and methodology

The model used is a Conditional Random Field (CRF) from the sklearn_crfsuite library. It is a type of discriminative probabilistic model often used for predicting sequences, such as in Named Entity Recognition tasks like this one.

The CRF model is trained with the L-BFGS training algorithm and regularized with both L1 and L2 penalties (given through c_1 *and* c_2 *in the code).* The maximum number of iterations is set to 200, and the model is set to consider all possible state transitions, even those not present in the training data.

The feature sets extracted from the sentences, along with the corresponding labels, are used to train the CRF model.

5.4.3 Evaluation method

The model is evaluated using the testing data. The performance is first evaluated on individual label classes and then averaged to give macro and micro scores. The reported performance metrics include precision, recall, and F1-score.

The F1-score is a harmonic mean of precision and recall. Precision is the proportion of true positive instances among all positive predictions (i.e., the accuracy of positive predictions). Recall is the proportion of true positive instances among all actual positive instances (i.e., the model's ability to detect positive instances).

The F1-score is an overall measure of the model's accuracy that considers both precision and recall. It is particularly useful when the data distribution is uneven (i.e., when the number of instances of different classes vary significantly).

The model's overall performance is reported as the weighted average F1-score, which gives more weight to the classes with more instances. The performance on individual label classes is also reported.

5.5 Neural Baseline

5.5.1 Data Pre-processing

In this baseline, the dataset was loaded and prepared by converting it into text files containing tokens and their corresponding NER tags. Each record in the text file contained the length of the tokens, the tokens themselves, and the NER tags.

The vocabulary of the dataset was created by converting all tokens to lower case and choosing the top 20,000 most common tokens. This limited vocabulary size helps manage the computational complexity of the model. The vocabulary was then used to create a lookup layer that could convert a sequence of tokens into their corresponding IDs.

The text files were then read into TensorFlow datasets. These datasets were preprocessed by mapping each record to its tokens and tags, lowercasing and converting the tokens to their IDs using the lookup layer, and finally padding each batch.

5.5.2 Model description and methodology

The NER model built for this task is based on the Transformer model architecture, renowned for its efficiency and performance in various NLP tasks. The key component of the Transformer architecture is the self-attention mechanism. Self-attention allows the model to weigh and consider all the tokens in the input sequence when processing each individual token. It captures the dependencies between all tokens regardless of their distance in the sequence. In other words, the self-attention mechanism enables the model to focus on different words, i.e., assign different attention scores, to understand the context better while encoding each word.

In our model, the Transformer block comprises the multi-head self-attention mechanism and a feed-forward neural network (FFN). Multi-head attention allows the model to focus on different positions, effectively capturing various aspects of the sequence. Post the attention mechanism, the attention output is passed through a feed-forward neural network, which applies a transformation to each token independently.

The model's architecture consists of several layers. It begins with an embedding layer that combines token and position embeddings. This layer translates each input token into a higher dimensional space. Position embeddings are necessary to provide the model with a sense of word order as the self-attention mechanism does not have any inherent understanding of positions.

Following the embedding layer, we have a Transformer block layer which incorporates the self-attention and feed-forward neural network mechanisms. This block is then followed by a dropout layer which helps prevent overfitting.

Next, we have a dense FFN layer with a ReLU activation function that further processes the outputs. This is followed by another dropout layer and finally, a dense layer with a softmax activation function is applied to predict the NER tags.

The model is trained using a custom loss function. This function calculates the Sparse Categorical Crossentropy loss but ignores the padding tokens in the sequence while calculating the loss. This ensures that the model's performance is not unfairly penalized due to the padding tokens.

```python
class NERModel(keras.Model):
    def __init__(
        self, num_tags, vocab_size, maxlen=128, embed_dim=32, num_heads=2, ff_dim=32
    ):
        super().__init__()
        self.embedding_layer = TokenAndPositionEmbedding(maxlen, vocab_size, embed_dim)
        self.transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
        self.dropout1 = layers.Dropout(0.1)
        self.ff = layers.Dense(ff_dim, activation="relu")
        self.dropout2 = layers.Dropout(0.1)
        self.ff_final = layers.Dense(num_tags, activation="softmax")

    def call(self, inputs, training=False):
        x = self.embedding_layer(inputs)
        x = self.transformer_block(x)
        x = self.dropout1(x, training=training)
        x = self.ff(x)
        x = self.dropout2(x, training=training)
        x = self.ff_final(x)
        return x
```

5.5.3 Evaluation method

The model was evaluated on the validation dataset using the precision, recall, and F1-score metrics. These metrics were calculated for each NER tag, excluding the padding token. The scores were calculated using the conlleval.py script, which is widely used for evaluating NER models on the CoNLL dataset.

5.6 Neural baseline model improvement

THe first attempt in improving the baseline models, was to start from the neural model itself. Given that the baseline was not computationaly expensive, I decided to increase certain hyperparameters and increase certain layers.

The main difference between the two models lies in their structure and complexity. Model 1 is a more complex version, implementing multiple TransformerBlock layers determined by the `num_transformer_layers` parameter, while Model 2 utilizes a single TransformerBlock. Furthermore, Model 1's parameters `embed_dim` and `num_heads` have larger default values, enabling it to capture more intricate features. These enhancements, though, augment the model's computational requirements. Therefore, although Model 1 offers the potential to model more complex patterns, it also demands more computational resources. We were also able to observe that the overall score improvement is not worth the required computational power.

5.7 BiLSTM model with character level information

Bi-directional Long Short-Term Memory (BiLSTM) is a variant of LSTM that adds a backward pass to the original LSTM architecture, effectively increasing the amount of information available to the network. In the context of NER, BiLSTMs have proven to be highly effective.

5.7.1 Data Pre-processing

The data pre-processing involves tokenizing, aligning labels, and preparing the data-sets for the machine learning model. The ConLL2003 dataset is first loaded and a tokenizer, based on the BERT base model, is defined. Dictionaries are created in order to map the labels to tokens and vice-versa

The function `tokenize_and_align_labels` tokenizes each sentence and aligns the labels. If a token is split into multiple subtokens by the tokenizer, the function ensures that the original label is assigned to the first subtoken and the remaining subtokens receive a 'padding' label. The output of this function includes tokenized inputs, attention masks, aligned labels, and character encoded tokens. We then convert the data-sets to the TensorFlow format with batching for efficiency.

`get_char_embed()` generates character-level embeddings for each word, treating special tokens like "[CLS]", "[SEP]", "[PAD]", "[UNK]" as exceptions. Using a dictionary, we map every character to a unique integer.

A weight of 1 is assigned to every label and a weight of 0 to every padding label with the function `get_sample_weights`. This helps the model ignore the padding labels during the training. We finally prepare the inputs for the models by creating final input arrays that contain the tokenized `input_ids`, `attention masks`, `character embeddings`, `labels`, and `sample weights`.

Remark.

Word Embeddings capture the semantic meaning of words and their context within a sentence. In my case, we use the DistilBert[9] to embed the words, that is famous to be one of the most efficient models, with state-of-the-art performance.

Character Embeddings, on the other hand, are representations at the character level. They capture the morphological structure of the words and can help in understanding out-of-vocabulary words or words with similar prefixes or suffixes.

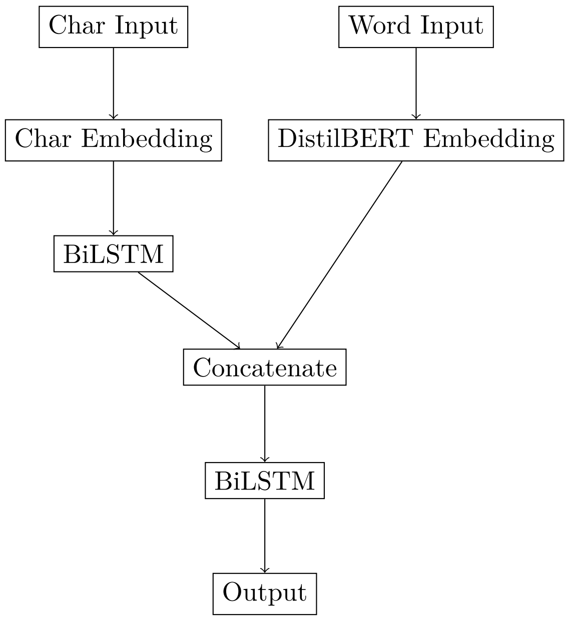
5.7.2 BiLSTM with character level information model: description and methodology

The second model introduced here is a Bi-directional Long Short Term Memory (BiLSTM) model which incorporates character-level information.

The model is structured as follows:

1. The model takes word sequences and character sequences as inputs.
2. Word sequences are encoded using the DistilBERT transformer model for word-level information.
3. Character sequences are encoded using a separate BiLSTM model for character-level information.
4. Both the word-level and character-level encodings are then concatenated together to create a joint representation.
5. This joint representation is passed through another BiLSTM layer, which is responsible for tagging the sequence.
6. Dropout regularization is applied to the BiLSTM layer's output to reduce overfitting.
7. Finally, a dense layer is applied to each time step's output with a softmax activation function, which outputs the probability distribution over the tags for each position in the sequence.

This model combines both character and word-level features, which helps the model learn both the morphological properties of the words and the contextual information in the sentence.



5.7.3 BiLSTM + character level information model + CRF Layer: description and methodology

This second BiLSTM models builds on top of our previous model, but with the addition of a CRF layer to top it. A motivation for using Conditional Random Fields (CRF), is that they can model the inter-dependencies between tags in a sequence, making them highly effective for sequence tagging tasks like NER.

5.7.4 Evaluation method

We simply use the classification report of `scikit-learn` to evaluate both models, with the usual P, R and F1-score. During training,over each epoch, we also track the model's loss using the Sparse Categorical Cross-entropy function, and the accuracy over the training **and validation sets**. We will see that using the latter improves the overall accuracy after each epoch, thus converging faster to the models' optimal possible accuracies.

6 Evaluation

We get the following results for each model:

	Precision	**Recall**	**FB1**	**Support**
LOC	82.26%	81.00%	81.62%	1809
MISC	74.94%	67.14%	70.82%	826
ORG	69.29%	57.20%	62.66%	1107
PER	69.11%	52.71%	59.81%	1405

Precision	74.70%
Recall	64.71%
F1	69.35%
Accuracy	93.28%
Accuracy non O	61.61%

Figure 1: Classification report for the neural baseline

	Precision	Recall	F1-score	Support
B-LOC	0.870	0.839	0.854	1668
B-MISC	0.800	0.748	0.773	702
B-ORG	0.802	0.723	0.761	1661
B-PER	0.829	0.853	0.841	1617
I-LOC	0.801	0.720	0.758	257
I-MISC	0.628	0.657	0.643	216
I-ORG	0.655	0.734	0.692	835
I-PER	0.867	0.947	0.905	1156
Micro avg	0.809	0.806	0.808	8112
macro avg	0.782	0.778	0.778	8112
weighted avg	0.811	0.806	0.807	8112

Table 2: CRF baseline model report

	Precision	**Recall**	**FB1**	**Support**
LOC	84.53%	83.29%	83.90%	1810
MISC	74.97%	68.55%	71.61%	843
ORG	59.93%	62.79%	61.33%	1405
PER	69.04%	61.62%	65.12%	1644

Precision	72.59%
Recall	69.66%
F1	71.09%
Accuracy	93.56%
Accuracy (non O)	66.70%

Figure 2: Classification report for the neural baseline model with improved hyper-parameters

	Precision	Recall	F1-score	Support
B-LOC	0.70	0.56	0.62	2985
B-MISC	0.50	0.38	0.43	1246
B-ORG	0.66	0.61	0.63	3479
B-PER	0.66	0.69	0.67	2974
I-LOC	0.73	0.32	0.44	413
I-MISC	0.31	0.25	0.28	319
I-ORG	0.53	0.58	0.55	1295
I-PER	0.80	0.88	0.84	2702
O	0.96	0.97	0.96	47529
accuracy			0.89	62942
macro avg	0.65	0.58	0.60	62942
weighted avg	0.88	0.89	0.89	62942

Figure 3: BiLSTM model report

	Precision	Recall	F1-score	Support
B-LOC	0.63	0.66	0.64	2985
B-MISC	0.50	0.40	0.44	1246
B-ORG	0.69	0.59	0.63	3479
B-PER	0.68	0.68	0.68	2974
I-LOC	0.55	0.43	0.48	413
I-MISC	0.35	0.26	0.30	319
I-ORG	0.56	0.56	0.56	1295
I-PER	0.79	0.87	0.83	2702
O	0.96	0.97	0.97	47529
accuracy			0.89	62942
macro avg	0.63	0.60	0.61	62942
weighted avg	0.89	0.89	0.89	62942

Figure 4: BiLSTM with CRF layer report

We will compare the the accuracy, the f1-score (with weighted average given the class imbalances), the f1-score without the 'O' tag:

	Neural Baseline (Transformer)	**CRF**	**Transformer model Improved**	**BiLSTM**	**BiLSTM + CRF**
Accuracy	0.69	0.80	0.71	0.89	0.89
F1-score Weighted Avg	0.71	0.81	0.71	0.89	0.89
F1-score w/o 'O' Tag	0.61	0.81	0.67	0.56	0.57

Table 3: Main metrics comparison for every improvement models and the baselines.

We can observe that, the BiLSTM models can achieve better results, but without the 'O' tag, they perform poorly. The CRF model is the most stable in terms of performing with or without the 'O' tag. Thus changing the parameters and better pre-processing the data for the BiLSTM+CRF model, we may be able to obtain better results.

Remark. Given that we cannot obtain the weighted average f1-score for the neural baseline, we manually compute it given the f1 scores and the support above, in order to fill the table above:

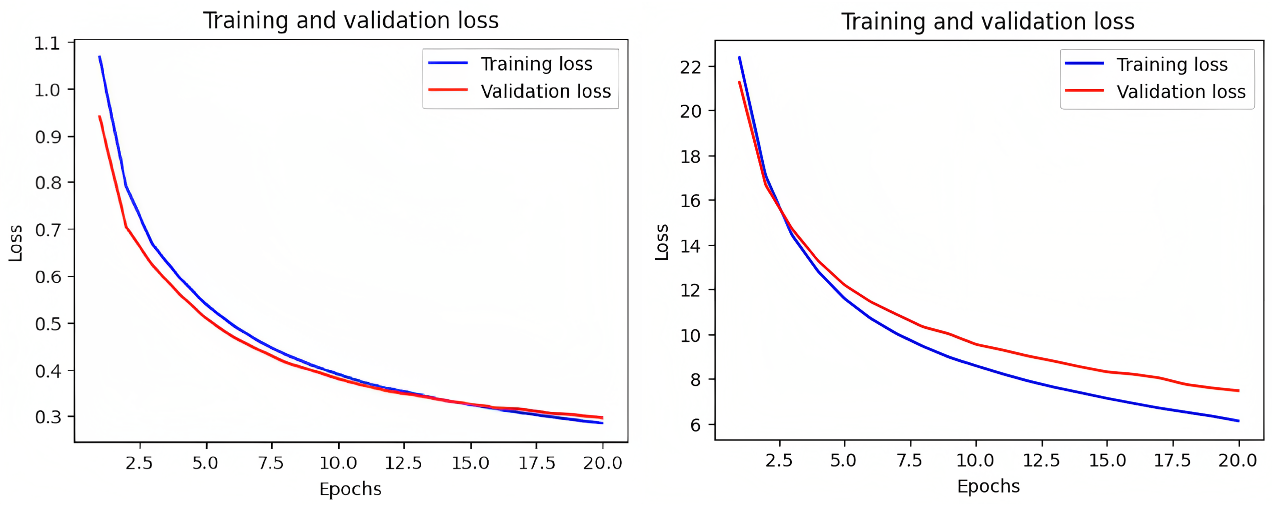
1. compute the weights, $\forall l = 1, ..., L$ labels with the support values:

$$\text{weight of l} = \frac{\text{support (l)}}{\sum_{l=1}^{L} \text{support(l)}}$$

2. multiply each F1-score with its weight and simply sum them together.

6.1 BiLSTM Models training analysis

For our main improvement, models, we can see below the training and validation losses, using the sparse categorical cross-entropy loss function. We can observe that it is more stable, and less prone to over-fitting, as



(a) BiLSTM Model (b) BiLSTM+CRF Model

Figure 5: BiLSTM models' training and validation loss over 20 epochs

compared to the BiLSTM-CRF model. With the latter model and more epochs, certain parameters would need to be changed, in order to reduce the over-fitting.

On average, BiLSTM takes on average 126 seconds while the model with an extra CRF layer takes on average 165 seconds. With the above results, we can already observe that the BiLSTM model without CRF is about 32.5% faster than BiLSTM+CRF.

7 Interpretation

I have been working on this problem and the results are interesting but also a bit mixed. The model seems to really understand the 'O' tag, predicting it correctly a lot of the time. However, when I compare my model to the one in [8], it's not performing as well. This raises a question whether their model was also proficient at 'O' tags or not.

Regarding our improved transformer model, it has become apparent that trying to improve the neural baseline model has its limits. By increasing its complexity, the computational power required increases too much to pursue its improvement even further.

Regarding the effectiveness of my solution, it could be satisfactory to some extent. The proficiency in predicting 'O' tags does boost the overall accuracy. However, if we ignore the 'O' tag, the model doesn't perform quite as well. I believe that with a better parameter initialization and more training epochs, the model could improve. Nevertheless, these models require substantial computing power. I ended up using Google Colab's GPU to expedite the training process. One potential method for improvement could be pre-processing the data to decrease the prevalence of sentences with an abundance of 'O' tags. This could enable the other classes to be more prevalent.

8 Discussion

Unexpectedly, the BiLSTM model performed comparably to the BiLSTM-CRF model, despite being less computationally intensive. Nevertheless, there's still potential for improving the BiLSTM+CRF model, such as performing a grid search of hyperparameters, utilizing dropout to mitigate overfitting, and enhancing the pre-processing of the data.

The major limitation I faced was the need for considerable computing power. To give an idea, one training epoch took about an hour on my local machine, compared to just 2 minutes per epoch on Google Colab.

Another unexpected observation was the use of the library Keras. Today, many NLP libraries of Keras are soon to be deprecated (expected date May 2024). Although aa year is still left, using quickly became a hurdle, as more attention was to be given to the code in order for the dependencies to work as Tensorflow would continue updating itself. The libraries on the other hand that enabled me to implement a CRF layer were not being updated, and the new Keras-NLP library does not currently have CRF implemented. In comparison, a much bigger community has grown in PyTorch and specific modules, such as CRF, are already being implemented.

In future work, a different approach to initializing parameters and lengthier training could help. Also, attempting to balance the data set by reducing the dominance of 'O' tags may be beneficial.

9 Conclusion

In conclusion, the research explored the application of different models for Named Entity Recognition (NER), including a baseline CRF model, a Transformer model, a BiLSTM model, and a BiLSTM model with a CRF layer. The models were evaluated using the CoNLL-2003 dataset, with performance metrics including precision, recall, and F1-score. The BiLSTM models achieved better results overall, but their performance was poor when the 'O' tag was excluded. The CRF model was the most stable in terms of performance with or without the 'O' tag. Finally, adjusting the parameters and improving the data pre-processing for the BiLSTM+CRF model, better results may be obtained.

References

[1] run galileo. What is ner and why is it’s hard to get right, 2022. May 2023.

[2] Ralph Grishman and Beth Sundheim. Message Understanding Conference- 6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996.

[3] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30, 08 2007.

[4] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition, 2003.

[5] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML ’00, page 591–598, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[6] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML ’01, page 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[8] Rrubaa Panchendrarajan and Aravindh Amaresan. Bidirectional lstm-crf for named entity recognition. 05 2019.

[9] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.