Université de Genève

Méthodes empiriques en traitement du langage

14x013

Named Entity Recognition

Author:

Petter Stahle

Email:

petter.stahle@etu.unige.ch

Project repo:

`github.com/petterstahle/ner_seq2seq_project`

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Département d'informatique

Contents

1 Introduction

It is fair to say that language has embedded within it the cumulated knowledge of human kind through history, and is the principal means through which elaboration and communication of ideas is achieved. As the bandwidth and scope of communication grows through technological innovations, from the printing press, the telegraph, radio, and now the internet to name a few, the increasing necessity to be able to make sense of and handle this explosion of information is obvious. Obvious too is the fact that if such capabilities are to scale, they must be automated.

The realm of natural language processing (NLP) is precisely oriented towards exploring and building such automated capabilities in a scientific manner. The field is vast, and the applications are numerous. One such application is the task of Named Entity Recognition (NER), which is the focus of this project.

NER is a sequence labelling task that aims to identify and classify named entities in text. On top of being extremely useful in every day business and administrative problems such as extracting key information in documents, it can also be a good tool for evaluating a language model's ability to understand the semantics of a sentence, which can be assumed to be a proxy for intelligence.

The problem of NER is not new, and has been tackled with various approaches, from rule-based systems, to statistical models, to deep learning. The latter has been very successful in recent years due to the advent of unsupervised large pre-trained language models. In this project I explore the use of such a model for NER, and compare its performance to a more traditional statistical approach, as well a simple baseline transformer model.

Lastly, with the prevailing result showcased in recent years that the best way to solve NLP tasks is to "solve language" itself, and that we can implement that knowledge in the form of an "artificial intelligence" (I use quotations as these terms have very loose definitions and carry many assumptions), I decided to explore the possibility of employing the well-known ChatGPT (GPT-3.5) as an NER model. While hard to quantitatively evaluate, the resulting converstation uncovered some surprising results and I include a discussion of these in the report.

2 State of the art

Originally defined in 1996 [1], the problem of NER has received considerable attention in the field of NLP, and researchers have made significant progress in addressing this challenge.

Early approaches were based on writing handcrafted rules, lexicons, orthographic features and ontologies. As computational capacities increased, more statistical methods could be leveraged, where feature-engineering together with machine learning became prevalent [2]. One notable method is the Conditional Random Fields (CRF) model [3]. CRF models have shown effectiveness in capturing sequential dependencies and labeling the named entities in a coherent manner. These models consider the contextual information of neighboring words and utilize features such as part-of-speech tags, word embeddings, and gazetteers to make informed predictions. By incorporating label constraints and optimizing the joint probability of the label sequence, CRF models have achieved competitive performance in NER tasks. However, one limitation of CRF models is their reliance on handcrafted features, which can be time-consuming and domain-specific to construct. Additionally, they may struggle to capture higher-level semantic information beyond local context.

First in 2011 [4] we saw the emergence of neural network systems achieving state-of-the-art results in NER [5]. Since then, similar systems using minimal feature engineering have become extremely appealing due to the fact that they require no domain specific resources like lexicons or ontologies. Various neural architectures were proposed, and a recurring trend was the use of convolutional neural networks (CNNs) with recurrent neural networks (RNNs), which are able to capture contextual information in sequential data. One notable example is the Long-Short-Term Memory with Bidirectional CRF (Bi-LSTM-CRF) [6].

In recent years, the emergence of large pre-trained language models has revolutionized the field of NLP. These architectures are essentially stacks of the transformer model first introduced in [7], which have shown remarkable results in various tasks, including NER. These models employ self-attention mechanisms to capture global dependencies, enabling them to generate representations that effectively capture context and improve entity recognition accuracy.

Despite these advancements, previous approaches to NER have often faced challenges in handling out-of-domain data and adapting to new languages or domains. Generalization remains a crucial issue, as models trained on specific domains or languages may struggle to perform well on diverse datasets. Additionally, the lack of sufficient labeled data for certain languages or domains has limited the applicability of supervised learning approaches.

To overcome these limitations, this project focuses on leveraging unsupervised pre-trained language models, particularly the BERT model, as a potential solution for NER. By fine-tuning the BERT model on the Conll2003 dataset [8], I aim to harness its powerful language representation capabilities. Additionally, I explore the possibility of utilizing ChatGPT (GPT-3.5) as a zero-shot NER model.

3 Approach

The goal of this project will be to train 2 baseline models for the NER task on the Connll2003 dataset, and improve on those results by fine-tuning the BERT model.

3.1 Baselines

The 2 baseline models are the following:

1. **CRF model trained on L-BFGS optimizer** This model was implemented using the *sklearn-crfsuite*[1] library, inspired from a basic default setup provided in the documentation.
2. **Neural Baseline** This basline is a simple Transformer model with a single transformer block, trained over 10 epochs.

Their implementations are included in the */src* directory of the repository, and their exploration and evaluation is found in the Jupyter Notebook *baseline_main.ipynb*. Specifically for the setup of the baselines, the repository was originally a fork from the repository of a colleague (Azeem Arschad) from the point of our exploration inspired from examples found on the internet. The basic structure is the same, with a few modifications added to handle the evaluation of the CRF model to handle chunking.

3.2 Fine-tuned BERT

In order to come up with an improvement, I decided to find a state-of-the-art pretrained model that is known to perform well over a large selection of NLP tasks, and use transfer-learning to optimize it to our specific task. There a many good candidates nowadays, but a classic which is open-source is BERT, which stands for Bidirectional Encoder Representations from Transformers [9]. The parameters can be downloaded from huggingface[2], and there are numerous examples of how it can be fine-tuned. Given the complexity involved and large amount of approaches possible to fine tune large language models, I believe it is important to stick to the maxim of not trying to reinvent the wheel, especially if the principal objective is to find the best model possible and that it is well known that models like BERT have state-of-the-art performance.

The overall pipeline for fitting and evaluating these models is the following:

1. Train on the connll2003 training set
2. Evaluate on the connll2003 test set by computing recall, precision and F1 score over each class

[1]https://sklearn-crfsuite.readthedocs.io/en/latest/index.html
[2]https://huggingface.co/docs/transformers/model_doc/bert

3. The overall metric we then compare is the Macro F1 score, because it reflects best the overall performance over all classes even if they are imbalanced.

I then report these results in tables and compare. It is important to note that for a truly statistically significant result, we would need to train these neural models a few times, for example in a cross-validation setup, given that neural models are well-known to have significant variance. Given time and resource constraints, I only compute for 1 instance of each model, and assume that the variance should be negligible compared to the performance difference of the Macro F1 scores. More about this will be discussed in the next section.

Finally, I also evaluate ChatGPT as a zero-shot NER model, but only in a qualitative way given the constraints of the interface. The conversation can be found in the following link: https://chat.openai.com/share/170f7649-72d7-4da7-8959-096fdce98b2b . 10 predictions based on the Connll2003 test set are reported in the */chatgpt_outputs* folder.

4 Data and methods

4.1 Dataset

As it is the benchmark for many NER tasks, the models in this project were trained and evaluated on the CoNLL-2003 dataset [8]. It is is compiled from various new articles from the Reuters corpus in English and German, with the English articles taken from August 1996 to August 1997.

4.1.1 Labels

The dataset is organized in sentences, with each token in the sentence (usually words) being labeled as on of the following 4 classes:

1. Person (PER)
2. Organization (ORG)
3. Location (LOC)
4. Miscellaneous (MISC)

With a 5th class corresponding to "Other" (O).

Furthermore, this dataset provides information about Part-of-Speech (PoS) tags, and chunking, which enables us to group tokens together which correspond to the same entity.

4.1.2 Format

As an example, here is a sentence and its corresponding labels:

Token	PoS	Chunk	NER
U.N.	NNP	I-NP	B-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	B-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	B-LOC
.	.	O	O

If the token is the first one of an entity, then the chunk "B" ("beginning") is added to it. If for example we had "United Nations" insead of "UN", we would have the first label "B-ORG", and then "I-ORG" because Nations is the second token "inside" the entity.

4.2 Organization

For each implementation, different methods are available to handle the data, and I used the simplest one available to each. For example for the Neural Baseline and Fine-tuned transformer models, there is the very useful *datasets* library which enables us to automate the processing. As for the CRF model, we retrieved the data manually by downloading it from a Kaggle repository.

The datasets are split into 3 subsets: training, validation and test, and all are stored in the */data* folder.

4.3 Evaluation

In order to evaluate the performance of the models, 3 metrics were computed over each class (we ignore the "O" class):

- **Precision**:
$$P = \frac{TP}{TP + FP}$$
- **Recal**:
$$R = \frac{TP}{TP + FN}$$
- **F1**:
$$F1 = 2\frac{PR}{P + R}$$

Where TP is the number of true positives, FP the false positives, and FN the false negative for each class.

On top of this, we aggregate the metrics into the weighted average and macro scores which are of particular interest to us when comparing the overall performance of the models, because they tell us how they handle imbalances in the labels.

The dataset used to compare the models was the test set. The validation set is also used in the neural models, as it gives an unbiased idea of the progress of models during training.

4.4 Models

4.4.1 CRF Baseline

Conditional Random Fields are statistical models that help us describe patterns in data that is contextual, i.e. where samples have intedependence, and the prediction for one is not static but changes based on neighboring samples.

In our case, we use the implementation given by the *sklearn_crfsuite* library. The main features used are handcrafted features of subtokens, for example suffixes, as well as part-of-speech tags. Other possible methods include using gazeteers, which are lists containing named entities of a specific class.

The optimization is done with gradient descent using the L-BFGS algorithm, a second-order parameter estimation method. We add a L2 and L1 penalty in the form of the $c1$ and $c2$ parameters (both set to 0.1).

The feature extraction is done in the *sent2features*, which utilizes the *word2features* for each token in the sentence. Some examples of features include the PoS tag, the word itself, whether it is upper case, and the suffix.

4.4.2 Neural Baseline

This baseline corresponds to a transformer encoder, as described in [7], implemented in Tensorflow with the Keras API. Figure 1 illustrate the architecture that we use, with $N = 1$ blocks.

In order to train the model, we first need to define the vocabulary, i.e. set of trainable tokens, which we do by selecting the 20000 most common tokens. This can then be embedded into a lookup layer that converts a sequence of tokens into their corresponding IDs. Furthermore, a token and positional embedding layer is added added at the input of the model. For the multi-head attention part, we use 10 attention heads, which correspond to the number of classes our model should predict.

For training, we optimize the model with the sparse categorical cross-entropy loss, computed between the logits (softmax outputs) of the model's prediction with the one-hot representation of the true label. This makes the output distribution converge towards a one-hot with the peak value corresponding to the most probable class.

As we can see from fig. 2, 10 epochs seem to be enough before the baseline starts to overfit, because the validation loss stagnates while the training loss keeps decreasing.

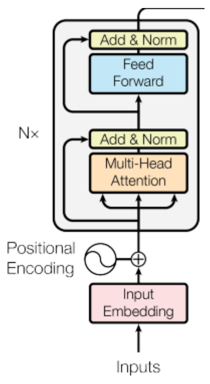

Figure 1: Transformer encoder from the Attention Is All You Need paper

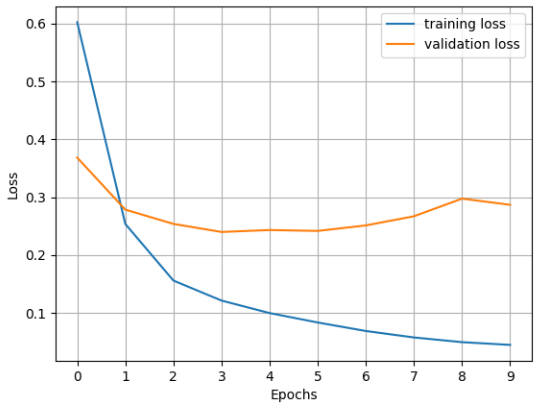

Figure 2: Neural Baseline training loss evolution

4.4.3 Fine-tuned BERT

Bert is a self-supervised pre-trained encoder model, which can be used for downstream tasks such as sequence classification by freezing all parameters except those in the final layer, and doing back-propagation on that last layer. In fig. 3 we can visualize the BERT architecture.

The main model and token embedding (tokenizer) are accessed via the huggingface API, and correspond to the "bert-base-uncased" model.

The datasets are retrieved in the same manner as for the Neural Baseline, with the *datasets* API. However, this task requires some additional pre-processing of the tokens in order to input them into BERT.

With BERT tokenization, we need to handle the 2 new tokens which are added at the beginning ([CLS]) and end ([SEP]) of a sentence. We need to make sure this embedding is aligned with the NER labels.

Additionally, the tokenizer uses sub-word tokenization, so we need to define a strategy to handle words in the dataset that are an aggregate of numerous subwords in the embedding. One strategy that I used is by setting the same label for all subwords of a word.

Furthermore, to make PyTorch ignore certain tokens in the loss computation, we can set their label to -100 which corresponds to the default *ignore_index* value.

The *tokenize_and_align_labels* function impements this processing and can be applied to the entire dataset. Here is an example of the processing of the sentence

$$['EU','rejects','German','call','to','boycott','British','lamb','.']$$

After tokenization, this becomes

$$['[CLS]','eu','rejects','german','call','to','boycott','british','lamb','.','[SEP]']$$

And we obtain the following token ids and label ids respectively:

$$[101, 7327, 19164, 2446, 2655, 2000, 17757, 2329, 12559, 1012, 102]$$

$$[-100, 3, 0, 7, 0, 0, 0, 7, 0, 0, -100]$$

Finally, the evaluation is facilitated by the *seqeval* and *evaluate* libraries, which automate the process and can be integrated into the training loop to give us a validation score.

For this project, I trained my model for 3 epochs using Google Colab GPUs, and obtained the following aggregated (Micro) metrics for each epoch:

Epoch	**Training Loss**	**Validation Loss**	**Precision**	**Recall**	**F1**	**Accuracy**
1	0.016000	0.068162	0.942761	0.939702	0.941229	0.985639
2	0.020600	0.060984	0.939307	0.948764	0.944012	0.986576
3	0.008100	0.064329	0.943488	0.950666	0.947063	0.987164

The implementation of the processing, fine-tuning and evaluation is done in the *bert_finetune.ipynb* notebook.

```
<bound method Module.state_dict of BertForTokenClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=9, bias=True)
)>
```

Figure 3: Bert Token Classifier structure

5 Evaluation

	Precision	Recall	F1	Support
B-LOC	0.870	0.839	0.854	1668
I-LOC	0.801	0.720	0.758	257
B-MISC	0.800	0.748	0.773	702
I-MISC	0.628	0.657	0.643	216
B-ORG	0.802	0.723	0.761	1661
I-ORG	0.655	0.734	0.692	835
B-PER	0.829	0.853	0.841	1617
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 1: Results for CRF baseline

	Precision	Recall	F1	Support
PER	0.857	0.904	0.880	2773
MISC	0.777	0.746	0.761	918
LOC	0.865	0.826	0.845	1925
ORG	0.766	0.747	0.756	2496
micro avg	0.823	0.820	0.821	8112
macro avg	0.816	0.806	0.811	8112
weighted avg	0.822	0.820	0.820	8112

Table 2: Results for CRF baseline with chunking removed

	Precision	Recall	F1	Support
PER	0.436	0.322	0.371	1195
MISC	0.638	0.583	0.609	641
LOC	0.750	0.740	0.745	1647
ORG	0.615	0.555	0.583	1497
micro avg	-	-	0.544	4980
macro avg	0.610	0.551	0.577	4980
weighted avg	0.620	0.564	0.589	4980

Table 3: Results for Neural Baseline

	Precision	Recall	F1	Support
PER	0.975	0.954	0.964	2718
MISC	0.759	0.758	0.977	996
LOC	0.910	0.932	0.921	2124
ORG	0.874	0.894	0.884	2588
micro avg	0.901	0.906	0.904	8426
macro avg	0.880	0.885	0.937	8426
weighted avg	0.902	0.907	0.930	8426

Table 4: Results for Fine-Tuned BERT

5.1 Bert finetuned output

Having fine-tuned the BERT model, we can observe here the outputs from an example sentence:

```
from transformers import pipeline
ner = pipeline("ner", model=model, tokenizer=tokenizer)
example = "Melissa works at the United Nations in Geneva"
ner_results = ner(example)

print(ner_results)
[{'entity': 'B-PER', 'score': 0.99661416, 'index': 1, 'word': 'melissa', 'start': 0, 'end': 7},
{'entity': 'B-ORG', 'score': 0.9990151, 'index': 5, 'word': 'united', 'start': 21, 'end': 27},
{'entity': 'I-ORG', 'score': 0.9986677, 'index': 6, 'word': 'nations', 'start': 28, 'end': 35},
{'entity': 'B-LOC', 'score': 0.99918383, 'index': 8, 'word': 'geneva', 'start': 39, 'end': 45}
```

6 Interpretation

We can clearly observe a sharp increase in the Macro F1 score, going from around 57.7% for the Neural Baseline, to 81.1% for the statistical CRF baseline, and finally 93.7% with the fine tuned model. This is not particularly surprising, given the fact that we didn't optimize the CRF model with hyperparameter tuning, or add any extra features, however I doubt that even then it would be able to beat a modern day large language model.

As for the Neural Baseline, it obtained very poor results compared to the other two models, and this is due mostly for 2 reasons, the first being that it only contains one transformer block, and thus may not have enough capacity to incorporate the necessary knowledge for the task. Secondly, it was only trained for 10 epochs, which in this case is justified by the over-fitting, but we could increase this with the capacity increase.

Aside from the quantitative analysis, we can see that the solution actually works when testing on an example, which correctly identifies the entities in a sentence.

7 Discussion

Having succeeded in the main objective of this project, it is worth mentioning that depending on the desired task, this solution may not be the best. Even though the procedure is definitely doable and not too complicated, it remains non-trivial to properly configure and fine-tune a large language model. Furthermore, if we want to achieve superb results, we quickly arrive at a point where we may require significantly more resources in terms of memory and compute to obtain just a few percentage points on the F1 score. In order to fine-tune BERT, I had to get access to a GPU such as one given by google colab, otherwise completing 1 epoch may have taken up to a few hours with my CPU.

As mentioned at the beginning of this report, I decided to test ChatGPT (version 3.5) on 10 samples of the test set. The resulting conversation[3] uncovered a very interesting feature that arises out of the nature of the GPT3 architecture, where the model outputs would completely collapse and repeat the same token ("O") without stopping, which is probably due to the fact that the next-word prediction posterior probability got saturated by previously repeating tokens. I managed to find a solution to this problem by demanding a different output format, which managed to give plausible results. Unfortunately due to the nature of the interface I was not able to test it effectively, but this could be done in a future work by using an API. In the end, this small experiment proved to me that while a model such as ChatGPT is extremely impressive in its ability to do "zero-shot" tasks just by context, it can not be "intelligent", insofar as it remains a slave of its own architecture and has no introspection of its outputs in real-time.

8 Conclusion

While statistical models remain viable and simple tools to identify named entities in text, and will probably beat any naive implementation of a neural network model, it has become obvious through this project that with current open-source large language models readily available on the internet and barely requiring half a gigabyte of memory, we can obtain significantly better results. On top of this, such models can be used for other downstream tasks by simply replacing and fine-tuning another output layer.

[3]https://chat.openai.com/share/170f7649-72d7-4da7-8959-096fdce98b2b

References

[1] 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.

[2] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. in lingvisticæ investigationes, volume 30, issue 1, p. 3 - 26, 2007.

[3] Andrew McCallum John Lafferty and Fernando C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data, 2001.

[4] Leon Bottou Michael Karlen Koray Kavukcuoglu Ronan Collobert, Jason Weston and Pavel Kuksa. Natural language processing (almost) from scratch, 2011.

[5] Vikas Yadav and Steven Bethard. A survey on recent advances in named entity recognition from deep learning models, 2019.

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

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

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

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.