Neural Network in Named Entity Recognition

Scott Fan
wf2060@nyu.edu

Siyi Zhang
wf2060@nyu.edu

Tianyue Zhang
tz2076@nyu.edu

Abstract

This paper presents neural network-based algorithms for named entity recognition (NER) that leverage deep learning techniques without the feature extraction process. By eliminating the need for manual feature engineering, our approach simplifies the overall process while potentially enhancing model performance. Our proposed algorithms incorporate Recurrent Neural Networks (RNN) and Transformers to create efficient and effective NER solutions. We also experiment with two standout models, the Flair, and BERT models, and implement specific optimizations and adjustments to improve their performances. We found that: 1) the Flair-based model performs the best with a weighted F1-score of 0.82; 2) the entity type GPE performs the best for all models; 3) neural networks have slightly better performance than the base model without manual features engineering.

1 Introduction

In our fifth homework assignment, we heavily depended on manually extracted features from input tokens and tags. However, this approach can be challenging in real-world situations, as it may not always be feasible. Consequently, we aimed to streamline the process or focus solely on input words and tags. This led us to explore the potential of neural networks for automating the feature extraction process. By employing deep learning, we can train models to learn pertinent features from the input data without the need for explicit engineering, thus simplifying the overall process and potentially enhancing performance. Neural networks are capable of detecting intricate patterns and relationships that might not be immediately obvious to human feature engineers. In the end, our goal is to create more efficient and effective natural language processing solutions by integrating neural networks into our methodology.

2 Previous Work

Many studies and experiments have been carried out for implementing neural networks, including Bidirectional LSTM-CNNs by Jason P.C. Chiu and Eric Nichols (2016), RNN by Hahusahin (Chiu and Nichols, 2016), and Iterated Dilated Convolutions by Emma, David, and Andrew (2017). Consequently, we have access to a wide variety of well-established base models for selection.

As we concentrate on Named Entity Recognition (NER) algorithms, it is important to mention that several researchers have proposed the use of embeddings to enhance model accuracy. For instance, Alan Akbik, Duncan Blythe, and Roland Vollgraf (Akbik et al., 2018) demonstrated the effectiveness of contextual string embeddings for sequence labeling tasks. Apart from our own project, we discovered an intriguing dataset called CoNLL 2003 (English), which is renowned for NER tasks. This dataset encompasses various models that can be utilized, such as LUKE (Yamada et al., 2020), ACE (Wang et al., 2021), and more.

After examining the dataset, we ultimately identified two models that stood out: the Flair model and the BERT model. By implementing specific optimizations and adjustments, we can boost the performance of these models. Besides these two models, we tried to self-implement a bi-LSTM model for comparison.

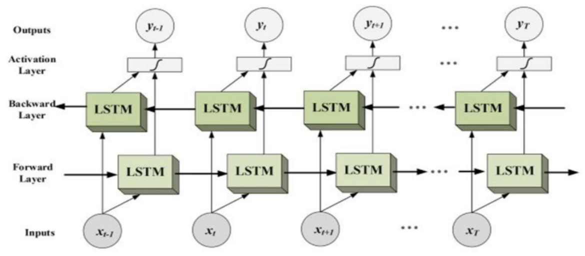

Figure 1: Word-level feature vectors are processed from the backward layer and forward layer

2.1 Bi-Directional Long-Short-Term Memory

A Bidirectional LSTM is a recurrent neural network mainly on sequence labeling made of two LSTM layers: receiving input from two directions-forward and backward, which means it reverses the information flow and increases the amount of information available. As a result, it can understand the context better and produce better performance.

2.2 Flair

Flair was developed to resolve the issue of word embeddings not being easily combined or concatenated, as the same word may require different embeddings based on its tag within the context. It is based on the Bi-LSTM-CRF model. The Flair framework introduces a unified interface for word embeddings, allowing users to create model architectures that can effortlessly embed words with any associated tags, without the need for extra embedding engineering. This approach simplifies the handling of context-dependent word embeddings in natural language processing tasks. (Akbik et al., 2019)

2.2.1 Embedding

Flair was developed to resolve the issue of word embeddings not being easily combined or concatenated, as the same word may require different embeddings based on its tag within the context. It is based on the Bi-LSTM-CRF model. The Flair framework introduces a unified interface for word embeddings, allowing users to create model architectures that can effortlessly embed words with any associated tags, without the need for extra embedding engineering. This approach simplifies the handling of context-dependent word embeddings in natural language processing tasks. (Akbik et al., 2019)

2.3 Bert

Bert is a language representation model that utilizes deep bidirectional representations to pre-train unlabeled text data. This is achieved by simultaneous conditioning on both the left and right context in all layers of the model. The advantage of this bidirectional approach is that it can overcome the limitations of unidirectional models. To achieve this, Bert employs a pre-training objective known as the ”masked language model” (MLM), which was developed based on the Cloze task(Taylor, 1953). The use of the MLM objective allows Bert to generate highly effective and context-aware language representations that can be fine-tuned for various natural language processing tasks.

3 Data Preparation

Our dataset is provided by Johan Bos in 2017. This annotated corpus for named entity recognition is based on the Groningen Meaning Bank (Bos et al., 2017) corpus and is designed to train a classifier for predicting named entities such as names, locations, and others. The dataset contains text documents that are tagged and annotated with IOB (Inside, Outside, Beginning) and POS (Part-of-Speech) tags. The IOB tagging scheme is commonly used in named entity recognition tasks to indicate the boundaries of named entities in a text, while the POS tags provide information about the grammatical structure of the text.

The dataset is feature engineered, meaning that it contains enhanced and popular features that are commonly used in natural language processing tasks. It can be loaded into a Pandas dataframe for easy handling in Python. However, in our model, we are not using the features provided by the corpus to train our model. The feature engineering part was implemented by ourselves. The corpus is particularly useful for training and evaluating named entity recognition models and can be used for various applications, such as information extraction, text mining, and sentiment analysis.

Upon reviewing the contents of the CSV files, it was found that the annotated corpus for named entity recognition consists of 47,959 individual sentences, which collectively contain 1,048,575 words, along with their corresponding POS tags and IOB tags. The dataset includes 17 types of named entities tags, namely O, B-geo, B-tim, B-org, I-per, B-per, I-org, B-gpe, I-geo, I-tim, B-art, B-eve, I-art, I-eve, B-nat, I-gpe, and I-nat.

For the validation of our system, we split this dataset into the training set and testing set at the sentence level. The testing and training ratio is 0.2:0.8. All models were trained on the training set, and the testing set was never seen until the evaluation for preventing any leakage.

4 Task Description

This system performs a Named Entity Recognition job which aims to automatically identify and extract named entities from text and classify them into one of the classes including names, location, time, etc. NER is crucial in natural language processing since it extracts structured information from unstructured text. To perform Named Entity Recognition (NER), the text is analyzed to identify entities that are present. There are different algorithms that can be used for this purpose, including rule-based systems, statistical models, and machine learning algorithms. The algorithm we are focusing on in this system is the neural network. Typically, NER models are trained on labeled datasets where each token in the text is assigned an IOB tag indicating the type of named entity it belongs to and its relative location. NER encounters various challenges when dealing with variations in spelling and structure. Additionally, there are ambiguities where a word may refer to different types of named entities depending on the context, such as a word representing a city or an organization. Despite these challenges, NER is a useful and important component in many NLP systems.

5 Our System

5.1 Evaluation

Evaluating the performance of our system, which generates IOB tags for named entity types and boundaries, requires a more comprehensive evaluation at the entity level, rather than just at the token level. Rather than self-implementing a new evaluation system, we import the "classification_report(true, pred)" from the "seqeval" module. The "classification_report" function takes two arguments: the first argument is a list of true labels and the second argument is a list of predicted labels. It calculates the precision, recall, and F1 score in micro, macro, and weighted scale for each named entity type and generates a report that displays these metrics in a tabular format. The table returned from "classification_report" help to identify areas where the model needs improvement. We can also use the report to fine-tune the model by distinguishing the types of entities the model is good at or struggles with.

5.2 Models

5.2.1 Base model(Maximum Entropy Modeling)

This model is based on the algorithm built in HW5, abbreviated Maxent. Maxent is a machine learning algorithm that is widely used in NER. The model can learn from the features which capture contextual information of surrounding words in a text sequence. Those features are commonly generated by human design which requires knowledge in natural language processing. We selected the part-of-speech tags, capitalization, last character, and whether it is punctuation, abbreviation, or preposition of a given token and information for two following and preceding tokens of it as features.

After receiving the data, the model extracts the features for the text sequence in this format: "The POS=DT previous_1_word=Beg_Stc previous_1_POS=Begin_Sent previous_1_isCap=0 Previous_BIO= previous_2_word=Beg_Stc previous_2_POS=Begin_Sent previous_2_isCap=0 next_1_wordĪraqi next_1_POS=JJ next_1_isCap=1 next_2_word=government next_2_POS=NN next_2_isCap=0 isCap=1 last_char=e isNumber=0 isPunc=0 isAbbr=0 isTheWordPrep=0 O". This is the transformed vector for one token. Each field is separated by a tab, and the first field represents the token itself, while the last field denotes the named entity IOB tags for that token. Any fields in between are presented in the format of "field_name=field_value".

The platform Maxent is built on is Java with the packed module "maxent-3.0.0.jar". Upon receiving the feature files, our system will use those files to train a model. During training, the model learns a set of weights that are used to assign probabilities to each possible label for the training set. Once the model has been trained, it can be used to predict labels for new, unseen data. During testing, the model uses the learned weights to calculate the probability of each possible label for the testing set and chooses the label with the highest probability. Finally, we use these results to produce a report evaluating how it performs.

5.2.2 Self-implemented Bi-LMST

This model is built with TensorFlow in Python. Before training on the data, we did several steps to pre-process it. We create a dictionary named "word_index" which assigns a unique integer to each word in the corpus, and a reversed dictionary, "index_word", which maps the integer indices back to the original words. Similarly, create two dictionaries for the tags, called "tag_index" and "index_tag". For instance, if we encode the sentence "Thousands of demonstrators have marched through London" with corresponding tags, the encoded sentence will be a sequence of integers based on the "word_index" dictionary, along with a list of integers based on the "tag_index" dictionary representing the tags. Besides that, the LSTM model requires fixed-length input. The sentences need to be padded to the maximal length in the corpus with the "key" of "–PADDING–". For example, the padded sentence from the previous example would have a length of nine, with two additional "–PADDING–" keys added at the end of the sentence, and the corresponding id-tags list would also be padded with zeros to match the length.

After processing the data, we build a neural network from TensorFlow for the model. The network contains 4 layers: 1) Embedding layer for semantic information; 2) bi-LSTM layer for retrieving information for preceding and subsequent tokens; 3) TimeDistributed layer for keeping track of how the model performs throughout the training stage; and 4) CRF layer for retrieving label predicted for last token in a given sequence. The optimizer we selected for this model is Adam, and the epoch and batch size we picked for the training stage is 10 and 32. Upon the trained model, we predict new labels with our testing set and evaluation how it performs with "classification_report()".

5.2.3 Flair-based

This model is built in Python with the help of the package Flair and PyTorch. To use our data as a corpus in the Flair environment, we transfer our data in the format of CoNLL-2003. CoNLL-2003 is a commonly used dataset for NER tasks. The data is in a space-separated format where each row represents a word and its corresponding features, with empty lines separating sentences. The first column contains the word itself, the second column is the part-of-speech (POS) tag, and the last column contains the named entity tag using the Inside-Outside-Beginning (IOB) format. Among the various embedding options available in Flair, we opted for the "GloVe" embedding for simplicity. GloVe is an unsupervised algorithm that generates dense vectors for tokens in the corpus, with the concept that words that appear together frequently are represented by vectors that are close together in the embedding space.

Our model is built upon a pre-trained model called "resources/taggers/example-ner", and here is the hyperparameter we selected for it: learning_rate: "0.100000", mini_batch_size: "32", patience: "3", anneal_factor: "0.5", max_epochs: "150". The neural network will keep storing the weight received from the optimization stage until it reaches a global minimum and terminates.

5.2.4 BERT-based

This model is also built in Python with packages BERT and PyTorch. In order to utilize a BERT model for entity classification of tokens, a series of data preprocessing steps are necessary. These include tokenization and label adjustment to match the tokenization process. Prior to tokenization, the tokens and corresponding tags within each sentence are concatenated into a single string to conform to the input format of the BERT tokenizer. BERT's tokenizer utilizes a sub-word tokenizer, referred to as a "word-piece" tokenizer, which splits uncommon words into one or more meaningful sub-words. For example, a person name "Geir" could be split into 'G', '##ei', and '##r'. To address the additional tokens generated by BERT's tokenizer, we realign the labels to the processed tokens by assigning the same index for split words to enable further processing.

BERT treats NER as a token-level classification task, and we employ BertForTokenClassification to accomplish this. The BertForTokenClassification class is a model that wraps BERT and adds linear layers on top of the BERT model to act as token-level classifiers. After configuring the model, the training process follows the standard PyTorch neural network process. The optimizer we have chosen is SGD, and we have selected the following hyperparameters: LEARNING_RATE = 5e-3, EPOCHS = 5, BATCH_SIZE = 2. Unfortunately, due to limitations in computing power, we have only utilized the first 10,000 sentences in the dataset. Finally, we input our testing set into the model, convert the predicted id vector back to labels, and evaluate the results using the "classification_report()" method.

6 Results

These four tables are the results of our four models generated from "classification_report()". Following from top to bottom, they are the base model, self-implemented model, Flair-based model, and BERT-based model.

	precision	**recall**	**f-1 score**	**support**
art	0.22	0.05	0.08	39
eve	0.44	0.22	0.29	32
geo	0.85	0.86	0.85	3776
gpe	0.97	0.93	0.95	1528
nat	0.38	0.25	0.30	20
org	0.65	0.69	0.67	1955
per	0.73	0.72	0.73	1645
tim	0.86	0.84	0.85	2056
micro	0.81	0.81	0.81	11051
macro	0.64	0.57	0.59	11051
weighted	0.81	0.81	0.81	11051

Table 1: Base Model

	precision	**recall**	**f-1 score**	**support**
art	0.09	0.40	0.15	10
eve	0.37	0.58	0.45	19
geo	0.89	0.81	0.85	4140
gpe	0.90	0.95	0.93	1511
nat	0.31	0.91	0.47	11
org	0.67	0.69	0.68	1895
per	0.71	0.73	0.72	1633
tim	0.77	0.86	0.81	1928
micro	0.80	0.81	0.80	11147
macro	0.59	0.574	0.63	11147
weighted	0.80	0.81	0.80	11147

Table 2: Self-Implemented Model

	precision	**recall**	**f-1 score**	**support**
art	0.75	0.13	0.23	45
eve	0.87	0.43	0.58	30
geo	0.81	0.89	0.85	3763
gpe	0.96	0.93	0.95	1588
nat	0.60	0.37	0.46	32
org	0.72	0.57	0.64	1980
per	0.77	0.78	0.78	1681
tim	0.86	0.85	0.85	2139
micro	0.82	0.81	0.82	11258
macro	0.79	0.62	0.67	11258
weighted	0.82	0.82	0.82	11258

Table 3: Flair Model

	precision	recall	f-1 score	support
art	0.00	0.00	0.00	7
eve	0.00	0.00	0.00	6
geo	0.75	0.69	0.72	761
gpe	0.86	0.74	0.80	317
nat	0.00	0.00	0.00	3
org	0.50	0.42	0.46	411
per	0.59	0.59	0.59	363
tim	0.74	0.59	0.66	444
micro	0.69	0.61	0.65	2312
macro	0.43	0.38	0.40	2312
weighted	0.69	0.61	0.65	2312

Table 4: BERT Model

Upon evaluating the models, we found that the accuracy of the base model, Self-Implemented Model, and Flair model was relatively comparable. However, BERT was a notable exception, exhibiting a significantly lower score due to runtime constraints. The BERT model could not successfully run on a large dataset because of the limitation of computing power, leading to a severe impact on its accuracy. Consequently, we incorporated the BERT model into our analysis primarily as a sample, rather than a reliable model for comparison.

Considering all the statistics, the Flair model achieved a slightly higher accuracy, precision, and F1 score on the weighted scale compared to others. The Flair model emerges as the best option because it effectively captures contextual information in text sequences through its unique embedding approach, which allows for a more accurate representation of the underlying data. Besides the weighted averages, we can also use the macro scores to see how they perform. As we can see, neural networks except BERT outperform the base model in macro average (global) which assigns an equal weight to each class, regardless of their frequency or imbalance in the dataset. Higher macro scores indicate that the performances are relatively balanced across all classes, which is consistent with our goal of simplifying the NER job with neural networks.

Additionally, tag GPE performs the best for all models, with the dominant tag GEO coming in second. The performance of GPE can be attributed to the fact that Geopolitical Entity is normally well structured, and the larger number of GEO tags in the training data allow the model to learn patterns related to it more effectively. What's more, Geographical and Geopolitical entities typically exhibit distinct and identifiable characteristics, such as capitalization or particular prefixes/suffixes (e.g., 'City of', 'Republic of'). These recognizable patterns facilitate easier and more accurate identification and classification by the model.

7 Conclusion

In conclusion, we developed three neural network models: Bi-directional LSTM, Flair, and Bert for name entity recognition that leverage deep learning techniques to automate the feature extraction process, eliminating the need for manual feature engineering. By inputting the same dataset and comparing the outputs of each model, we found that Flair has the best performance after implementing specific embedding layers. Under our evaluations, the models demonstrate their effectiveness in identifying and classifying named entities within the text, showing promising results that pave the way for future improvements and applications in sequence labeling tasks. Overall, our work shows that neural networks can be powerful tools for automating NER tasks and can lead to more efficient and effective natural language processing solutions.

8 Future Work

Despite the successes achieved in our project, there remain areas for potential improvement in the system compared to the state-of-the-art performance on the dataset CoNLL-2003, which has an average F1 score of around 93. One such area involves the dataset we employed, which contains engineered features specifically designed for NER tasks that we have yet to utilize. As a result, we cannot definitively assert that our base model has reached its full capacity. Moreover, we have not incorporated any domain-specific embeddings from Flair to enhance performance; the

”GloVe” embedding solely extracts semantic information based on co-occurrence. Additionally, due to computational constraints, we were unable to utilize the complete dataset for the BERT model. In future work, we recommend re-running the system on high-performance computing resources, such as those offered by NYU’s High-Performance Computing facility, to further optimize our model and potentially achieve even better results.

9 Contribution

In the coding stage, Scott, Siyi, and Tianyue implemented different models - bi-LSTM, BERT-based, and base/Flair-based respectively. During the writing stage, Scott, Siyi, and Tianyue divided their responsibilities among different sections. Scott handled the Previous work, Data preparation, and Result sections, Siyi was responsible for Previous work, generating tables sections, and formatting and revising the report, and Tianyue was in charge of System description, task description, future work sections, and formatting and revising the report. The Introduction, Conclusion, and abstract sections were collaboratively written.

References

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, Minneapolis, Minnesota. Association for Computational Linguistics.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Johan Bos, Valerio Basile, Kilian Evang, Noortje Venhuizen, and Johannes Bjerva. 2017. The groningen meaning bank. In Nancy Ide and James Pustejovsky, editors, *Handbook of Linguistic Annotation*, volume 2, pages 463–496. Springer.

Jason P.C. Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Hahusahin. 2021. Named entity recognition (ner) system using rnn. `https://github.com/hahusahin/ner_system_rnn`. Accessed: April 23, 2023.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Hiroki Nakayama. 2018. seqeval: A python framework for sequence labeling evaluation. Software available from https://github.com/chakki-works/seqeval.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2670–2680, Copenhagen, Denmark. Association for Computational Linguistics.

Wilson L Taylor. 1953. “cloze procedure”: A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021. Automated concatenation of embeddings for structured prediction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2643–2660, Online. Association for Computational Linguistics.

Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. 2020. LUKE: Deep contextualized entity representations with entity-aware self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6442–6454, Online. Association for Computational Linguistics.

Jie Yang, Yue Zhang, and Fei Dong. 2017. Neural reranking for named entity recognition. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP*

2017, pages 784–792, Varna, Bulgaria. INCOMA Ltd.