A SPAN-LEVEL STRATEGY FOR NESTED NAMED ENTITY RECOGNITION

A cute kid
Home-squated doesn't want to understand what is double-blind review

ABSTRACT

The idea of this paper originates from Finkel & Manning (2009), who proposed a technique for recognizing nested named entities. However, the sequence tagging models may produce false positive entities, which decrease the precision of the models. In this paper, I propose a training strategy for the NER task. With this approach, my model avoids generating far more false positive entities and achieves state-of-the-art performance on four named entity recognition datasets. Furthermore, my model obtains an F1 score of 95.70% on the GENIA and an F1 score of 93.58% on the LitBank. Likewise, my model proves its effectiveness on the flat NER task and achieves an F1 score of 98.41% on the MSRA. For the discontinuous task, my model achieves an F1 score of 88.57% on the CADEC. Moreover, I apply this training strategy to four other datasets to demonstrate the model's generalization ability across languages. Thorough experiments demonstrate that this training strategy is more effective than standard semi-CRF because it uses regression for entity information rather than generating tags by tagging each word within a sentence. The gains are particularly strong for the nested NER task corpus; for example, I'll take each piece of data in the training set containing entity tags one by one. I use the CPU inside the MacBook Pro to train the model. For the trained model, I use violin plots to show the data distribution of it, and perform difference analysis with volcano plot on it. Similarly, I use redundancy analysis on the trained model. I also show the model's hyperparameters and the overall training process, which allows my model to achieve level-headed improvement over other sequence-tagging-based approaches.

1 INTRODUCTION

Named Entity Recognition (NER) refers to identifying and classifying entities within a sentence. These entities are comprised of phrases, individual words, or characters. Named entity recognition tasks are usually categorized into three typologies, flat, nested, and discontinuous. The description

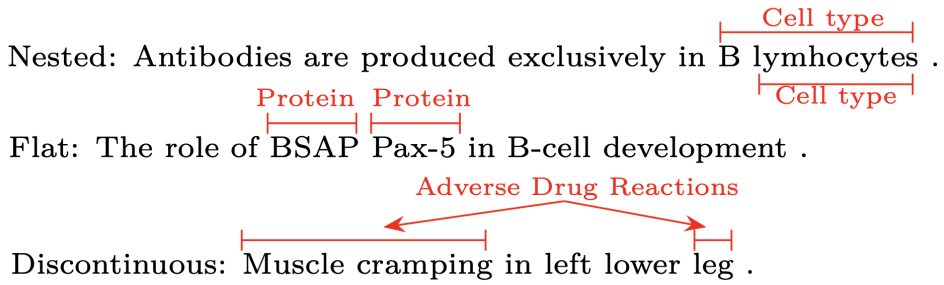

Figure 1: NER task typology

of the ner tasks is shown in Figure 1 . Most current state-of-the-art models on different datasets are using ELECTRA or generative models. For ELECTRA, no Pytorch version has been found on

the internet yet. Therefore, ELECTRA is not recommended. For generative models, it is easy to generate false positive entities to decrease the precision.

In pursuit of excellence, I propose a training strategy that can improve the model's performance. Meanwhile, this strategy can be done without wasting resources and saving power. Instead of the CRF which only generates one optimal sequence, my training strategy uses attention mechanisms that can switch label types flexibly and be applicable to all NER tasks. This training strategy splits the process of pre-train and finetuned where the model can choose to load the pretraining representation or not when the model finetuned on downstream tasks. I first collate the corpus in the dataset and put it into my preferred pre-training mode for pre-training. Then, I let the model load my pre-trained token's representation and fine-tune it using bidirectional LSTM and attention mechanism. A crux advantage of my training strategy is that the model can load the token representation of any pre-trained model instead of just the pre-trained representations that are specialized to the current corpus, making it more versatile and non-wasteful. Although my approach is easily reminiscent of training the model with CRF, my model does not belong to the generative model. I train the model with the method of linear regression due to the difficulty of applying CRF to the nested NER task.

I call my training strategy SNG for "Strive for National Glory." In the same way with prior work of traditional NER tasks, pre-training on tokens from the corpus and fine-tuning downstream tasks. To the best of my knowledge, my work is the first to successfully integrate this training strategy for fine-tuning pre-trained language models. I empirically demonstrate that the new training strategy has desirable properties across several different datasets. My contributions to this work are listed in the following:

- I propose a training strategy for fine-tuning pre-trained language models described in Section 2.
- I obtain strong improvements in the NER task. For the discontinuous NER task, my model's performance outperforms the model proposed by Sydney University by 19.57 points in the CADEC dataset. For the nested NER task, my model improves the current state-of-the-art model by 13.93 points on the GENIA dataset.
- I perform redundancy analysis for the model trained with my strategy.
- I use violin plots to show the data distribution of the model trained with my strategy.
- I use scatterplots to perform a variance analysis of the model trained with my strategy.

2 STRATEGY

I first describe the whole training process; see Figure 2 for an overview. I suggest and evaluate

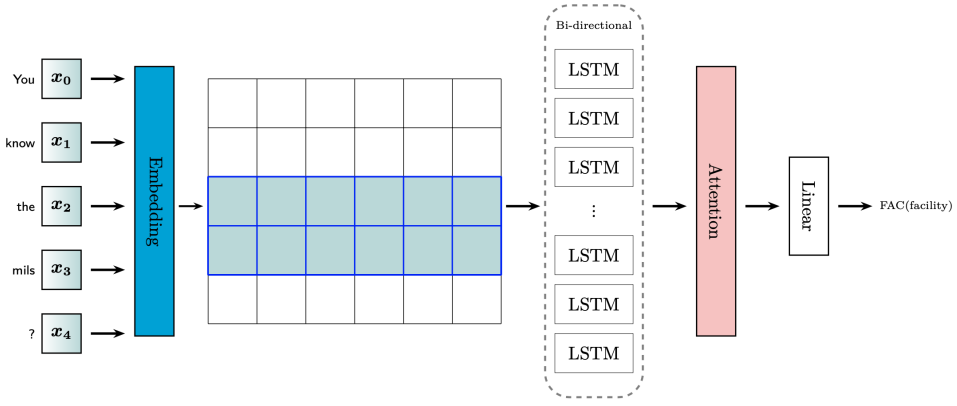

Figure 2: The whole process of the Span-level training strategy.

several modeling improvements for this training strategy in Section 3.2.

My training strategy consists of pre-training and fine-tuning. For the pretraining part, I first tidy up the dataset's contents into the format needed for pretraining and store them in a file. Then, I put the file into the pre-training mode for pre-training. After completing the pre-training, I define the words or symbols in the dataset that are tokenized to be $\boldsymbol{x}$. Each $\boldsymbol{x}$ can be represented by a range of numbers, which I describe using equation 1.

$$\boldsymbol{x} = \{x_0, x_1, \ldots, x_n\},\ x_i \in \mathbb{R} \tag{1}$$

The $\mathbb{R}$ means the set of real numbers.

For the finetuning part, it consists of four steps. Firstly, use the pretraining file to pad the embedding layer which is initialized from Pytorch. Secondly, use the LSTM layer to receive the embedding representation of an entity in a piece of data in the training set to get the hidden state. In this paper, the interface parameter of the Lstm layer I call with PyTorch is bidirectional and returns three tensor variables. The difference between the first two tensor variables is the shape. Thirdly, change the shape of the first tensor. Optimize the first two tensors using the attention mechanism and get the outputs. The attention mechanism in this paper can be summarized in matrix multiplication and softmax. More details on the attention mechanism in this paper need to be understood by reading the code[1]. Fourthly, the output is brought into the linear layer to get the final output. The whole finetuning process requires traversing every entity for every piece of data in the training set and calculating the cross-entropy loss by taking the model's predicted value for the entity and its real value. The model needs to be back-propagated with cross-entropy loss and optimized by epoch.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

I evaluate the datasets including GENIA, Litbank, CADEC, NEREL, GermEval14, ANERcorp, HAREM, and MSRA. I use the standard evaluation metrics of F1 scores described using equation 2, equation 3, equation 4 and equation 5.

$$Accuracy = \frac{Correct\ \ Entities}{Total\ \ Entities} \tag{2}$$

$$Precision = \frac{True\ \ Positives}{True\ \ Positives\ \ +\ \ False\ \ Positives} \tag{3}$$

$$Recall = \frac{True\ \ Positives}{True\ \ Positives\ \ +\ \ False\ \ Negatives} \tag{4}$$

$$F1score = 2 \cdot \frac{Precision \cdot Recall}{Precision\ \ +\ \ Recall} \tag{5}$$

In this paper, the negative sample refers to the discontinuous entity. For all experiments, I pre-train on Glove Pennington et al. (2014), the best pretraining model for the CPU version. All of the pre-training and evaluation contains English, Russian, German, Arabic, Portuguese, and Chinese.

My model most hyperparameters are shown in the Table 1 .

Hyparameters	**Value**
Learning Rate	0.001
Batch Size	1
The Best Train Epochs	1500 for NEREL, 600 for other datasets

Table 1: Fine-tune hyperparameters

3.2 MODEL RESULTS

Table 2 shows the results of the model on the test set.

[1] https://github.com/Qfy1997/NER

Model test on the datasets	Train Time + Hardware	Accuracy	Precision	Recall	F1score
GENIA	46777.8s CPU	91.76%	91.76%	100%	95.70%
Litbank	forget to record CPU	87.93%	87.93%	100%	93.58%
CADEC	24173.2s CPU	88.65%	89.05%	88.09%	88.57%
NEREL	289014.6s CPU	68.66%	68.75%	68.64%	68.69%
GermEval14	78343.9s CPU	69.47%	69.47%	100%	81.99%
ANERcorp	31676.4s CPU	72.73%	72.73%	100%	84.21%
HAREM	10231.1s CPU	64.98%	64.98%	100%	78.77%
MSRA	179908.1s CPU	86.87%	96.87%	100%	98.41%

Table 2: Model Results

Figure 3 illustrates the variation between the mean loss value of a epoch and the epoch. In contrast, the model converges most slowly on the NEREL dataset.

Figure 3: The model loss plot.

3.3 Principle component analysis

Figure 4 shows the results of the PCA analysis. I first scaled the model's output using softmax functions. Then, I reduce the model's output dimension using the covariance matrix, eigenvalues, and eigenvectors. The aim is to allow the different model's outputs with various dimensions to be implemented uniformly into the coordinate system. The different colors in the figure represent the types of entities, the triangular-shaped points represent the points the model predicted correctly, and the circular shapes represent the points that the model predicted incorrectly. Observation reveals that the points in each figure are triangularly distributed in the coordinate system. Further, the output points of the high-performance model are more distributed at the edges of the triangle and are sparser in the center. Since the negatives are inherently non-recommended, this paper only analyzes positives that were predicted correctly or not.

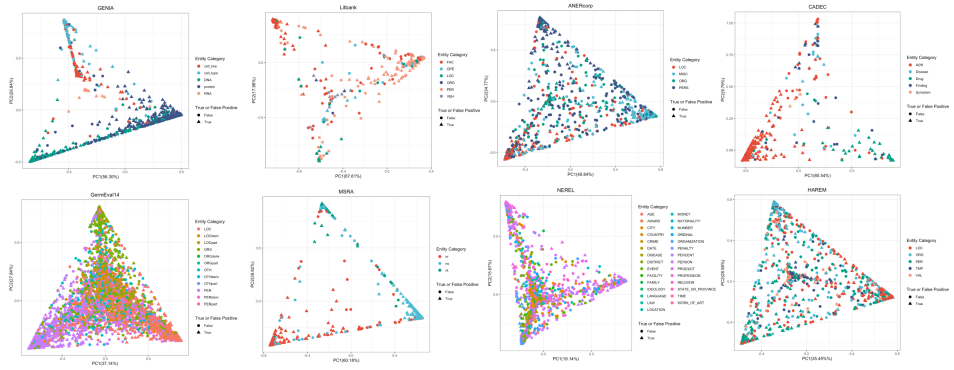

Figure 4: The model's PCA analysis.

3.4 PROBABILITY DISTRIBUTION ANALYSIS

Figure 5 shows the results of the probability distribution analysis. I analyze the probability distribution of the model's output of the real labels. The pink part of the figure represents the probability distribution of the model for false positives. The blue part of the figure represents the probability distribution of the model for true positives. The white horizontal bar inner the each category indicates the median of each distribution. It can be found that most of the probabilities output by the model for correctly predicted entities are distributed around 1, while most of the probabilities output by the model for incorrectly predicting real entities are distributed around 0.

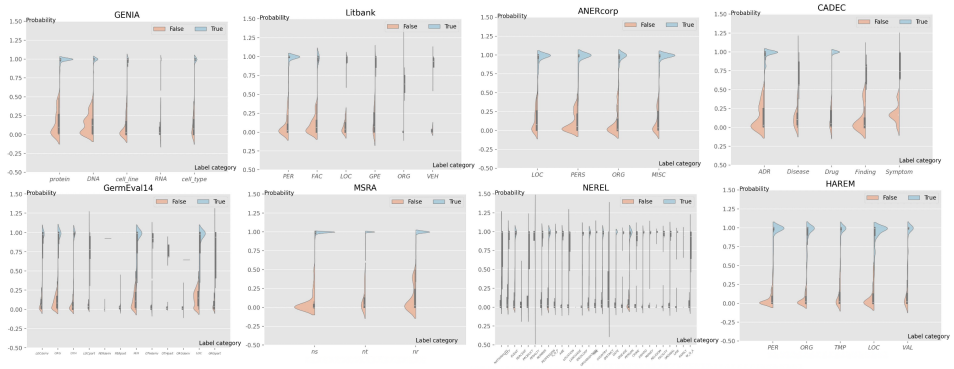

Figure 5: The model's violin plot.

3.5 DISCREPANCY ANALYSIS

Figure 6 shows the results of the discrepancy analysis. I analyzed the difference between the model's output label probability and the true label probability. For the variables on the horizontal axis, I use the model's output probability value to make a ratio with the probability corresponding to the data's real label. Then, I define the category of the results. The results greater than 1 refer to 'discrepancy obviously'. I represent it with an "Up" tag. The results equal to 1 refer to the predict value equal to real label value. I represent it with "Stable" tag. The remaining results indicate that the discrepancy are not obvious. I represent it with "Down" tag. And I logarithmize the results to represent the horizontal coordinate. For the D-value, I use the ratio of the variance of probability values between categories and the probability of the real label probability to represent. In this paper, D means desire. Through observation of the figure 6, the difference significance points with the "Up" tag have a high D-value.

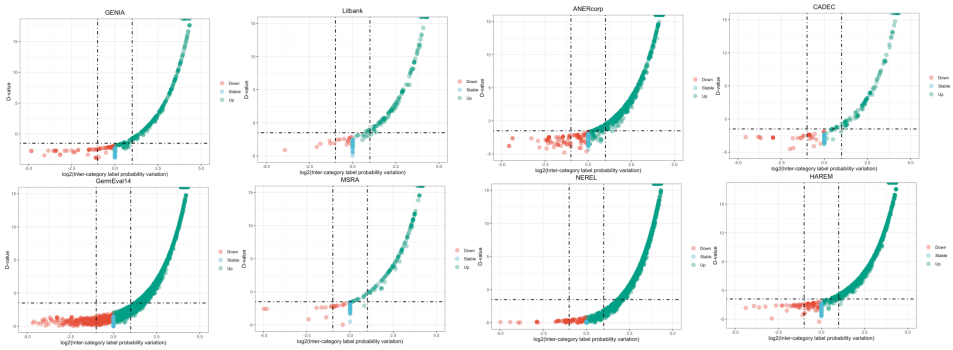

Figure 6: The model's variance analysis.

4 Related Work

4.1 Pre-training for NLP

One of the earliest pre-training methods in NLP can be traced back to the bag-of-words model, a representation that turns arbitrary text into fixed-length vectors by counting how many times each word appears. To address the effect of high-frequency words on the whole sentence, the TF-IDF has been proposed to improve the bag-of-words model. The TF-IDF model effectively emphasizes key words and reduces the weight of high-frequency words. And it is a classical technique widely used in text mining and information retrieval. To introduce the notion of similarity between words, embedding-based models are proposed, where a word or a character is represented by a distributed vector. There have been numerous CPU versions of the embedding-based model. For example, word2vec has two architectures, CBOW and Skip-gram, for learning distributed representations of words that try to minimize computational complexity. The glove introduces the concept of co-occurrence probabilities and utilizes weighted least squares for regression which improves the performance and generalization of the embedding-based model. Along with existing vector dimensions and static embedding not satisfying the status quo, the transformer-based models are proposed successively. The most typical representatives of the transformer-based models include BERT, Roberta, and ELECTRA. The difference between these three models is the training strategy. The training strategy for the BERT includes masking the language model task(MLM) and the next sentence prediction task(NSP). Roberta removes the NSP task on the basic of BERT. ELECTRA expands on the MLM task while eliminating the NSP task by adding a classifier to determine whether the MLM was generated correctly. ELECTRA is current the state-of-the-art pre-training model in CUDA verion. ELECTRA is not recommended that it has no reproduction in Pytorch.

4.2 Long Short Term Memory

LSTM was first proposed by Germans in 1997 and then improved by Google with Pytorch in 2014. Every LSTM cell computes the following equations.

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{ii}\boldsymbol{x}_t + \boldsymbol{b}_{ii} + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hi}) \tag{6}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{if}\boldsymbol{x}_t + \boldsymbol{b}_{if} + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hf}) \tag{7}$$

$$\boldsymbol{g}_t = \tanh(\boldsymbol{W}_{ig}\boldsymbol{x}_t + \boldsymbol{b}_{ig} + \boldsymbol{W}_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hg}) \tag{8}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{io}\boldsymbol{x}_t + \boldsymbol{b}_{io} + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{ho}) \tag{9}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t \tag{10}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(c_t) \tag{11}$$

where σ is sigmoid function, and $\odot$ is the Hadamard product. The symbols in the equations correspond to those in the figure.The symbols in the equations correspond to those in the figure 7.

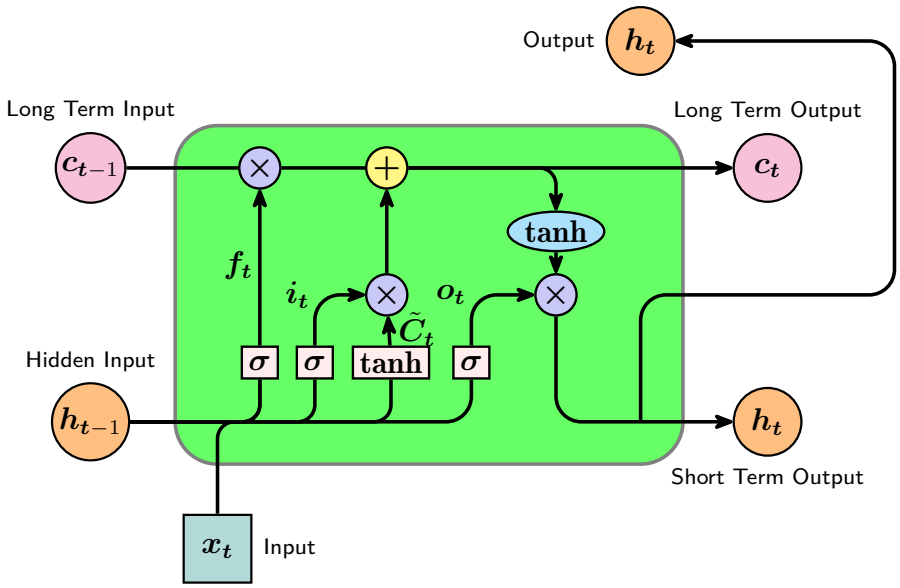

Figure 7: The LSTM cell atchitecture.

4.3 Name entity recognition

Named Entity Recognition was first proposed in the sixth Message Understanding Conference, and is mainly used to recognize organization names, people's names, geographic locations, currencies, time and percentage expressions in text. Traditional methods can be divided into two folders: tagging-based and span-based. Traditional methods can be divided into two folders: generative-based and discriminative-based. For generative-based methods, they usually perform sequence labeling at the token level and then translate into predictions at the span level. Meanwhile, the discriminative-based methods directly perform entity classification on potential spans for prediction. Besides, some methods attempt to take NER to be sequence-to-set or readingcomprehension tasks for prediction. These works designed various transformative approaches, including from word index sequence to entities and from label-enhanced sequence to entities, to unify NER to the text generation task and achieved promising performance and generalizability.

5 Conclusion

I introduce a span-level training strategy that, while being more efficient, performs competitively with recent systems for the NER tasks. This allows for the effortless incorporation of NER models into larger natural language processing pipelines. In addition, the separation of pre-training and fine-tuning makes the model's results more interpretable. This will potentially simplify using various data sources to build robust NER systems. In future work, I plan to study the pre-training embedding in the CUDA version and steadily advance the NLP development process.

Acknowledgments

All experiments were done on a Macbook Pro. Thank for Apple and Microsoft. I use both Apple and Microsoft.

References

Jenny Rose Finkel and Christopher D. Manning. Nested named entity recognition. *DBLP*, 2009.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*, 2014.