Exploring neural architectures for NER

Vincent Billaut
Department of Statistics
Stanford University
Stanford, CA 94305
vbillaut@stanford.edu

Marc Thibault
Institute for Computational and Mathematical Engineering
Stanford University
Stanford, CA 94305
marcthib@stanford.edu

Abstract

Named Entity Recognition is a complex labeling task, and algorithms built to tackle it need to incorporate the whole sentence information to best understand a word's context, in order to reach an optimal predicting performance. In this report, we have explored various ways of taking into account a sentence's crucial past and forward dependencies to enhance the performance of standard deep learning algorithms. We have focused on the impact of memory in a network (using LSTMs), we have expanded it to incorporate forward information (through BiLSTMs), and we have explored a contribution from [2], using a Conditional Random Field to directly consider the dependencies between predicted labels.

We show that using LSTMs, and especially bi-directional LSTMs yields superior performance. We also show that incorporating a CRF at prediction step does not contribute to a better quality of prediction.

1 Introduction

Named Entity Recognition (NER) is one of the main classes of problems tackled by Natural Language Processing. The goal is to identify named entities in sentences, and it can be most relevant for information retrieval or text classification in domains like healthcare. One of the main challenges with NER is the lack of labeled data, and most approaches, before deep learning became commonplace in NLP, used to mostly rely on hand-made rules and features to perform well, but therefore lacked a great deal of robustness and generalizability, in particular when trying to learn or be applied across languages (and a language-agnostic model is one of the key objectives in NER today).

The more recent approaches to NER all tackle the problem in a *deep* fashion, and perform significantly better than previous ones. Some try to use unsupervised learning as a way to augment existing features and provide more insight to the model, but the core of the task remains supervised.

Our goal is to implement various techniques that we have come across in the literature, and to compare their performances against simple, well-tuned models (like simple bi-directional LSTMs) to try and assess whether fanciness is really necessary for NER.

2 Background/Related Work

[2] was the first source of motivation for our work. The relatively recent idea of bringing together a classical Markovian tool — namely Conditional Random Fields (CRF) — that had shown promising results on its own (see [3]) and the modern deep learning approach, as has been tried on sequence tagging in [1], particularly appealed to us. As a matter of fact, one of our main concerns with this hybrid approach is that is doesn't seem like a very natural way to tackle the problem, and so one question guiding our experimentation process was whether it would be possible to reach similar performances without it, but instead by fine-tuning a neural model.

3 Approach

3.1 Base Recurrent model

Our task is classification of each word within a sentence into one of the NER categories. Our model consists of an embedding layer, projecting each word of the vocabulary into a 50-dimensional space. The word embedding is initialized with GloVe ([4]) vectors, and is further trained for this specific task.

We then use a Recurrent Neural Network (RNN) on these embedded words, to leverage the appearance of memory in the network, in this highly past-dependent task. On the outputs of each RNN cell, we apply an extra non-linearly activated layer, to transform the outputs of the Recurrent Network into vectors of the same shape as the number of predicted classes. We finally apply a softmax transformation, and use Cross-Entropy loss.

We also used enhancements of this base model:

- using LSTM cells instead of classical RNN cells to have a more stable memory flux;
- using a Bi-directional Recurrent Network to take into account future information;
- using two (instead of one) extra layers on the outputs of the RNN;
- stacking two Recurrent Networks (one being the input of the other);
- using L2-regularization on learned non-bias parameters;
- using Dropout on the outputs of the RNN.

Consequently, our base Bi-directional LSTM model may be described in the following fashion, with x_t being the input words' embeddings:

$$
\begin{aligned}
&s_0^F, h_0^F = 0;\ s_T^B, h_T^B = 0; && \textit{initialize LSTMs states}\\
&s_t^F, o_t^F = LSTM^F(s_{t-1}^F, o_{t-1}^F, x_t)\ \ \forall t \in [1, T] && \textit{propagate in forward LSTM}\\
&s_t^B, o_t^B = LSTM^B(s_{t+1}^B, o_{t+1}^B, x_t)\ \ \forall t \in [0, T-1] && \textit{propagate in backward LSTM}\\
&o_t = [o_t^F, o_t^B] && \textit{concatenate outputs}\\
&\theta_t = \sigma(W.o_t + b) && \textit{add extra layer}\\
&\hat{y}_t = softmax(\theta_t) && \textit{apply softmax}\\
&loss = CE(\hat{y}_t, y_t) && \textit{compute cross-entropy}
\end{aligned}
$$

3.2 Incorporating the Conditional Random Field

We also chose to incorporate the information of a CRF fitted on the train data, as suggested by Lample et al[2]. It encodes a Markov Chain, describing the probability of transitioning from a given predicted label to another:

$$P[i, j] = Pr(y_{t+1} = y_j | y_t = y_i) \textit{ in training set}$$

, where P is a stochastic matrix of size (k, k), when k is the number of considered classes. However, we tried to use the information of the CRF in a different way than in this paper. Instead of using the transition matrix P at each learning step, we use it only for prediction. This approach allows the CRF to be incorporated as an extraneous source of information on top of any trained predictor.

Consequently, we chose to use this information to help us in predicting labels, from the outputs $\hat{y}_t$ of the Recurrent Network, using an iterative method. We begin by labeling the first word by taking the optimal prediction outputed by the network. For further words, we select the maximum of a weighted combination of the network's output, and the probability of transitioning from the previously labeled word to the other considered labels:

$$
\begin{aligned}
&z_0 = argmax_j(\hat{y}_0[j])\\
&\text{and}\\
&z_t = argmax_j(\hat{y}_t[j] + \alpha.P[z_{t-1}, j])\ \ \forall t \in [1, T]
\end{aligned}
$$

Where α is a hyper-parameter which has to be chosen. This choice of method is meant to partially reproduce the method developed in [2], by reducing the search beam to a greedy iterative algorithm.

4 Experiments

4.1 The data

The CoNLL-2002 ([5]) and CoNLL-2003 ([6]) shared tasks provide frameworks and benchmarks for language-independent NER, as well as comprehensive labeled datasets. It has become the standard dataset for benchmarking NER performances since, and we have relied on it (specifically, CoNLL-2002) for our training and evaluation. It consists in about one million labeled words, across around 50,000 sentences.

4.2 Tuning the network architecture

Our main study consists in exploring the different architectures and parameters, and determining which perform best for our task. First we compared the main architecture schemes: a simple feed-forward neural network, a LSTM, and a bi-directional LSTM, each time finding near-optimal settings for hidden layer sizes and learning rates. Then, restraining ourselves on the best performing one, we tried tuning two other parameters: the L2 regularization parameter and the dropout rate.

4.2.1 Network architecture

First of all, we wanted to have a good-enough baseline, upon which to build and experiment later on. We tried the following architectures:

- simple feed-forward neural network,
- LSTM with prediction layer on top,
- LSTM with two layers on top,
- bi-LSTM with prediction layer on top,
- bi-LSTM with two layers on top,
- stacked LSTMs,

and present the aggregated results in Fig 1. We chose to continue the study with a **bi-LSTM**, because we found it to be the simplest model among the best performing ones.

4.2.2 Dropout rate

The first learning parameter we looked at in details is the dropout rate. It gives the portion of a layer's neurons we want arbitrarily desactivated at each pass over the network, and it is designed to give the network more robustness (because the neurons "learn" to not depend too much on each other). We find that the best performances, with everything else fixed, are achieved with a dropout rate around 20%, as shown in Fig 2 — comparing learning curves — and Fig 3 — comparing best performances.

4.2.3 L2 regularization

Once satisfied with our bi-LSTM model with 20% dropout, we tried tuning L2 regularization, keeping everything else fixed.

L2 regularization can be written as follows:

$$regularized\,loss = loss + \beta\,(||W_1||_2 + ||W_2||_2 + ...)$$

where the W_j are the weights matrices involved in the model, and β is the regularization hyper-parameter, which determines how strong we want to penalize large weights.

We tried a rather wide grid of values for β, and could compare each one's performance, both looking at the learning curves (Fig 4) and at the best performance (Fig 5). Surprinsingly, the unregularized

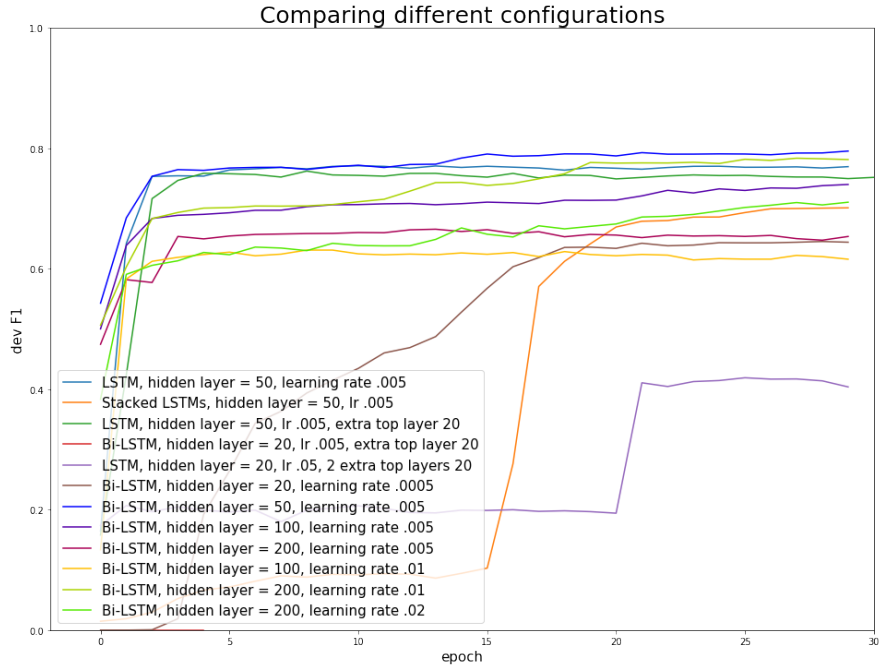

Figure 1: Comparing many different architectures and learning parameters led us to choose a **bi-LSTM** for the rest of our study. Some models with additional layers may have been able to perform as well with finer tuning, but we considered that the simpler would bring the most robustness

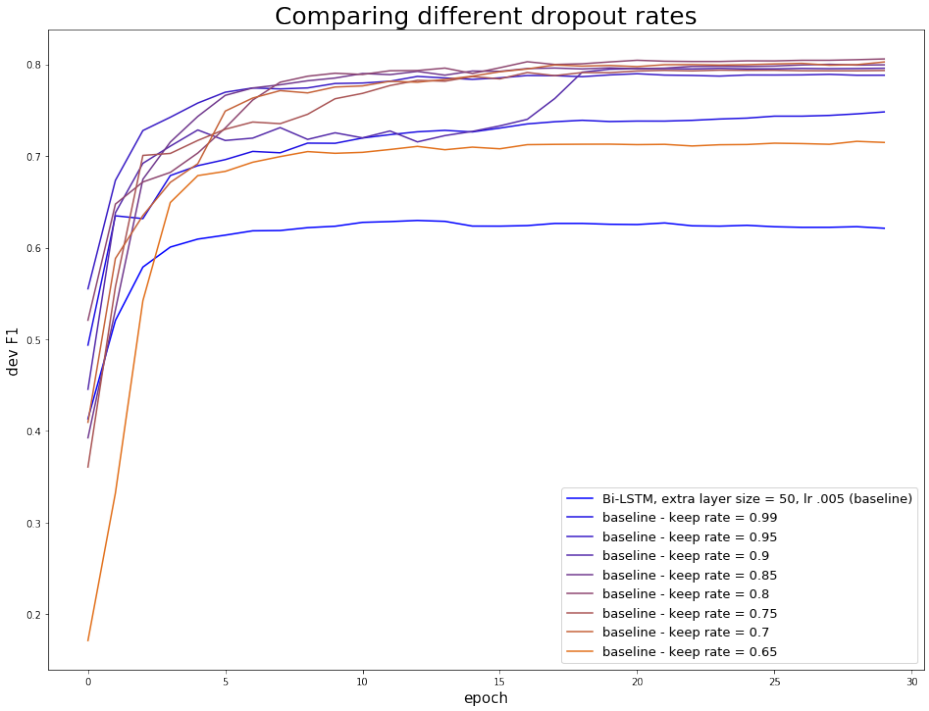

Figure 2: Learning curves for different dropout rates (dev set F_1 score as a function of the number of epochs) ; we see that the most optimal dropout rate for learning is around 20%

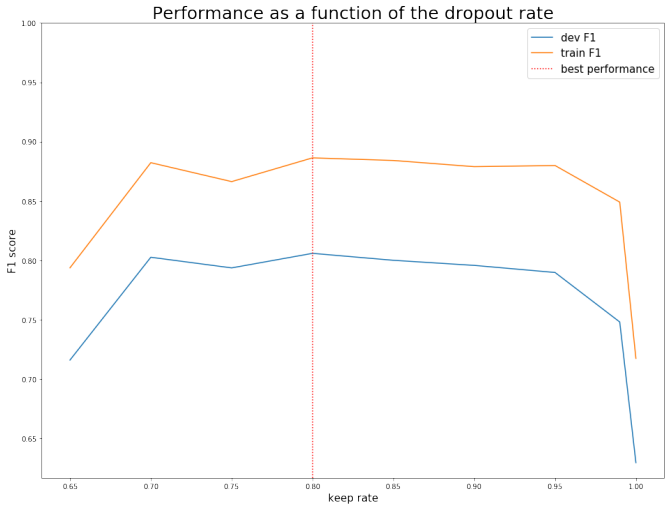

Figure 3: Best performance of the model, as a function of the keep rate (1 - dropout rate)

baseline is almost unbeatable. Still, we consider it sane to introduce some weight penalization, and therefore keep $\beta = 3\ 10^{-5}$.

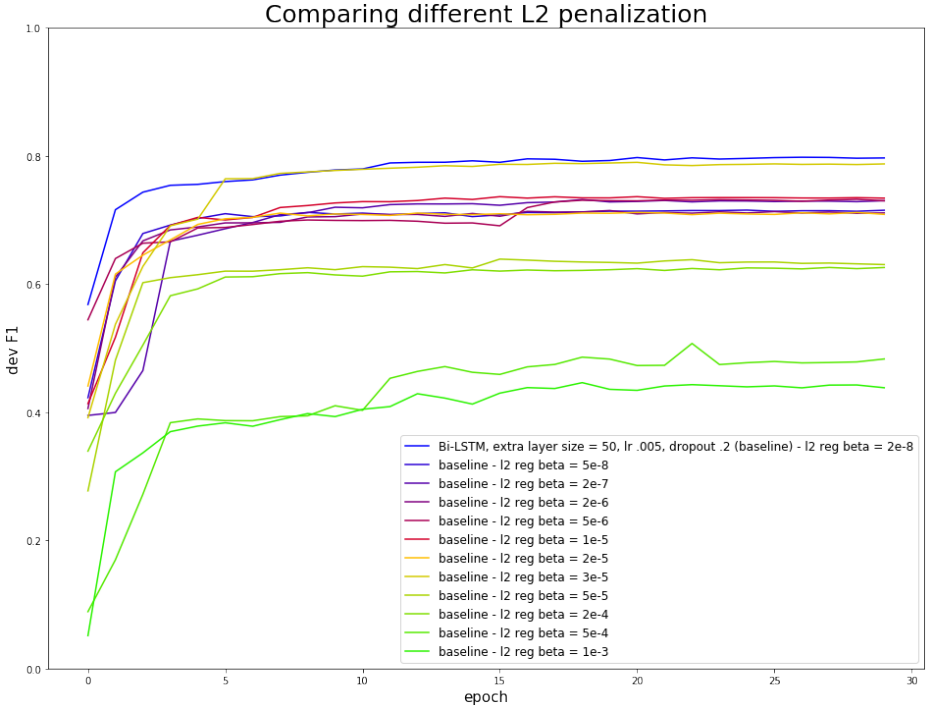

Figure 4: Learning curves for different β values (dev set F_1 score as a function of the number of epochs) ; we see that surprinsingly, few regularized models reach the out-of-sample performance of the unregularized baseline

4.3 Incorporating the CRF

We have tried to incorporate the prediction of a pre-trained CRF on top of two models: a simple LSTM model with 50 recurrent cells and an extra layer, as well as our best-performing model, a

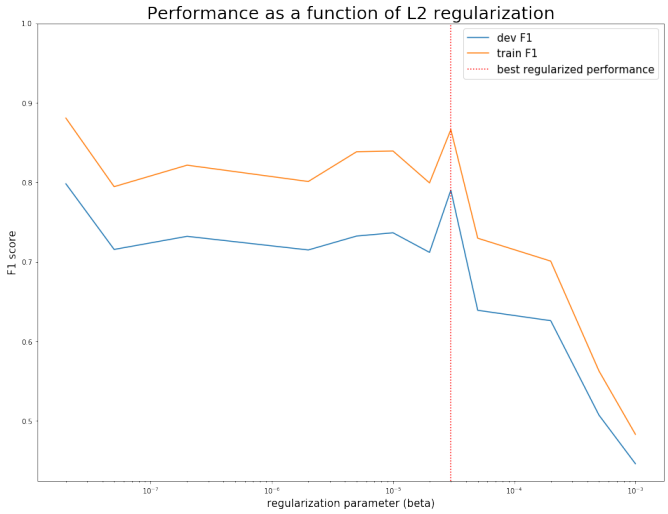

Figure 5: Best performance of the model, as a function of the regularization parameter ; we select the most penalized model among the best performing ones.

Bidirectional LSTM. We have tried several values of the α parameter on an exponential scale, to see which one yields the best predicting performance. The α value which performs best on the train set is the one we will choose for evaluating our dev set.

We have summarized the results of varying the CRF's α parameter on our flagship Bi-LSTM model in Fig 6. First, we see that the dev F_1 performance curve closely follows the train F_1 curve. This means that tuning the α parameter does not lead to an overfitting on specific subsets of the studied set. However, we obtain disappointing results on the contribution of the information of the CRF. The best results are obtained by not considering the CRF. Using more information from the CRF yields a steady decrease in F_1 performance.

It appears that the way we included the CRF in our model is not relevant for prediction. The information carried by the labels transition matrix P is not predictive enough, and it is over-expressed compared to the recurrent networks before being able to contribute in a constructive way.

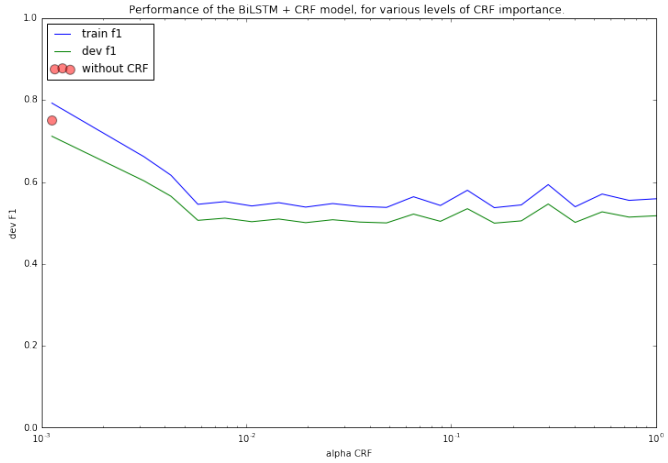

Figure 6: Train and Dev set performance of a Bi-LSTM model, incorporating a CRF on the prediction step, versus the importance α given to the CRF. No positive level of importance yields superior performance. The best decision is not to use the extra information from the CRF.

5 Conclusion

We have produced the following table to summarize our main results for each attempted technique.

Model	Dev F_1
Naive word-by-word	0.691
LSTM	0.597
LSTM + CRF	0.475
LSTM + extra layer	0.770
LSTM + extra layer + CRF	0.751
Bi-LSTM	0.796

Overall, we see that it is highly relevant for a NER task to incorporate context information using recurrent networks. Our best-performing model is the Bi-LSTM, which means that future information is also crucial when labelling named entities.

We chose to use a simplified version of [2] to use the information of a Conditional Random Field on the set of labels. We have shown that the way we implemented prediction algorithm using both a Neural Network and a CRF is not relevant when labelling.

The next leads we are currently following to extend and enhance our models are using an attention layer on our recurrent network, as well as using CRFs in a smarter way, such as at training step, or using a bidirectional CRF.

References

[1] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[2] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[3] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics, 2003.

[4] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[5] Tjong Kim Sang and Erik F. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158, 2002.

[6] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.