Named Entity Recognition with Stack Residual LSTM and Trainable Bias Decoding

Quan Tran, Andrew MacKinlay and **Antonio Jimeno Yepes**
IBM Research Australia
hung.tran@monash.edu, admackin@au1.ibm.com, ayepes@au1.ibm.com

Abstract

Recurrent Neural Network models are the state-of-the-art for Named Entity Recognition (NER). We present two innovations to improve the performance of these models. The first innovation is the introduction of residual connections between the Stacked Recurrent Neural Network model to address the degradation problem of deep neural networks. The second innovation is a bias decoding mechanism that allows the trained system to adapt to non-differentiable and externally computed objectives, such as the entity-based F-measure. Our work improves the state-of-the-art results for both Spanish and English languages on the standard train/development/test split of the CoNLL 2003 Shared Task NER dataset.

1 Introduction

In Natural Language Processing, the term "Named Entity" refers to special information units such as people, organizations, location names, numerical expression (Nadeau and Sekine, 2007). Identifying the references to these special entities in text is a crucial step toward Language Understanding. Thus, there have been many works on these areas.

Some of the early systems employed hand-crafted rules (Rau, 1991; Sekine and Nobata, 2004), however, the vast majority of current systems rely on machine learning models (Nadeau and Sekine, 2007) such as Conditional Random Field (CRF) (McCallum and Li, 2003), Hidden Markov Model (HMM) (Bikel et al., 1997) and Support Vector Machine (SVM) (Asahara and Matsumoto, 2003). Although the traditional machine learning models do not rely on manual rules, they require a manual feature engineering process, which is rather expensive and dependent on the domain and language.

In recent years, Recurrent Neural Network (RNN) models such as Long-Short-Term-Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Chung et al., 2014) have been very successful in sequence modeling tasks, for example, Language Modeling (Mikolov et al., 2010; Sundermeyer et al., 2012), Machine Translation (Bahdanau et al., 2014) and Dialog Act Classification (Kalchbrenner and Blunsom, 2013; Tran et al., 2017). RNN models can learn from basic components of text (i.e. words and characters). This generalization capability facilitates the construction of Language Independent NER models (Ma and Hovy, 2016; Lample et al., 2016) that rely on unsupervised feature learning and a small annotated corpus.

One simple way of adding representational power to a neural network is layer stacking. A traditional feed forward neural network usually has three fully connected layers: an input layer, a hidden layer, and an output layer. For a Convolutional Neural Network (CNN) or Recurrent Neural Network, the number of stacked layers might be much larger (Amodei et al., 2016). One problem with this stacking scenario is the degraded representation problem (He et al., 2016). The proposed solution for this problem is the residual-identity connection (He et al., 2016). With the information from the lower-level inputs, the upper neural network layers can learn to compensate for the representation errors of lower layers. We adopt this idea for Stacking RNN, however, with a different implementation.

Most of the RNN-based models for NER and machine translation are trained with some form of maximum log-likelihood loss. However, it is often desirable to optimize task-specific metrics (Xu et al., 2016), for example, F-measure

Proceedings of the The 8th International Joint Conference on Natural Language Processing, pages 566–575,
Taipei, Taiwan, November 27 – December 1, 2017 ©2017 AFNLP

in NER, but optimizing the F-measure directly is not trivial (Busa-Fekete et al., 2015), especially in the case of complex Deep Neural Network models. It is even more difficult considering the way the F-measure is calculated in Named Entity Recognition in the CoNLL-2003 shared task[1] , where it depends on the actual/predicted entities and *not* on each token-prediction for which the system is trained for. Inspired by the idea of trainable decoding recently proposed in machine translation (Gu et al., 2017), we introduce a trainable *percentage bias decoding* system that manipulates the outputs of a base system trained with normal loss to adapt to a new objective. Our trainable bias decoding system also bears similarity to the thresholding technique (Lipton et al., 2014), traditionally used to maximize F-measures given a classifier. The proposed decoding system is trained directly on the externally computed F-measures (which relies on the the CoNLL evaluation script) using finite different gradient.

In the next sections, we describe the proposed innovations with detailed motivations and discussions. Results show that our proposed innovations improve the NER state-of-the-art for the English and Spanish languages in the CoNLL-2003 shared task data set.

2 Models

We describe first our RNN-CRF base architecture and then we describe our two modelling innovations: the Stack Residual RNN and the bias decoding.

2.1 The base RNN-CRF architecture

Our system is built upon the RNN-CRF architecture for Named Entity Recognition. Let us denote the input sequence of words as $w_0, ..., w_n$. In general, the RNN component encodes the words into a sequence of hidden vectors $h_0, ..., h_n$. This sequence of hidden vectors is then treated as features for a linear-chain CRF layer. The training objective will then be the log-likelihood of the correct sequence.

Following Lample et al. (2016); Ma and Hovy (2016); Yang et al. (2016), we employ the character level information and word-embeddings as features in NER. Similar to Lample et al. (2016), in our system, character information is encoded using a bi-directional RNN (biRNN) over characters. Given a word $w_k \in w_0, ..., w_n$ with m characters, let us denote the character-embedding sequence of this word as $\mathbf{c}_0, ..., \mathbf{c}_m$, the biRNN function as ρ and the concat function as ψ. The character embedding representation $\mathbf{h}^c_k$ of word w_k is calculated using a biRNN as in Equation 1, in which $\mathbf{f}_0, ..., \mathbf{f}_m$ and $\mathbf{b}_0, ..., \mathbf{b}_m$ denote the hidden units in the forward and backward RNNs respectively.

$$\begin{aligned} \mathbf{f}_0, ..., \mathbf{f}_m &= \rho(\mathbf{c}_0, ..., \mathbf{c}_m) \\ \mathbf{b}_0, ..., \mathbf{b}_m &= \rho(\mathbf{c}_m, ..., \mathbf{c}_0) \\ \mathbf{h}^c_k &= \psi([\mathbf{f}_m, \mathbf{b}_m]) \end{aligned} \tag{1}$$

The feature vector $\mathbf{x}_k$ of word w_k is then the concatenation of $\mathbf{h}^c_k$ and the traditional word-embedding $\mathbf{e}_k$ as shown in Equation 2. Figure 1 shows the feature extraction procedures. All the parameters of the biRNN as well as the embedding tables are jointly trained with other component of the model. The word embedding table is initialized with a pre-trained embedding table.

$$\mathbf{x}_k = \psi([\mathbf{h}^c_k, \mathbf{e}_k]) \tag{2}$$

The most simple architecture would be a one-directional RNN over the word features: $h_0, ..., h_n = RNN(x_0, ..., x_n)$. However, it has been shown to be beneficial to have a bidirectional RNN over the input layer, as a bidirectional RNN captures both the left and the right context of a word: $h_0, ..., h_n = \rho(x_0, ..., x_n)$.

The final sequence of hidden vectors $\mathbf{h}_0, ..., \mathbf{h}_n$ is treated as the features for a linear-chained CRF layer. Similar to Lample et al. (2016), the observation scores λ are calculated with a linear transformation from the hidden vectors, as show in Equation 3, where λ_i is the vector observation scores for all the labels in ith time-step, $\mathbf{W}_p$ is an $l \times d$ weight matrix, and $\mathbf{b}_p$ is a bias vector of size l (with d is the size of vector $\mathbf{h}_i$), and l is the size of the label set $\mathbb{Y}$ (including the special sequence begin and end labels).

$$\lambda_i = \mathbf{W}_p \mathbf{h}_i + \mathbf{b}_p \tag{3}$$

Given a sequence of input words $S = w_0, ..., w_n$, the score of a particular sequence of labels $Y = Y[0], ..., Y[n]$ is calculated using the observation scores λ and the transition scores δ as in Equation 4, where δ is a square matrix of dimension $l \times l$, $\delta(Y[j], Y[j+1])$ denotes the transition

[1] https://www.aclweb.org/aclwiki/index.php?title=CONLL-2003_(State_of_the_art)

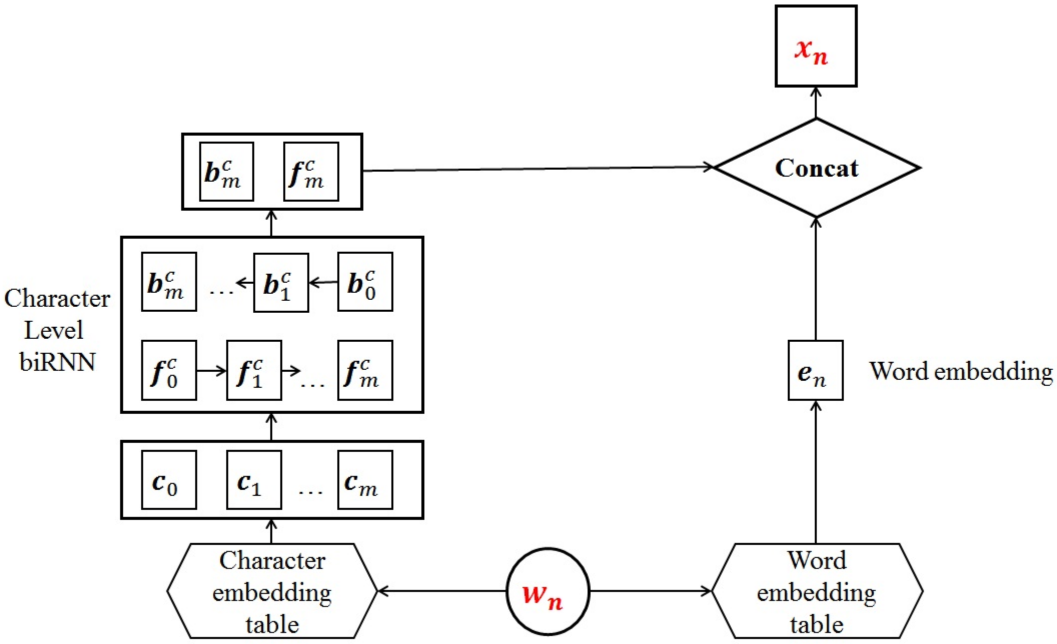

Figure 1: Extracting word features with word embeddings and character level biRNN

score between the label in position j and the label in position $j+1$ in sequence Y, and $\lambda_i(Y[i])$ is the observation score of i-th label $Y[i]$.

$$\zeta(Y,S) = \sum_{i:[0..n]} \lambda_i(Y[i]) + \sum_{j:[0..n-1]} \delta(Y[j], Y[j+1]) \tag{4}$$

The probability of a sequence Y is calculated using a softmax over all the possible sequences $\mathbb{Y}$ (Equation 5).

$$Pr(Y|S) = \frac{e^{\zeta(Y,S)}}{\sum_{\bar{Y}\in\mathbb{Y}} e^{\zeta(\bar{Y},S)}} \tag{5}$$

During training, we maximize the log-likelihood of the correct sequence Y_c. The loss function $\mathbb{L}$ is defined in Equation 6. Because we employ a linear-chain CRF, the term $log(\sum_{\bar{Y}\in\mathbb{Y}} e^{\zeta(\bar{Y},S)})$ in Equation 6 can be efficiently calculated with dynamic programming.

$$\mathbb{L}(Y_c) = \zeta(Y_c, S) - log(\sum_{\bar{Y}\in\mathbb{Y}} e^{\zeta(\bar{Y},S)}) \tag{6}$$

During decoding, the best sequence can be found using the Viterbi algorithm. The original Viterbi decoding algorithm builds an $l \times n$ score table ξ in which l is the size of the label set (including the beginning and end labels) and n is the length of the sequence. $\xi_j(y_i)$ denotes the score of the most probable partial path (up to position j) with position j having the label y_i. $\xi_j(y_i)$ is calculated using dynamic programming as in Equation 7.

$$\xi_j(y_i) = \sum_{y_k\in\mathbb{Y}} (\xi_{j-1}(y_k) + \delta(y_k, y_i)) + \lambda_j(y_i) \tag{7}$$

At the end of the decoding process, sequence $\hat{Y}$ is predicted by selecting the best score at the end of the sequence $j = n$ and then completing the sequence with a backward pointer (Equation 8). Figure 3 depicts the whole RNN-CRF architecture

$$\begin{aligned} \hat{Y}[n] &= \arg\max_{y_i} \xi_n(y_i) \\ \hat{Y}[n-1] &= \arg\max_{y_k} (\xi_{j-1}(y_k) + \delta(y_k, \hat{Y}[n])) \end{aligned} \tag{8}$$

2.2 Stacked Residual RNN

A traditional way of adding more representational power to a neural network is layer stacking. RNN stacking has been successfully used in a lot of works (Amodei et al., 2016). However, stacking

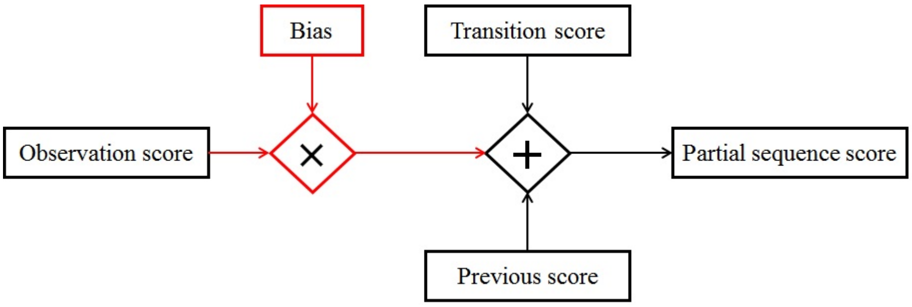

Figure 2: The application of percentage bias to Viterbi decoding

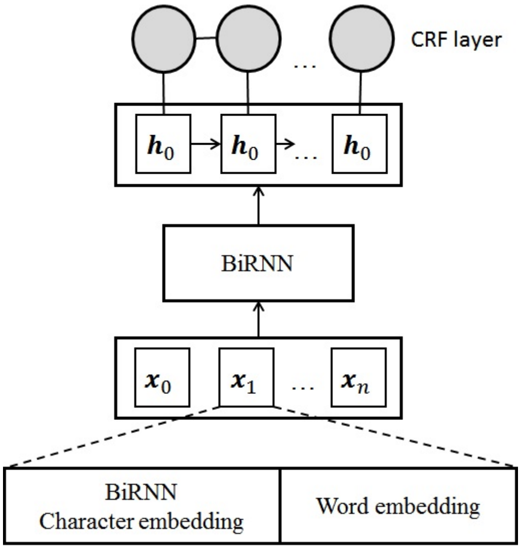

Figure 3: The RNN-CRF architecture

layers of neural networks suffers from the *degradation problem* (He et al., 2016). This is due to the difficulty in training a lot of stacked layers and fit these layers to desired underlying mappings, which leads to representational degradation.

The solution proposed to this problem, the *residual connection* (He et al., 2016; Prakash et al., 2016) tries to create shortcuts between non-consecutive layers. However, the original additional residual connection (adding the input vector to the hidden representation) adds several constraints on the dimensionality of the hidden and input layers, which might require vector clipping (Prakash et al., 2016), and it might lead to a loss of information.

In the original residual connection proposed for image recognition, the residual information is summed to the output of the upper layers ($\mathbb{F}(x) + x$). In our proposal, we want the upper layer of a neural network to have direct access to the original input, thus, the original input is now appended to the output of the lower layers instead of being summed. With this formulation, there is no dimensionality restriction, and furthermore, we argue that our proposed residual connection can be used to mix feature learners of different complexity (Figure 4). For example, when equipped with our proposed residual connection, the top neural network layer can act like a shallow one-layer feature learner. The two top layers can act like a deeper two-layer feature learner. Equation set 9 shows the exact formulation of our proposed residual connection within the Stack RNN. Similar to the Equation 1, we denote the biRNN function as ρ and the concat function as ψ. This modelling procedure is depicted in Figure 4.

$$\begin{aligned}
\mathbf{h}_0^0, ..., \mathbf{h}_n^0 &= \rho(\mathbf{x}_0, ..., \mathbf{x}_n) \\
\hat{\mathbf{h}}_0^0, ..., \hat{\mathbf{h}}_n^0 &= \psi([\mathbf{x}_0, \mathbf{h}_0^0]), ..., \psi([\mathbf{h}_n^0, \mathbf{x}_n]) \\
\mathbf{h}_0^1, ..., \mathbf{h}_n^1 &= \rho(\hat{\mathbf{h}}_0^0, ..., \hat{\mathbf{h}}_n^0) \\
\mathbf{h}_0^M, ..., \mathbf{h}_n^M &= \rho(\hat{\mathbf{h}}_0^{M-1}, ..., \hat{\mathbf{h}}_n^{M-1})
\end{aligned} \tag{9}$$

2.3 The bias decoding

Usually NER systems are evaluated with some form of F-measure. For example, for the CoNLL 2013 Shared Task NER dataset, the evaluation is performed by an external script using entity-based F1-measure. Although it has been noted that training on the evaluation metric is beneficial (Xu et al., 2016), most of the deep models for NER are trained with log-likelihood. The main reason for this discrepancy is the difficulty in training with F-measures. Instead of trying to train on F-measure directly, we look into a hybrid solution where we train a model on log-likelihood first, and then use a simpler "adaptation model" to manipulate the output of the base model to fit it to the F-measure.

Inspired by Machine Translation research on decoding with trainable noise (Gu et al., 2017), we

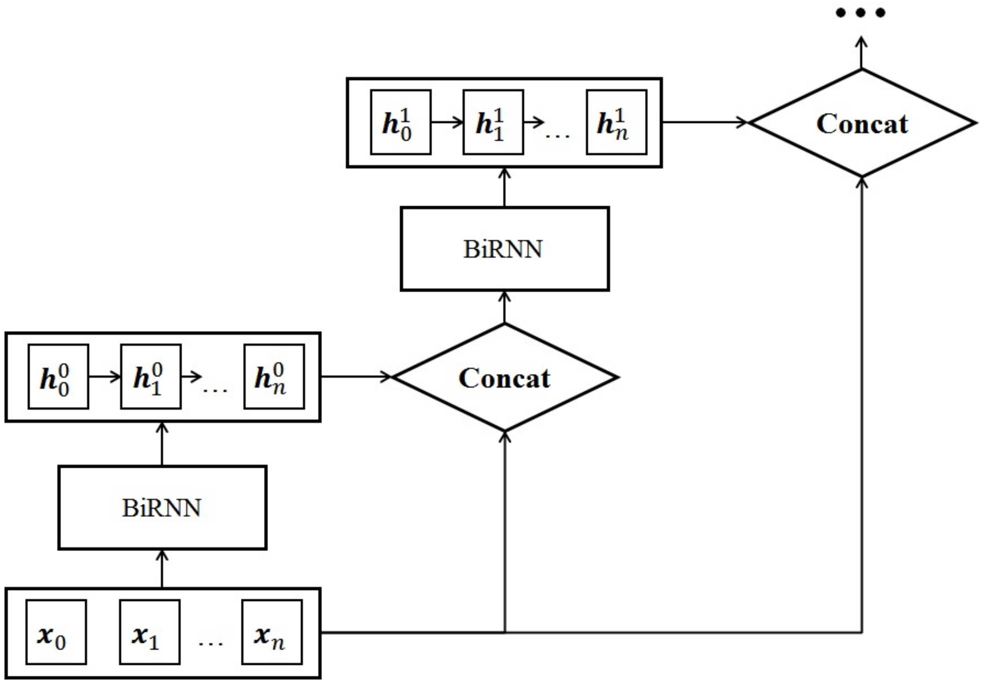

Figure 4: Feature learner mixture with residual connection

explore the possibility of adding trainable noise to the Viterbi decoding process. Analogously to the traditional threshold technique for maximizing the F1 score in binary classification, we introduce a simple *percentage noise* to the decoding process. That is, during the construction of the score table ξ (Equation 7), a label-specific percentage bias is added to the calculation as in Equation 10. Figure 2 shows the application of this bias to the Viterbi decoding.

$$\xi_j(y_i) = \sum_{y_k \in \mathbb{Y}} (\xi_{j-1}(y_k) + \delta(y_k, y_i)) + b_y \lambda_j(y_i) \tag{10}$$

To test this new percentage bias idea, we perform a quick experiment, where we limit the use of bias to the most numerous class in the CoNLL tag set, class O (words that do not belong to any entity). We search for the best bias b_O from the range of $[0.5, 1.5]$ using a value loop with step of 0.1. For each value of b_O, we calculate the F1-measure on the validation set, and choose the value with the highest F1. We use our trained model based on the Stack Residual architecture above as the base probabilistic model. We find that the best b_O value is 1.1 (a value of 1.0 means without any bias). Using this b_O bias for the test data yields the F1-measures of 91.22 compared to the original score of 91.07 in the test set. This experiment supports our claim that the base model trained with log-likelihood might not optimize well on a different performance measure, and adding this percentage bias noise is really beneficial.

We extended this idea treating the biases as parameters. Thus the trainable bias decoding system has the number of parameters equal to the number of classes. Training with gradient descent with CoNLL's entity-based F1 loss is rather difficult, as it is hard to calculate the exact gradient. This is solved using the numerical gradient methods as an approximation, which is shown in Equation 11.

$$f'_b \sim \frac{f(b+\epsilon) - f(b-\epsilon)}{2\epsilon} \tag{11}$$

The training procedure is then very similar to stochastic gradient descent. Details on the choice of hyper parameters and other experimental settings are presented in the Experiment section.

3 Experiments

3.1 Dataset and Experimental settings

We have prepared and evaluated the proposed methods on the English and Spanish sets of the CoNLL 2003 NER data set[2] (Tjong Kim Sang and De Meulder, 2003). We have reused the training, development and test set configuration of the CoNLL-2003 Shared Task in our study.

The training set has been used to train the system using several hyperparameter configurations, the development set has been used to select the best configuration and the reported performance of the final system is based on the test set. The Spanish dataset has 8323/1915/1517 sentences in train/dev/test sets respectively. The English dataset is almost twice as large with 14041/3250/3453 sentences in train/dev/test set. For all of our models, the word-embedding size is set to 100 for English and 64 for Spanish. The hidden vector size is 100 for both English and Spanish sets without the LM embeddings. With the LM embeddings, the hidden vector size is changed to 300 for English. We trained the model with Stochastic Gradient Descent (SGD) with momentum, using the learning rate of 0.005. For the bias decoding, the ϵ hyperparameter for each update is randomly chosen from a range of $[0.01, ..., 0.1]$ with step-size of 0.01. Because the base model trained with sequence level log-likelihood fits very well on the training set, the gradient calculated with Equation 11 might be every small, thus we opt to calculate the finite difference with respect to the loss: $log2(1-F1/100)$ instead of the $F1$ to boost the gradient information in the points where F1 is very close to 100 (perfect classification). The learning rate for bias training is also set to 0.005. Statistical significance has been determined using a randomization version of the paired sample t-test (Cohen, 1996).

We first conduct several series of experiments to confirm the effectiveness of our two proposed ideas: the Stack Residual RNN and the bias decoding, and the new Language Model embedding in sub-section 3.2. The second sub-section: 3.4 compares our method with state-of-the-art results.

3.2 Component Analysis

Adding stack Residual RNN

Due to computational complexity, there is a practical limit on how many RNN layers we can stack. In this series of experiment, we tested our model without Stacked Residual RNN, and with 2, 3 and 4 Stacked layers. The word embeddings are initialized using the pre-trained word vectors described below. The result of this series of experiements is presented in Table 1.

From the result, we can see that the performance seems to increase as we add more stacked layers, and peak at three before dropping. We continue to analyze other components using 3 Stacked Residual Layers of CRF-RNN as the base model, we call this model *3 Res-RNN* for short.

For English, the 3 and 4 stacked layer improvements are significant ($p < 0.025$) compared to the baseline model and between the stacked layer models, the improvement between 2 and 3 layers is significant ($p < 0.035$).

For Spanish, the 3 stacked layer improvement is significant ($p < 0.03$), with respect to the baseline model. Improvement between the 3 stacked layer and the 4 stacked layer models is significant ($p < 0.03$).

Adding Language Model Embedding

Pre-trained word embeddings have shown useful in Natural Language Processing tasks, but provide information about the word but not about its context. Previous work has explored using language models in addition to word embeddings (Peters et al., 2017) with positive results. We have evaluated our system using pre-trained language models using the 3 Stacked Residual Layer configuration. First, we test the models with forward-only LM embeddings (foreLM), then we test the model with both forward and backLM (backLM). The result of this series of experiments is presented in Table 2.

The gain from the LM embedding is not consistent. It seems to work very well with English, where it improves performance substantially even though this improvement is not specially significant. However, the LM does not improve the performance at all in Spanish. Adding the foreLM and backLM significantly decreases performance.

Adding Bias Decoding

We test the bias decoding on models with and without LM embeddings, with results shown in Table 3. The bias-decoding increases the performance across the board, however the performance increases are not consistent. The increases are notable on some cases (3 Res-RNN + bias on both English and Spanish, 3 Res-RNN + foreLM + backLM + bias for Spanish), while in some cases the increases are minimal (3 Res-RNN + foreLM + bias on both English and Spanish, 3 Res-RNN + foreLM + backLM + bias on English).

[2] http://www.cnts.ua.ac.be/conll2003/ner

System	F1 English	F1 Spanish
CRF-RNN no Stack Residual	90.43	85.41
CRF-RNN 2 Stack Residual	90.72	85.88
CRF-RNN 3 Stack Residual	**91.07** $\star$	**86.24** $\star$
CRF-RNN 4 Stack Residual	91.02 $\star$	85.51

Table 1: Analysis of the Stack Residual Component. $\star$ indicates significance ($p < 0.05$) versus CRF-RNN no Stack Residual.

System	F1 English	F1 Spanish
3 Res-RNN	91.07 $\star$	**86.24** $\star$
3 Res-RNN+foreLM	91.43 $\star$	86.13 $\star$
3 Res-RNN+foreLM +backLM	**91.66** $\star$	85.83

Table 2: Analysis of the Language Model Embedding. $\star$ indicates significance ($p < 0.05$) versus CRF-RNN no Stack Residual in Table 1.

For English, adding bias to the 3 Res-RNN without LM yields a significant improvement ($p < 0.013$), while for Spanish, the boost from adding bias to the 3 Res-RNN + foreLM + backLM model is significant ($p < 0.011$).

3.3 External Knowledge Learning

3.3.1 Word embedding

English word embedding was obtained from Word2vec-api[3]. The embedding dimension is 100 and it was trained using GloVe with AdaGrad. For the generation of Spanish word embeddings we followed Lample et al. (2016), using Spanish Gigaword Third Edition[4] as corpus with an embedding dimension of 64, a minimum word frequency cutoff of 4 and a window size of 8.

3.3.2 Language Modeling

In some experiments, we used both forward and backward language models. The English forward language model was obtained from TensorFlow[5] using the One Billion Word Benchmark[6] (Chelba et al., 2013) and has a perplexity of 30. As the code generating this pre-trained model is not available, we made use of a substitute which produces a higher perplexity language model. For the backward English language model and the Spanish forward and backward ones, they were generated using an LSTM based baseline[7] (Jozefowicz et al., 2016). This code estimates a forward language model and was adapted to estimate a backward language model. Language models were estimated using the One Billion Word benchmark. The vocabulary for the backward English model is the same as the pre-generated forward model. The perplexity for estimated backward English language model is 46; despite the discrepancy in perplexity with the forward language model the performance using this language model still improves the named entity recognition task. The vocabulary for the Spanish language models was generated using tokens with frequency > 2. The perplexity for the forward and backward Spanish language models are 56 and 57 respectively.

3.4 Comparative performance

Table 4 shows the performance of our best systems compared to the state-of-the-art results on ConLL dataset. We focus our comparison to the systems with the same experimental setups (standard train/val/test split, without the use of external label data). The best previous systems (Ma and Hovy, 2016; Lample et al., 2016) are based upon a similar architecture (CRF-RNN) to ours. Lample et al. (2016) employed LSTM for character-based embedding, while Ma and Hovy (2016) employed CNN for character-based embedding[8]. Overall, we achieve state-of-the-art results on both English and Spanish.

[3] https://github.com/3Top/word2vec-api/blob/master/README.md

[4] https://catalog.ldc.upenn.edu/ldc2011t12

[5] https://github.com/tensorflow/models/tree/master/lm_1b

[6] https://github.com/ciprian-chelba/1-billion-word-language-modeling-benchmark

[7] https://github.com/rafaljozefowicz/lm

[8] There are several other works reporting very strong result on English NER: Chiu et al. (91.62) (2015), Yang et al. (91.20) (2016) and Peter et al.(91.93) (2017), however, these results are not comparable to ours due to the difference in experimental setup (Ma and Hovy, 2016).

System	F1 on English	F1 on Spanish
3 Res-RNN	91.07 ⋆	86.24⋆
3 Res-RNN + foreLM	91.43 ⋆	86.13⋆
3 Res-RNN + foreLM + backLM	91.66 ⋆	85.83
3 Res-RNN + bias	91.23 ⋆†	**86.31** ⋆
3 Res-RNN + foreLM + bias	91.45 ⋆	86.14 ⋆
3 Res-RNN + foreLM + backLM + bias	**91.69** ⋆	86.00 ⋆†

Table 3: Analysis of the bias decoding. ⋆ indicates significance ($p < 0.05$) versus CRF-RNN no Stack Residual in Table 1. † indicates significance versus the configuration with no bias.

System	F1 on English	F1 on Spanish
CRF-RNN no Stack Residual	90.43	85.41
(Passos et al., 2014)	90.05	–
(dos Santos and Guimarães, 2015)	–	82.21
(Gillick et al., 2016)	84.57	81.83
(Lample et al., 2016)	90.94	85.75
(Ma and Hovy, 2016)	91.21	–
3 Res-RNN + bias	91.23	**86.31**
3 Res-RNN + foreLM + bias	91.45	86.14
3 Res-RNN + foreLM + backLM + bias	**91.69**	86.00

Table 4: Compare our model with systems with comparable experimental settings

4 Discussion

Overall, our model achieves the state-of-the-arts for both English and Spanish Named Entity Recognition. For Spanish, our base model with three layers of Stacked Residual RNN already outperforms the current state-of-the-art.

From the results above, we can see that our innovations, the Stacked Residual connection and bias decoding consistently improve the performance across both data sets. However, the improvements from bias decoding is somewhat small in some models. The numerical gradient for training is noisy, and sometimes the SGD process might take several epochs to find an improvement on the development set. This happens especially on the English dataset because the base model trained with sequence level log-likelihood fits very well on the training set. Even with the boosting trick presented during the Experiments section, the training is still very slow. At first, we expected that the biases might give us some ideas about the trade-off between precision and recall similar to the thresholding technique for binary classification, i.e. the based log-likelihood model might favors precision or recall. However, from the analysis of the biases, we found no obvious trends favoring precision or recall.

Interestingly, the Language Model embeddings seem to have opposite effects on Spanish and English. While it is very helpful in English, it only degrades the performance for Spanish. The English LMs also improve convergence rate, while it is the opposite for Spanish. We attribute this difference in the quality of the Language Model involved. For English, the LMs are arguably better, with much lower perplexities than the LMs for Spanish. The Spanish models also have less data to train with, and it might affect the performance.

5 Conclusions and Future Work

We have explored two innovations over the baseline CRF-RNN model for sequence classification: the Stacked Residual Connection, and bias decoding. With these two improvements, it is possible to achieve state-of-the-art performance in Named Entity Recognition for both English and Spanish.

As future work, we will further investigate trainable bias decoding, and try to solve the problems presented. As the methods presented are general and language/domain independent, we plan to apply it to other domains such as health-care and expand the applications beyond NER.

References

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*. pages 173–182.

Masayuki Asahara and Yuji Matsumoto. 2003. Japanese named entity extraction with redundant morphological analysis. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, pages 8–15.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Daniel M Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. 1997. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*. Association for Computational Linguistics, pages 194–201.

Robert Busa-Fekete, Baiázs Szörényi, Krzysztof Dembczyński, and Eyke Hüllermeier. 2015. Online f-measure optimization. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*. MIT Press, Cambridge, MA, USA, NIPS'15, pages 595–603.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005* .

Jason PC Chiu and Eric Nichols. 2015. Named entity recognition with bidirectional lstm-cnns. *arXiv preprint arXiv:1511.08308* .

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* .

Paul R Cohen. 1996. Empirical methods for artificial intelligence. *IEEE Intelligent Systems* (6):88.

Cícero Nogueira dos Santos and Victor Guimarães. 2015. Boosting named entity recognition with neural character embeddings. *CoRR* abs/1505.05008.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2016. Multilingual language processing from bytes. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 1296–1306.

Jiatao Gu, Kyunghyun Cho, and Victor OK Li. 2017. Trainable greedy decoding for neural machine translation. *arXiv preprint arXiv:1702.02429* .

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 770–778.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410* .

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent convolutional neural networks for discourse compositionality. *arXiv preprint arXiv:1306.3584* .

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*. Association for Computational Linguistics, San Diego, California, pages 260–270.

Zachary C Lipton, Charles Elkan, and Balakrishnan Naryanaswamy. 2014. Optimal thresholding of classifiers to maximize f1 measure. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pages 225–239.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354* .

Andrew McCallum and Wei Li. 2003. 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*. Association for Computational Linguistics, pages 188–191.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*. volume 2, page 3.

David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes* 30(1):3–26.

Alexandre Passos, Vineet Kumar, and Andrew McCallum. 2014. Lexicon infused phrase embeddings for named entity resolution. *arXiv preprint arXiv:1404.5367* .

Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108* .

Aaditya Prakash, Sadid A Hasan, Kathy Lee, Vivek Datla, Ashequl Qadir, Joey Liu, and Oladimeji Farri. 2016. Neural paraphrase generation with stacked residual lstm networks. *arXiv preprint arXiv:1610.03098* .

Lisa F Rau. 1991. Extracting company names from text. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on*. IEEE, volume 1, pages 29–32.

Satoshi Sekine and Chikashi Nobata. 2004. Definition, dictionaries and tagger for extended named entity hierarchy. In *LREC*. pages 1977–1980.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. Lstm neural networks for language modeling. In *Interspeech*. pages 194–197.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. 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*. Association for Computational Linguistics, pages 142–147.

Quan Hung Tran, Ingrid Zukerman, and Gholamreza Haffari. 2017. A hierarchical neural model for learning sequences of dialogue acts. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Association for Computational Linguistics, Valencia, Spain, pages 428–437.

Wenduan Xu, Michael Auli, and Stephen Clark. 2016. Expected f-measure training for shift-reduce parsing with recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 210–220.

Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. 2016. Multi-task cross-lingual sequence tagging from scratch. *arXiv preprint arXiv:1603.06270* .