Named Entity Recognition With Parallel Recurrent Neural Networks

Andrej Žukov-Gregorič*,†, Yoram Bachrach*,†, and Sam Coope*

†DigitalGenius, 1 Canada Square, London E14 5AB
‡Department of Computer Science, Royal Holloway, University of London, Egham TW20 0EX

andrej.zukovgregoric.2010@live.rhul.ac.uk
yorambac@gmail.com
sam@digitalgenius.com

Abstract
We present a new architecture for named entity recognition. Our model employs multiple independent bidirectional LSTM units across the same input and promotes diversity among them by employing an inter-model regularization term. By distributing computation across multiple smaller LSTMs we find a reduction in the total number of parameters. We find our architecture achieves state-of-the-art performance on the CoNLL 2003 NER dataset.

1 Introduction
The ability to reason about entities in text is an important element of natural language understanding. Named entity recognition (NER) concerns itself with the identification of such entities. Given a sequence of words, the task of NER is to label each word with its appropriate corresponding entity type. Examples of entity types include Person, Organization, and Location. A special Other entity type is often added to the set of all types and is used to label words which do not belong to any of the other entity types.

Recently, neural network based approaches which use no language-specific resources, apart from unlabeled corpora for training word embeddings, have emerged. There has been a shift of focus from handcrafting better features to designing better neural architectures for solving NER.

In this paper, we propose a new parallel recurrent neural network model for entity recognition. We show that rather than using a single LSTM component, as many other recent architecture have, we instead resort to using multiple smaller LSTM units. This has the benefit of reducing the total number of parameters in our model. We present results on the CoNLL 2003 English dataset and achieve the new state of the art results for models without help from an outside lexicons.

1.1 Related Work
Various approaches have been proposed to NER. Many of these approaches rely on handcrafted feature engineering or language-specific or domain-specific resources (Zhou and Su, 2002; Chieu and Ng, 2002; Florian et al., 2003; Settles, 2004; Nadeau and Sekine, 2007). While such approaches can achieve high accuracy, they may fail to generalize to new languages, new corpora or new types of entities to be identified. Thus, applying such techniques in new domains requires making a heavy engineering investment.

Over time neural methods such as (Chiu and Nichols, 2015; Ma and Hovy, 2016; Luo et al., 2015; Lample et al., 2016) emerged. More recently (Peters et al., 2017; Reimers and Gurevych, 2017; Sato et al., 2017) have set the top benchmarks in the field.

Architecturally, our model is similar to those of (Zhu et al., 2017; Hidasi et al., 2016) with the most pronounced difference being that we (1) apply our parallel RNN units across the same input (2) explore a new regularization term for promoting diversity across what features our parallel RNNs extract and (3) explicitly motivate the architecture with a discussion about parameter complexity.

The need for a wider discussion on parameter complexity in the deep learning community is being pushed by the need to make complex neural models runnable in constrained environment such as field-programmable gate arrays (FPGAs) - for a great discussion relating to running LSTMs on FPGAs see (Guan et al., 2017). Additionally, complex models have proven difficult to use in certain

† Now at Google DeepMind, 6 Pancras Square, London N1C 4AG.
domains such as embedded systems or finance due to their slowness. Our architecture lends itself to parallelization and attempts to tackle this problem.

2 Named Entity Recognition

Named Entity Recognition can be posited as a standard sequence classification problem where the dataset \( D = \{ (X_i, y_i) \}_{i=1}^{k} \) consists of example label pairs where both the examples and the labels are themselves sequences of word vectors and entity types, respectively.

Specifically, an input example \( X_i = (x_{i,1}, \ldots, x_{i,|X_i|}) \) is a variable-length sequence of word vectors \( x_{i,j} \in \mathbb{R}^d \); the example’s corresponding label \( y_i = (y_{i,1}, \ldots, y_{i,|X_i|}) \) is an equal-length sequence of entity-type labels \( y_{i,j} \in \mathcal{Y} \) where \( \mathcal{Y} \) is the set of all entity type labels and includes a special other ‘O’-label with which all words that are not entities are labeled.

The goal is then to learn a parametrized mapping \( f_\theta : X \rightarrow y \) from input words to output entity labels. One of the most commonly used class of models that handle this mapping are recurrent neural networks.

2.1 LSTM complexity

Long short term memory (LSTM) models belong to the family of recurrent neural network (RNN) models. They are often used as a component of much larger models, particularly in many NLP tasks including NER.

Classically, an LSTM cell is defined as follows (biases excluded for brevity):

\[
\begin{align*}
i_t &= \sigma(W_i h_{t-1} + U_i x_t) \\
f_t &= \sigma(W_f h_{t-1} + U_f x_t) \\
o_t &= \sigma(W_o h_{t-1} + U_o x_t) \\
c_t &= \tanh(W_c h_{t-1} + U_c x_t) \\
h_t &= o_t \odot \tanh(c_t)
\end{align*}
\]

One way of measuring the complexity of a model is through its total number of parameters. Looking at the above, we note there are two parameter matrices, \( W \) and \( U \), for each of the three input gates and during cell update. If we let \( W \in \mathbb{R}^{n \times n} \) and \( U \in \mathbb{R}^{n \times m} \) then the total number of parameters in the model (excluding the bias terms) is \( 4(nm+n^2) \) which grows quadratically as \( n \) grows. Thus, increases in LSTM size can substantially increase the number of parameters.

3 Parallel RNNs

To reduce the total number of parameters we split a single LSTM into multiple equally-sized smaller ones:

\[
h_{k,t} = \text{LSTM}_k(h_{k,t-1}, x)
\]

where \( k \in \{1, \ldots, K\} \). This has the effect of dividing the total number of parameters by a constant factor. The final hidden state \( h_t \) is then a concatenation of the hidden states of the smaller LSTMs:

\[
h_t = [h_{1,t}; h_{2,t}; \ldots; h_{K,t}]
\]

3.1 Promoting Diversity

To promote diversity amongst the constituent smaller LSTMs we add an orthogonality penalty across the smaller LSTMs. Recent research has used similar methods but applied to single LSTMs (Vorontsov et al., 2017).

We take the cell update recurrence parameters \( W_i \) across LSTMs (we omit the \( c \) in the subscript for brevity; the index \( i \) runs across the smaller LSTMs) and for any pair we wish the following to be true:

\[
\langle \text{vec}(W_{c}^{(i)}), \text{vec}(W_{c}^{(j)}) \rangle \approx 0
\]

To achieve this we pack the vectorized parameters into a matrix:

\[
\Phi = \begin{pmatrix}
\text{vec}(W_{c}^{(1)}) \\
\text{vec}(W_{c}^{(2)}) \\
\vdots \\
\text{vec}(W_{c}^{(N)})
\end{pmatrix}
\]

and apply the following regularization term to our final loss:

\[
\lambda \sum_i \| \Phi \Phi^\top - I \|_F^2 (1)
\]

3.2 Output and Loss

The concatenated output \( h_t \) is passed through a fully connected layer with bias before being passed through a final softmax layer:

\[
o_t = \text{softmax}(W_{out} h_t + b_{out})
\]
To extract a predicted entity type \( \hat{y}_t \) at time \( t \), we select the entity type corresponding to the most probable output:

\[
\hat{y}_t = \text{argmax}(o_t)
\]

The loss is defined as the sum of the softmax cross-entropy losses along the words in the input sequence. More precisely, we denote by \( y_t^j \in \{0, 1\} \) a binary indicator variable indicating whether word \( x_t \) truly is an entity of type \( j \). The loss at time \( t \) is then defined to be \( L_t = -\sum_j y_t^j \log(o_t^j) \). Thus the overall loss is:

\[
L = -\sum_t \sum_j y_t^j \log(o_t^j)
\]

### 3.3 Implementation Details

We use bidirectional LSTMs as our base recurrent unit and use pretrained word embeddings of size 100. These are the same embeddings used in (Lample et al., 2016). We concatenate to our word embeddings character-level embeddings similar to (Lample et al., 2016) but with a max pooling layer instead. Unlike with the parallel LSTMs, we only use a single character embedding LSTM.

Parameters are initialized using the method described by Glorot and Bengio (Glorot and Bengio, 2010). This approach scales the variance of a uniform distribution with regard to the root of the number of parameters in a layer. This approach has been found to speed up convergence compared to using a unit normal distribution for initialization.

Our model uses variational dropout (Gal and Ghahramani, 2016) between the hidden states of the parallel LSTMs. Recent work has shown this to be very effective at training LSTMs for language models (Merity et al., 2017). In our experiments, we use \( p = 0.1 \) as our dropping probability.

We experiment with different values of the regularization term parameter but settled on \( \lambda = 0.01 \).

Although vanilla stochastic gradient descent has been effective at training RNNs on language problems (Merity et al., 2017), we found that using the ADAM optimizer (Kingma and Ba, 2014) to be more effective at training our model. We experimented with different values for the learning rate \( \alpha \), increasing \( \alpha \) from \( 10^{-3} \) to as high as \( 5 \times 10^{-3} \) and still obtained good results.

Similarly, we kept a constant size for the character-level embeddings, using a unit bidirectional LSTM output size of \( \text{dim}(\text{char}) = 50 \).

As previously discussed, we trained the network parameters using stochastic gradient descent (Werbos, 1990), augmented with the Adam optimizer (Kingma and Ba, 2014).

### 3.4 Relation to Ensemble Methods

Our model bears some resemblance to ensemble methods (Freund et al., 1996; Dietterich et al., 2000), which combine multiple “weak learners” into a single “strong learner”; One may view each of the parallel recurrent units of our model as a single “weak” neural network, and may consider our architecture as a way of combining these into a single “strong” network.

Despite the similarities, our model is very different from ensemble methods. First, as opposed to many boosting algorithms (Freund et al., 1996; Schapire and Singer, 1999; Dietterich et al., 2000) we do not “reweigh” training instances based on the loss incurred on them by a previous iteration. Second, unlike ensemble methods, our model is trained end-to-end, as a single large neural network. All the subcomponents are co-trained, so different subparts of the network may focus on different aspects of the input. This avoids redundant repeated computations across the units (and indeed, we encourage diversity between the units using our inter-module regularization). Finally, we note that our architecture does not simply combine the prediction of multiple classifiers; rather, we take the final hidden layer of each of the LSTM units (which contains more information than merely the entity class prediction), and combine this information using a feedforward network. This allows our architecture to examine inter-dependencies between pieces of information computed by the various components.

### 4 Experiments

We achieve state-of-the-art results on the CoNNL 2003 English NER dataset (see Table 1). Although we do not employ additional external resources (language specific dictionaries or gazetteers), our model is competitive even with some of the models that do.

To gain a better understanding of the performance of our model including how its various components affect performance we prepared four additional tables of runs.

Table 2 shows performance as a function of the number of RNN units with a fixed unit size. The
Table 1: English NER F1 score of our model on the test set of CoNLL-2003 (English). During training we optimize for the development set and report test set results for our best performing development set model. The bounded F1 results we report (±0.22) are taken after 10 runs. For the purpose of comparison, we also list F1 scores of previous top-performance systems. ‡ marks the neural models. ∗ marks model which use external resources.

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Chieu and Ng, 2002)</td>
<td>88.31</td>
</tr>
<tr>
<td>(Florian et al., 2003)</td>
<td>88.76</td>
</tr>
<tr>
<td>(Ando and Zhang, 2005)</td>
<td>89.31</td>
</tr>
<tr>
<td>(Collobert et al., 2011)†</td>
<td>89.59</td>
</tr>
<tr>
<td>(Huang et al., 2015)‡</td>
<td>90.10</td>
</tr>
<tr>
<td>(Chiu and Nichols, 2015)‡</td>
<td>90.77</td>
</tr>
<tr>
<td>(Ratinov and Roth, 2009)</td>
<td>90.80</td>
</tr>
<tr>
<td>(Lin and Wu, 2009)</td>
<td>90.90</td>
</tr>
<tr>
<td>(Passos et al., 2014)‡*</td>
<td>90.90</td>
</tr>
<tr>
<td>(Lample et al., 2016)‡</td>
<td>90.94</td>
</tr>
<tr>
<td>(Luo et al., 2015)‡</td>
<td>91.20</td>
</tr>
<tr>
<td>(Ma and Hovy, 2016)‡</td>
<td>91.21</td>
</tr>
<tr>
<td>(Sato et al., 2017)</td>
<td>91.28</td>
</tr>
<tr>
<td>(Chiu and Nichols, 2015)‡*</td>
<td>91.62</td>
</tr>
<tr>
<td>(Peters et al., 2017)‡*</td>
<td>91.93</td>
</tr>
<tr>
<td>This paper†</td>
<td>91.48 ±0.22</td>
</tr>
</tbody>
</table>

Table 2: Performance as a function of the number of RNN units with a fixed unit size of 64; averaged across 5 runs apart from the 16 unit (average across 10 runs).

<table>
<thead>
<tr>
<th># RNN units</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90.53 ±0.31</td>
</tr>
<tr>
<td>2</td>
<td>90.79 ±0.18</td>
</tr>
<tr>
<td>4</td>
<td>90.64 ±0.24</td>
</tr>
<tr>
<td>8</td>
<td>91.09 ±0.28</td>
</tr>
<tr>
<td>16</td>
<td>91.48 ±0.22</td>
</tr>
<tr>
<td>32</td>
<td>90.68 ±0.18</td>
</tr>
</tbody>
</table>

Table 3: Performance of our model with various unit sizes resulting in a fixed final output size $h_t$. Single runs apart from 16 unit.

<table>
<thead>
<tr>
<th>Unit size</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>89.78</td>
</tr>
<tr>
<td>16</td>
<td>89.77</td>
</tr>
<tr>
<td>32</td>
<td>90.26</td>
</tr>
<tr>
<td>64</td>
<td>91.48 ±0.22</td>
</tr>
<tr>
<td>128</td>
<td>89.28</td>
</tr>
</tbody>
</table>

Table 4: Performance as a function of the unit size for our best performing model (16 biLSTM units). Single runs apart from with size 64.

<table>
<thead>
<tr>
<th>Component</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No character embeddings</td>
<td>90.39</td>
</tr>
<tr>
<td>No orthogonal regularization</td>
<td>90.79</td>
</tr>
<tr>
<td>No Xavier initialization</td>
<td>91.09</td>
</tr>
<tr>
<td>No variational dropout</td>
<td>91.03</td>
</tr>
<tr>
<td>Mean pool instead of concat</td>
<td>90.49</td>
</tr>
</tbody>
</table>

Table 5: Impact of various architectural decisions on our best performing model (16 biLSTM units, 64 unit size). Single runs.

5 Conclusion

We achieve state-of-the-art results on the CoNLL 2003 English dataset and introduce a new model motivated primarily by its ability to be easily distributable and reduce the total number of parameters. Further work should be done on evaluating it across different classification and sequence classification tasks to study its performance. Additionally, a run-time analysis show be conducted to compare speedups if the model is parallelized across CPU cores.
References


