Neural Architectures for Named Entity Recognition

Guillaume Lample♠ **Miguel Ballesteros**♣♠
Sandeep Subramanian♠ **Kazuya Kawakami**♠ **Chris Dyer**♠
♠Carnegie Mellon University ♣NLP Group, Pompeu Fabra University
{glample,sandeeps,kkawakam,cdyer}@cs.cmu.edu,
miguel.ballesteros@upf.edu

Abstract

State-of-the-art named entity recognition systems rely heavily on hand-crafted features and domain-specific knowledge in order to learn effectively from the small, supervised training corpora that are available. In this paper, we introduce two new neural architectures—one based on bidirectional LSTMs and conditional random fields, and the other that constructs and labels segments using a transition-based approach inspired by shift-reduce parsers. Our models rely on two sources of information about words: character-based word representations learned from the supervised corpus and unsupervised word representations learned from unannotated corpora. Our models obtain state-of-the-art performance in NER in four languages without resorting to any language-specific knowledge or resources such as gazetteers. [1]

1 Introduction

Named entity recognition (NER) is a challenging learning problem. One the one hand, in most languages and domains, there is only a very small amount of supervised training data available. On the other, there are few constraints on the kinds of words that can be names, so generalizing from this small sample of data is difficult. As a result, carefully constructed orthographic features and language-specific knowledge resources, such as gazetteers, are widely used for solving this task. Unfortunately, language-specific resources and features are costly to develop in new languages and new domains, making NER a challenge to adapt. Unsupervised learning from unannotated corpora offers an alternative strategy for obtaining better generalization from small amounts of supervision. However, even systems that have relied extensively on unsupervised features (Collobert et al., 2011; Turian et al., 2010; Lin and Wu, 2009; Ando and Zhang, 2005b, *inter alia*) have used these to augment, rather than replace, hand-engineered features (e.g., knowledge about capitalization patterns and character classes in a particular language) and specialized knowledge resources (e.g., gazetteers).

In this paper, we present neural architectures for NER that use no language-specific resources or features beyond a small amount of supervised training data and unlabeled corpora. Our models are designed to capture two intuitions. First, since names often consist of multiple tokens, reasoning jointly over tagging decisions for each token is important. We compare two models here, (i) a bidirectional LSTM with a sequential conditional random layer above it (LSTM-CRF; §2), and (ii) a new model that constructs and labels chunks of input sentences using an algorithm inspired by transition-based parsing with states represented by stack LSTMs (S-LSTM; §3). Second, token-level evidence for "being a name" includes both orthographic evidence (what does the word being tagged as a name look like?) and distributional evidence (where does the word being tagged tend to occur in a corpus?). To capture orthographic sensitivity, we use character-based word representation model (Ling et al., 2015b) to capture distributional sensitivity, we combine these representations with distributional representations (Mikolov et al., 2013b). Our word representations combine both of these, and dropout training is used to encourage the model to learn to trust both sources of evidence (§4).

Experiments in English, Dutch, German, and Spanish show that we are able to obtain state-

[1]The code of the LSTM-CRF and Stack-LSTM NER systems are available at `https://github.com/glample/tagger` and `https://github.com/clab/stack-lstm-ner`

of-the-art NER performance with the LSTM-CRF model in Dutch, German, and Spanish, and very near the state-of-the-art in English without any hand-engineered features or gazetteers (§5). The transition-based algorithm likewise surpasses the best previously published results in several languages, although it performs less well than the LSTM-CRF model.

2 LSTM-CRF Model

We provide a brief description of LSTMs and CRFs, and present a hybrid tagging architecture. This architecture is similar to the ones presented by Collobert et al. (2011) and Huang et al. (2015).

2.1 LSTM

Recurrent neural networks (RNNs) are a family of neural networks that operate on sequential data. They take as input a sequence of vectors $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ and return another sequence $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n)$ that represents some information about the sequence at every step in the input. Although RNNs can, in theory, learn long dependencies, in practice they fail to do so and tend to be biased towards their most recent inputs in the sequence (Bengio et al., 1994). Long Short-term Memory Networks (LSTMs) have been designed to combat this issue by incorporating a memory-cell and have been shown to capture long-range dependencies. They do so using several gates that control the proportion of the input to give to the memory cell, and the proportion from the previous state to forget (Hochreiter and Schmidhuber, 1997). We use the following implementation:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{c}_t &= (1 - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \\
&\qquad \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

where σ is the element-wise sigmoid function, and $\odot$ is the element-wise product.

For a given sentence $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ containing n words, each represented as a d-dimensional vector, an LSTM computes a representation $\overrightarrow{\mathbf{h}_t}$ of the left context of the sentence at every word t. Naturally, generating a representation of the right context $\overleftarrow{\mathbf{h}_t}$ as well should add useful information. This can be achieved using a second LSTM that reads the same sequence in reverse. We will refer to the former as the forward LSTM and the latter as the backward LSTM. These are two distinct networks with different parameters. This forward and backward LSTM pair is referred to as a bidirectional LSTM (Graves and Schmidhuber, 2005).

The representation of a word using this model is obtained by concatenating its left and right context representations, $\mathbf{h}_t = [\overrightarrow{\mathbf{h}_t}; \overleftarrow{\mathbf{h}_t}]$. These representations effectively include a representation of a word in context, which is useful for numerous tagging applications.

2.2 CRF Tagging Models

A very simple—but surprisingly effective—tagging model is to use the $\mathbf{h}_t$'s as features to make independent tagging decisions for each output y_t (Ling et al., 2015b). Despite this model's success in simple problems like POS tagging, its independent classification decisions are limiting when there are strong dependencies across output labels. NER is one such task, since the "grammar" that characterizes interpretable sequences of tags imposes several hard constraints (e.g., I-PER cannot follow B-LOC; see §2.4 for details) that would be impossible to model with independence assumptions.

Therefore, instead of modeling tagging decisions independently, we model them jointly using a conditional random field (Lafferty et al., 2001). For an input sentence

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n),$$

we consider $\mathbf{P}$ to be the matrix of scores output by the bidirectional LSTM network. $\mathbf{P}$ is of size $n \times k$, where k is the number of distinct tags, and $P_{i,j}$ corresponds to the score of the j^{th} tag of the i^{th} word in a sentence. For a sequence of predictions

$$\mathbf{y} = (y_1, y_2, \ldots, y_n),$$

we define its score to be

$$s(\mathbf{X}, \mathbf{y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}$$

where $\mathbf{A}$ is a matrix of transition scores such that $A_{i,j}$ represents the score of a transition from the tag i to tag j. y_0 and y_n are the *start* and *end* tags of a sentence, that we add to the set of possible tags. $\mathbf{A}$ is therefore a square matrix of size $k+2$.

A softmax over all possible tag sequences yields a probability for the sequence $\mathbf{y}$:

$$p(\mathbf{y}|\mathbf{X}) = \frac{e^{s(\mathbf{X},\mathbf{y})}}{\sum_{\widetilde{\mathbf{y}} \in \mathbf{Y_X}} e^{s(\mathbf{X},\widetilde{\mathbf{y}})}}.$$

During training, we maximize the log-probability of the correct tag sequence:

$$\begin{aligned} \log(p(\mathbf{y}|\mathbf{X})) &= s(\mathbf{X},\mathbf{y}) - \log\left(\sum_{\widetilde{\mathbf{y}} \in \mathbf{Y_X}} e^{s(\mathbf{X},\widetilde{\mathbf{y}})}\right) \\ &= s(\mathbf{X},\mathbf{y}) - \underset{\widetilde{\mathbf{y}} \in \mathbf{Y_X}}{\text{logadd}}\, s(\mathbf{X},\widetilde{\mathbf{y}}), \quad (1) \end{aligned}$$

where $\mathbf{Y_X}$ represents all possible tag sequences (even those that do not verify the IOB format) for a sentence $\mathbf{X}$. From the formulation above, it is evident that we encourage our network to produce a valid sequence of output labels. While decoding, we predict the output sequence that obtains the maximum score given by:

$$\mathbf{y}^* = \underset{\widetilde{\mathbf{y}} \in \mathbf{Y_X}}{\text{argmax}}\, s(\mathbf{X},\widetilde{\mathbf{y}}). \quad (2)$$

Since we are only modeling bigram interactions between outputs, both the summation in Eq. 1 and the maximum a posteriori sequence $\mathbf{y}^*$ in Eq. 2 can be computed using dynamic programming.

2.3 Parameterization and Training

The scores associated with each tagging decision for each token (i.e., the $P_{i,y}$'s) are defined to be the dot product between the embedding of a word-in-context computed with a bidirectional LSTM—exactly the same as the POS tagging model of Ling et al. (2015b) and these are combined with bigram compatibility scores (i.e., the $A_{y,y'}$'s). This architecture is shown in figure 1. Circles represent observed variables, diamonds are deterministic functions of their parents, and double circles are random variables.

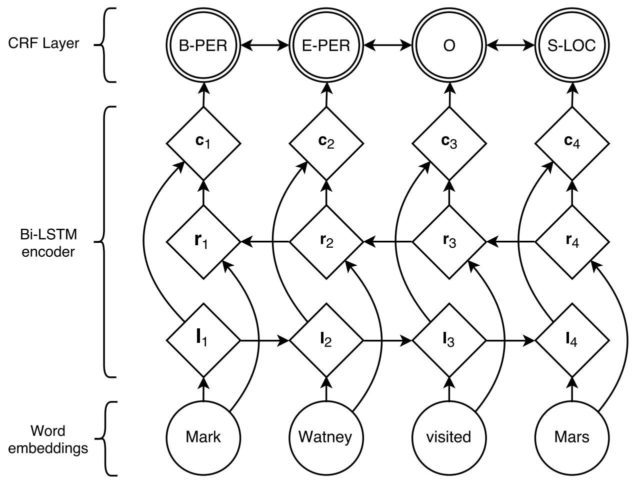

Figure 1: Main architecture of the network. Word embeddings are given to a bidirectional LSTM. $\mathbf{l}_i$ represents the word i and its left context, $\mathbf{r}_i$ represents the word i and its right context. Concatenating these two vectors yields a representation of the word i in its context, $\mathbf{c}_i$.

The parameters of this model are thus the matrix of bigram compatibility scores $\mathbf{A}$, and the parameters that give rise to the matrix $\mathbf{P}$, namely the parameters of the bidirectional LSTM, the linear feature weights, and the word embeddings. As in part 2.2, let $\mathbf{x}_i$ denote the sequence of word embeddings for every word in a sentence, and y_i be their associated tags. We return to a discussion of how the embeddings $\mathbf{x}_i$ are modeled in Section 4. The sequence of word embeddings is given as input to a bidirectional LSTM, which returns a representation of the left and right context for each word as explained in 2.1.

These representations are concatenated ($\mathbf{c}_i$) and linearly projected onto a layer whose size is equal to the number of distinct tags. Instead of using the softmax output from this layer, we use a CRF as previously described to take into account neighboring tags, yielding the final predictions for every word y_i. Additionally, we observed that adding a hidden layer between $\mathbf{c}_i$ and the CRF layer marginally improved our results. All results reported with this model incorporate this extra-layer. The parameters are trained to maximize Eq. 1 of observed sequences of NER tags in an annotated corpus, given the observed words.

2.4 Tagging Schemes

The task of named entity recognition is to assign a named entity label to every word in a sentence. A single named entity could span several tokens within a sentence. Sentences are usually represented in the IOB format (Inside, Outside, Beginning) where every token is labeled as B-*label* if the token is the beginning of a named entity, I-*label* if it is inside a named entity but not the first token within the named entity, or O otherwise. However, we decided to use the IOBES tagging scheme, a variant of IOB commonly used for named entity recognition, which encodes information about singleton entities (S) and explicitly marks the end of named entities (E). Using this scheme, tagging a word as I-*label* with high-confidence narrows down the choices for the subsequent word to I-*label* or E-*label*, however, the IOB scheme is only capable of determining that the subsequent word cannot be the interior of another label. Ratinov and Roth (2009) and Dai et al. (2015) showed that using a more expressive tagging scheme like IOBES improves model performance marginally. However, we did not observe a significant improvement over the IOB tagging scheme.

3 Transition-Based Chunking Model

As an alternative to the LSTM-CRF discussed in the previous section, we explore a new architecture that chunks and labels a sequence of inputs using an algorithm similar to transition-based dependency parsing. This model directly constructs representations of the multi-token names (e.g., the name *Mark Watney* is composed into a single representation).

This model relies on a stack data structure to incrementally construct chunks of the input. To obtain representations of this stack used for predicting subsequent actions, we use the Stack-LSTM presented by Dyer et al. (2015), in which the LSTM is augmented with a "stack pointer." While sequential LSTMs model sequences from left to right, stack LSTMs permit embedding of a stack of objects that are both added to (using a push operation) and removed from (using a pop operation). This allows the Stack-LSTM to work like a stack that maintains a "summary embedding" of its contents. We refer to this model as Stack-LSTM or S-LSTM model for simplicity.

Finally, we refer interested readers to the original paper (Dyer et al., 2015) for details about the Stack-LSTM model since in this paper we merely use the same architecture through a new transition-based algorithm presented in the following Section.

3.1 Chunking Algorithm

We designed a transition inventory which is given in Figure 2 that is inspired by transition-based parsers, in particular the arc-standard parser of Nivre (2004). In this algorithm, we make use of two stacks (designated *output* and *stack* representing, respectively, completed chunks and scratch space) and a *buffer* that contains the words that have yet to be processed. The transition inventory contains the following transitions: The SHIFT transition moves a word from the buffer to the stack, the OUT transition moves a word from the buffer directly into the output stack while the REDUCE(y) transition pops all items from the top of the stack creating a "chunk," labels this with label y, and pushes a representation of this chunk onto the output stack. The algorithm completes when the stack and buffer are both empty. The algorithm is depicted in Figure 2, which shows the sequence of operations required to process the sentence *Mark Watney visited Mars*.

The model is parameterized by defining a probability distribution over actions at each time step, given the current contents of the stack, buffer, and output, as well as the history of actions taken. Following Dyer et al. (2015), we use stack LSTMs to compute a fixed dimensional embedding of each of these, and take a concatenation of these to obtain the full algorithm state. This representation is used to define a distribution over the possible actions that can be taken at each time step. The model is trained to maximize the conditional probability of sequences of reference actions (extracted from a labeled training corpus) given the input sentences. To label a new input sequence at test time, the maximum probability action is chosen greedily until the algorithm reaches a termination state. Although this is not guaranteed to find a global optimum, it is effective in practice. Since each token is either moved directly to the output (1 action) or first to the stack and then the output (2 actions), the total number of actions for a sequence of length n is maximally $2n$.

It is worth noting that the nature of this algorithm

$\mathbf{Out}_t$	$\mathbf{Stack}_t$	$\mathbf{Buffer}_t$	**Action**	$\mathbf{Out}_{t+1}$	$\mathbf{Stack}_{t+1}$	$\mathbf{Buffer}_{t+1}$	**Segments**
O	S	$(\mathbf{u}, u), B$	SHIFT	O	$(\mathbf{u}, u), S$	B	—
O	$(\mathbf{u}, u), \dots, (\mathbf{v}, v), S$	B	REDUCE(y)	$g(\mathbf{u}, \dots, \mathbf{v}, \mathbf{r}_y), O$	S	B	$(u \dots v, y)$
O	S	$(\mathbf{u}, u), B$	OUT	$g(\mathbf{u}, \mathbf{r}_\varnothing), O$	S	B	—

Figure 2: Transitions of the Stack-LSTM model indicating the action applied and the resulting state. Bold symbols indicate (learned) embeddings of words and relations, script symbols indicate the corresponding words and relations.

Transition	**Output**	**Stack**	**Buffer**	**Segment**
	[]	[]	[Mark, Watney, visited, Mars]	
SHIFT	[]	[Mark]	[Watney, visited, Mars]	
SHIFT	[]	[Mark, Watney]	[visited, Mars]	
REDUCE(PER)	[(Mark Watney)-PER]	[]	[visited, Mars]	(Mark Watney)-PER
OUT	[(Mark Watney)-PER, visited]	[]	[Mars]	
SHIFT	[(Mark Watney)-PER, visited]	[Mars]	[]	
REDUCE(LOC)	[(Mark Watney)-PER, visited, (Mars)-LOC]	[]	[]	(Mars)-LOC

Figure 3: Transition sequence for *Mark Watney visited Mars* with the Stack-LSTM model.

model makes it agnostic to the tagging scheme used since it directly predicts labeled chunks.

3.2 Representing Labeled Chunks

When the REDUCE(y) operation is executed, the algorithm shifts a sequence of tokens (together with their vector embeddings) from the stack to the output buffer as a single completed chunk. To compute an embedding of this sequence, we run a bidirectional LSTM over the embeddings of its constituent tokens together with a token representing the type of the chunk being identified (i.e., y). This function is given as $g(\mathbf{u}, \dots, \mathbf{v}, \mathbf{r}_y)$, where $\mathbf{r}_y$ is a learned embedding of a label type. Thus, the output buffer contains a single vector representation for each labeled chunk that is generated, regardless of its length.

4 Input Word Embeddings

The input layers to both of our models are vector representations of individual words. Learning independent representations for word types from the limited NER training data is a difficult problem: there are simply too many parameters to reliably estimate. Since many languages have orthographic or morphological evidence that something is a name (or not a name), we want representations that are sensitive to the spelling of words. We therefore use a model that constructs representations of words from representations of the characters they are composed of (4.1). Our second intuition is that names, which may individually be quite varied, appear in regular contexts in large corpora. Therefore we use embeddings learned from a large corpus that are sensitive to word order (4.2). Finally, to prevent the models from depending on one representation or the other too strongly, we use dropout training and find this is crucial for good generalization performance (4.3).

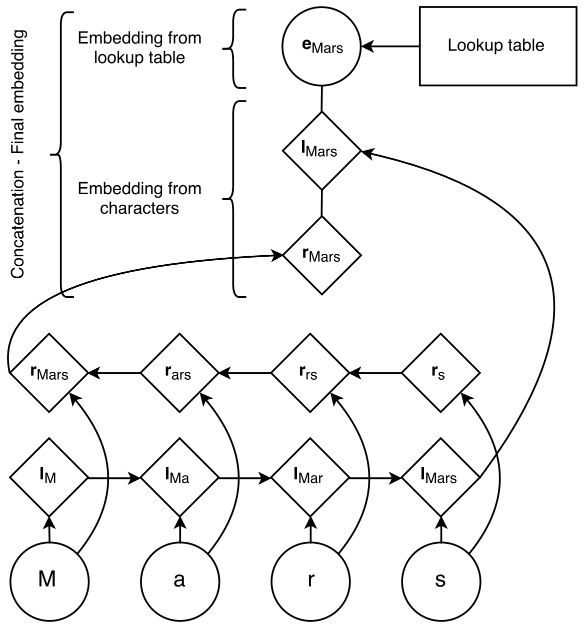
Concatenation - Final embedding
Embedding from lookup table
Embedding from characters
$\mathbf{e}_{\mathrm{Mars}}$
Lookup table
$\mathbf{l}_{\mathrm{Mars}}$
$\mathbf{r}_{\mathrm{Mars}}$
$\mathbf{r}_{\mathrm{Mars}}$
$\mathbf{r}_{\mathrm{ars}}$
$\mathbf{r}_{\mathrm{rs}}$
$\mathbf{r}_{\mathrm{s}}$
$\mathbf{l}_{\mathrm{M}}$
$\mathbf{l}_{\mathrm{Ma}}$
$\mathbf{l}_{\mathrm{Mar}}$
$\mathbf{l}_{\mathrm{Mars}}$
M
a
r
s

Figure 4: The character embeddings of the word "Mars" are given to a bidirectional LSTMs. We concatenate their last outputs to an embedding from a lookup table to obtain a representation for this word.

4.1 Character-based models of words

An important distinction of our work from most previous approaches is that we learn character-level features while training instead of hand-engineering prefix and suffix information about words. Learning character-level embeddings has the advantage of learning representations specific to the task and domain at hand. They have been found useful for morphologically rich languages and to handle the out-of-vocabulary problem for tasks like part-of-speech tagging and language modeling (Ling et al., 2015b) or dependency parsing (Ballesteros et al., 2015).

Figure 4 describes our architecture to generate a word embedding for a word from its characters. A character lookup table initialized at random contains an embedding for every character. The character embeddings corresponding to every character in a word are given in direct and reverse order to a forward and a backward LSTM. The embedding for a word derived from its characters is the concatenation of its forward and backward representations from the bidirectional LSTM. This character-level representation is then concatenated with a word-level representation from a word lookup-table. During testing, words that do not have an embedding in the lookup table are mapped to a UNK embedding. To train the UNK embedding, we replace singletons with the UNK embedding with a probability 0.5. In all our experiments, the hidden dimension of the forward and backward character LSTMs are 25 each, which results in our character-based representation of words being of dimension 50.

Recurrent models like RNNs and LSTMs are capable of encoding very long sequences, however, they have a representation biased towards their most recent inputs. As a result, we expect the final representation of the forward LSTM to be an accurate representation of the suffix of the word, and the final state of the backward LSTM to be a better representation of its prefix. Alternative approaches—most notably like convolutional networks—have been proposed to learn representations of words from their characters (Zhang et al., 2015; Kim et al., 2015). However, convnets are designed to discover position-invariant features of their inputs. While this is appropriate for many problems, e.g., image recognition (a cat can appear anywhere in a picture), we argue that important information is position dependent (e.g., prefixes and suffixes encode different information than stems), making LSTMs an *a priori* better function class for modeling the relationship between words and their characters.

4.2 Pretrained embeddings

As in Collobert et al. (2011), we use pretrained word embeddings to initialize our lookup table. We observe significant improvements using pretrained word embeddings over randomly initialized ones. Embeddings are pretrained using skip-n-gram (Ling et al., 2015a), a variation of word2vec (Mikolov et al., 2013a) that accounts for word order. These embeddings are fine-tuned during training.

Word embeddings for Spanish, Dutch, German and English are trained using the Spanish Gigaword version 3, the Leipzig corpora collection, the German monolingual training data from the 2010 Machine Translation Workshop and the English Gigaword version 4 (with the LA Times and NY Times portions removed) respectively.[2] We use an embedding dimension of 100 for English, 64 for other languages, a minimum word frequency cutoff of 4, and a window size of 8.

4.3 Dropout training

Initial experiments showed that character-level embeddings did not improve our overall performance when used in conjunction with pretrained word representations. To encourage the model to depend on both representations, we use dropout training (Hinton et al., 2012), applying a dropout mask to the final embedding layer just before the input to the bidirectional LSTM in Figure 1. We observe a significant improvement in our model's performance after using dropout (see table 5).

5 Experiments

This section presents the methods we use to train our models, the results we obtained on various tasks and the impact of our networks' configuration on model performance.

5.1 Training

For both models presented, we train our networks using the back-propagation algorithm updating our parameters on every training example, one at a time, using stochastic gradient descent (SGD) with

[2](Graff, 2011; Biemann et al., 2007; Callison-Burch et al., 2010; Parker et al., 2009)

a learning rate of 0.01 and a gradient clipping of 5.0. Several methods have been proposed to enhance the performance of SGD, such as Adadelta (Zeiler, 2012) or Adam (Kingma and Ba, 2014). Although we observe faster convergence using these methods, none of them perform as well as SGD with gradient clipping.

Our LSTM-CRF model uses a single layer for the forward and backward LSTMs whose dimensions are set to 100. Tuning this dimension did not significantly impact model performance. We set the dropout rate to 0.5. Using higher rates negatively impacted our results, while smaller rates led to longer training time.

The stack-LSTM model uses two layers each of dimension 100 for each stack. The embeddings of the actions used in the composition functions have 16 dimensions each, and the output embedding is of dimension 20. We experimented with different dropout rates and reported the scores using the best dropout rate for each language.[3] It is a greedy model that apply locally optimal actions until the entire sentence is processed, further improvements might be obtained with beam search (Zhang and Clark, 2011) or training with exploration (Ballesteros et al., 2016).

5.2 Data Sets

We test our model on different datasets for named entity recognition. To demonstrate our model's ability to generalize to different languages, we present results on the CoNLL-2002 and CoNLL-2003 datasets (Tjong Kim Sang, 2002; Tjong Kim Sang and De Meulder, 2003) that contain independent named entity labels for English, Spanish, German and Dutch. All datasets contain four different types of named entities: locations, persons, organizations, and miscellaneous entities that do not belong in any of the three previous categories. Although POS tags were made available for all datasets, we did not include them in our models. We did not perform any dataset preprocessing, apart from replacing every digit with a zero in the English NER dataset.

[3]English (D=0.2), German, Spanish and Dutch (D=0.3)

5.3 Results

Table 1 presents our comparisons with other models for named entity recognition in English. To make the comparison between our model and others fair, we report the scores of other models with and without the use of external labeled data such as gazetteers and knowledge bases. Our models do not use gazetteers or any external labeled resources. The best score reported on this task is by Luo et al. (2015). They obtained a F_1 of 91.2 by jointly modeling the NER and entity linking tasks (Hoffart et al., 2011). Their model uses a lot of hand-engineered features including spelling features, WordNet clusters, Brown clusters, POS tags, chunks tags, as well as stemming and external knowledge bases like Freebase and Wikipedia. Our LSTM-CRF model outperforms all other systems, including the ones using external labeled data like gazetteers. Our Stack-LSTM model also outperforms all previous models that do not incorporate external features, apart from the one presented by Chiu and Nichols (2015).

Tables 2, 3 and 4 present our results on NER for German, Dutch and Spanish respectively in comparison to other models. On these three languages, the LSTM-CRF model significantly outperforms all previous methods, including the ones using external labeled data. The only exception is Dutch, where the model of Gillick et al. (2015) can perform better by leveraging the information from other NER datasets. The Stack-LSTM also consistently presents state-the-art (or close to) results compared to systems that do not use external data.

As we can see in the tables, the Stack-LSTM model is more dependent on character-based representations to achieve competitive performance; we hypothesize that the LSTM-CRF model requires less orthographic information since it gets more contextual information out of the bidirectional LSTMs; however, the Stack-LSTM model consumes the words one by one and it just relies on the word representations when it chunks words.

5.4 Network architectures

Our models had several components that we could tweak to understand their impact on the overall performance. We explored the impact that the CRF, the character-level representations, pretraining of our

Model	F_1
Collobert et al. (2011)*	89.59
Lin and Wu (2009)	83.78
Lin and Wu (2009)*	90.90
Huang et al. (2015)*	90.10
Passos et al. (2014)	90.05
Passos et al. (2014)*	90.90
Luo et al. (2015)* + gaz	89.9
Luo et al. (2015)* + gaz + linking	**91.2**
Chiu and Nichols (2015)	90.69
Chiu and Nichols (2015)*	90.77
LSTM-CRF (no char)	90.20
LSTM-CRF	**90.94**
S-LSTM (no char)	87.96
S-LSTM	90.33

Table 1: English NER results (CoNLL-2003 test set). * indicates models trained with the use of external labeled data

Model	F_1
Florian et al. (2003)*	72.41
Ando and Zhang (2005a)	75.27
Qi et al. (2009)	75.72
Gillick et al. (2015)	72.08
Gillick et al. (2015)*	76.22
LSTM-CRF – no char	75.06
LSTM-CRF	**78.76**
S-LSTM – no char	65.87
S-LSTM	75.66

Table 2: German NER results (CoNLL-2003 test set). * indicates models trained with the use of external labeled data

Model	F_1
Carreras et al. (2002)	77.05
Nothman et al. (2013)	78.6
Gillick et al. (2015)	78.08
Gillick et al. (2015)*	**82.84**
LSTM-CRF – no char	73.14
LSTM-CRF	**81.74**
S-LSTM – no char	69.90
S-LSTM	79.88

Table 3: Dutch NER (CoNLL-2002 test set). * indicates models trained with the use of external labeled data

Model	F_1
Carreras et al. (2002)*	81.39
Santos and Guimarães (2015)	82.21
Gillick et al. (2015)	81.83
Gillick et al. (2015)*	82.95
LSTM-CRF – no char	83.44
LSTM-CRF	**85.75**
S-LSTM – no char	79.46
S-LSTM	83.93

Table 4: Spanish NER (CoNLL-2002 test set). * indicates models trained with the use of external labeled data

word embeddings and dropout had on our LSTM-CRF model. We observed that pretraining our word embeddings gave us the biggest improvement in overall performance of $+7.31$ in F_1. The CRF layer gave us an increase of $+1.79$, while using dropout resulted in a difference of $+1.17$ and finally learning character-level word embeddings resulted in an increase of about $+0.74$. For the Stack-LSTM we performed a similar set of experiments. Results with different architectures are given in table 5.

Model	Variant	F_1
LSTM	char + dropout + pretrain	89.15
LSTM-CRF	char + dropout	83.63
LSTM-CRF	pretrain	88.39
LSTM-CRF	pretrain + char	89.77
LSTM-CRF	pretrain + dropout	90.20
LSTM-CRF	pretrain + dropout + char	**90.94**
S-LSTM	char + dropout	80.88
S-LSTM	pretrain	86.67
S-LSTM	pretrain + char	89.32
S-LSTM	pretrain + dropout	87.96
S-LSTM	pretrain + dropout + char	90.33

Table 5: English NER results with our models, using different configurations. “pretrain” refers to models that include pretrained word embeddings, “char” refers to models that include character-based modeling of words, “dropout” refers to models that include dropout rate.

6 Related Work

In the CoNLL-2002 shared task, Carreras et al. (2002) obtained among the best results on both Dutch and Spanish by combining several small fixed-depth decision trees. Next year, in the CoNLL-2003 Shared Task, Florian et al. (2003) obtained the best score on German by combining the output of four diverse classifiers. Qi et al. (2009) later improved on this with a neural network by doing unsupervised learning on a massive unlabeled corpus.

Several other neural architectures have previously been proposed for NER. For instance, Collobert et al. (2011) uses a CNN over a sequence of word embeddings with a CRF layer on top. This can be thought of as our first model without character-level embeddings and with the bidirectional LSTM being replaced by a CNN. More recently, Huang et al. (2015) presented a model similar to our LSTM-CRF, but using hand-crafted spelling features. Zhou and Xu (2015) also used a similar model and adapted it to the semantic role labeling task. Lin and Wu (2009) used a linear chain CRF with L_2 regularization, they added phrase cluster features extracted from the web data and spelling features. Passos et al. (2014) also used a linear chain CRF with spelling features and gazetteers.

Language independent NER models like ours have also been proposed in the past. Cucerzan and Yarowsky (1999; 2002) present semi-supervised bootstrapping algorithms for named entity recognition by co-training character-level (word-internal) and token-level (context) features. Eisenstein et al. (2011) use Bayesian nonparametrics to construct a database of named entities in an almost unsupervised setting. Ratinov and Roth (2009) quantitatively compare several approaches for NER and build their own supervised model using a regularized average perceptron and aggregating context information.

Finally, there is currently a lot of interest in models for NER that use letter-based representations. Gillick et al. (2015) model the task of sequence-labeling as a sequence to sequence learning problem and incorporate character-based representations into their encoder model. Chiu and Nichols (2015) employ an architecture similar to ours, but instead use CNNs to learn character-level features, in a way similar to the work by Santos and Guimarães (2015).

7 Conclusion

This paper presents two neural architectures for sequence labeling that provide the best NER results ever reported in standard evaluation settings, even compared with models that use external resources, such as gazetteers.

A key aspect of our models are that they model output label dependencies, either via a simple CRF architecture, or using a transition-based algorithm to explicitly construct and label chunks of the input. Word representations are also crucially important for success: we use both pre-trained word representations and "character-based" representations that capture morphological and orthographic information. To prevent the learner from depending too heavily on one representation class, dropout is used.

Acknowledgments

This work was sponsored in part by the Defense Advanced Research Projects Agency (DARPA) Information Innovation Office (I2O) under the Low Resource Languages for Emergent Incidents (LORELEI) program issued by DARPA/I2O under Contract No. HR0011-15-C-0114. Miguel Ballesteros is supported by the European Commission under the contract numbers FP7-ICT-610411 (project MULTISENSOR) and H2020-RIA-645012 (project KRISTINA).

References

Rie Kubota Ando and Tong Zhang. 2005a. A framework for learning predictive structures from multiple tasks and unlabeled data. *The Journal of Machine Learning Research*, 6:1817–1853.

Rie Kubota Ando and Tong Zhang. 2005b. Learning predictive structures. *JMLR*, 6:1817–1853.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based dependency parsing by modeling characters instead of words with LSTMs. In *Proceedings of EMNLP*.

Miguel Ballesteros, Yoav Golderg, Chris Dyer, and Noah A. Smith. 2016. Training with Exploration Improves a Greedy Stack-LSTM Parser. In *arXiv:1603.03793*.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Chris Biemann, Gerhard Heyer, Uwe Quasthoff, and Matthias Richter. 2007. The leipzig corpora collection-monolingual corpora of standard size. *Proceedings of Corpus Linguistic*.

Chris Callison-Burch, Philipp Koehn, Christof Monz, Kay Peterson, Mark Przybocki, and Omar F Zaidan. 2010. Findings of the 2010 joint workshop on statistical machine translation and metrics for machine translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 17–53. Association for Computational Linguistics.

Xavier Carreras, Lluís Màrquez, and Lluís Padró. 2002. Named entity extraction using adaboost, proceedings of the 6th conference on natural language learning. *August*, 31:1–4.

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

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Silviu Cucerzan and David Yarowsky. 1999. Language independent named entity recognition combining morphological and contextual evidence. In *Proceedings of the 1999 Joint SIGDAT Conference on EMNLP and VLC*, pages 90–99.

Silviu Cucerzan and David Yarowsky. 2002. Language independent ner using a unified model of internal and contextual evidence. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–4. Association for Computational Linguistics.

Hong-Jie Dai, Po-Ting Lai, Yung-Chun Chang, and Richard Tzong-Han Tsai. 2015. Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization. *Journal of cheminformatics*, 7(Suppl 1):S14.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. ACL*.

Jacob Eisenstein, Tae Yano, William W Cohen, Noah A Smith, and Eric P Xing. 2011. Structured databases of named entities from bayesian nonparametrics. In *Proceedings of the First Workshop on Unsupervised Learning in NLP*, pages 2–12. Association for Computational Linguistics.

Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. 2003. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 168–171. Association for Computational Linguistics.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2015. Multilingual language processing from bytes. *arXiv preprint arXiv:1512.00103*.

David Graff. 2011. Spanish gigaword third edition (ldc2011t12). *Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA*.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM networks. In *Proc. IJCNN*.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

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

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. 2011. Robust disambiguation of named entities in text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 782–792. Association for Computational Linguistics.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2015. Character-aware neural language models. *CoRR*, abs/1508.06615.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*.

Dekang Lin and Xiaoyun Wu. 2009. Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1030–1038. Association for Computational Linguistics.

Wang Ling, Lin Chu-Cheng, Yulia Tsvetkov, Silvio Amir, Rámon Fernandez Astudillo, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015a. Not all contexts are created equal: Better word representations with variable attention. In *Proc. EMNLP*.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015b. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Gang Luo, Xiaojiang Huang, Chin-Yew Lin, and Zaiqing Nie. 2015. Joint named entity recognition and disambiguation. In *Proc. EMNLP*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proc. NIPS*.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*.

Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R Curran. 2013. Learning multilingual named entity recognition from wikipedia. *Artificial Intelligence*, 194:151–175.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2009. English gigaword fourth edition (ldc2009t13). *Linguistic Data Consortium, Univer-sity of Pennsylvania, Philadelphia, PA*.

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

Yanjun Qi, Ronan Collobert, Pavel Kuksa, Koray Kavukcuoglu, and Jason Weston. 2009. Combining labeled and unlabeled data with word-class distribution learning. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1737–1740. ACM.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Cicero Nogueira dos Santos and Victor Guimarães. 2015. Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proc. CoNLL*.

Erik F. Tjong Kim Sang. 2002. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proc. CoNLL*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proc. ACL*.

Matthew D Zeiler. 2012. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1).

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657.

Jie Zhou and Wei Xu. 2015. End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.