Named Entity Recognition with Bidirectional LSTM-CNNs

Jason P.C. Chiu
University of British Columbia
jsonchiu@gmail.com

Eric Nichols
Honda Research Institute Japan Co.,Ltd.
e.nichols@jp.honda-ri.com

Abstract

Named entity recognition is a challenging task that has traditionally required large amounts of knowledge in the form of feature engineering and lexicons to achieve high performance. In this paper, we present a novel neural network architecture that automatically detects word- and character-level features using a hybrid bidirectional LSTM and CNN architecture, eliminating the need for most feature engineering. We also propose a novel method of encoding partial lexicon matches in neural networks and compare it to existing exact match approaches. Extensive evaluation shows that, given only tokenized text, publicly available word vectors, and an automatically constructed lexicon from open sources, our system is able to surpass the reported state-of-the-art on the OntoNotes 5.0 dataset by 2.35 F1 points and achieves competitive results on the CoNLL 2003 dataset, rivaling systems that employ heavy feature engineering, proprietary lexicons, and rich entity linking information.

1 Introduction

Named entity recognition is an important task in NLP. High performance approaches have been dominated by applying statistical models such as CRF, SVM, or perceptron models to hand-crafted features (Ratinov and Roth, 2009; Passos et al., 2014; Luo et al., 2015). However, Collobert et al. (2011b) proposed an effective neural network model that requires little feature engineering and instead learns important features from word embeddings trained on large quantities of unlabelled text – an approach made possible by recent advancements in unsupervised learning of word embeddings on massive amount of data (Collobert and Weston, 2008; Mikolov et al., 2013) and neural network training algorithms permitting deep architectures (Rumelhart et al., 1986).

Unfortunately there are many limitations to the model proposed by Collobert et al. (2011b). First, it uses a simple feed-forward neural network, which restricts the use of context to a fixed sized window around each word – an approach that discards useful long-distance relations between words. Second, by depending solely on word embeddings, it is unable to exploit explicit character level features such as prefix and suffix, which could be useful especially with rare words where word embeddings are poorly trained. We seek to address these issues by proposing a more powerful neural network model.

A well-studied solution for a neural network to process variable length input and have long term memory is the recurrent neural network (RNN) (Goller and Kuchler, 1996). Recently, RNNs have shown great success in diverse NLP tasks such as speech recognition (Graves et al., 2013), machine translation (Cho et al., 2014), and language modeling (Mikolov et al., 2011). The long-short term memory (LSTM) unit with the forget gate allows highly non-trivial long-distance dependencies to be easily learned (Gers et al., 2000). For sequential labelling tasks such as NER and speech recognition, a bi-directional LSTM model can take into account an effectively infinite amount of context on both sides of a word and eliminates the problem of limited context that applies to any feed-forward model (Graves et al., 2013). While LSTMs have been studied in the past for the NER task by Hammerton (2003), the lack of computational power (which led to the use of very small models) and quality word embeddings limited their effectiveness.

Similarly, convolutional neural networks (CNN) have become very popular in image processing for feature extraction. Recently, dos Santos (2015) and Labeau (2015) successfully employed CNNs to extract character-level features for use in NER and POS-tagging respectively. Collobert et al. (2011b) also applied CNNs to semantic role labeling, and variants of the architecture have been applied to parsing and other tasks requiring tree structures (Blunsom et al., 2014). However, the effectiveness of character-level CNNs has not been evaluated for English NER.

Our main contribution lies in combining these neural network models for the NER task. We present a hybrid model of bidirectional LSTMs and CNNs that learns both character- and word-level features, presenting the first evaluation of such an architecture on well-established English language evaluation datasets. Furthermore, as lexicons are crucial to NER performance, we propose a new lexicon encoding scheme and matching algorithm that can make use of partial matches, and we compare it to the exact match only approach of Collobert et al. (2011b). Extensive evaluation shows that our proposed method performs competitively on the CoNLL-2003 NER shared task and outperforms all known state-of the-art systems on the OntoNotes 5.0 dataset where significantly more training data is available, albeit with a more complex tagset.

2 Model

Our neural network is inspired by the work of Collobert et al. (2011b), where feature vectors are computed by lookup tables and concatenated together, and then fed into a multi-layer network. Instead of a feed-forward network, we use the more powerful recurrent neural network; in particular, we choose the bidirectional long-short term memory (LSTM) network, which has been used successfully in other sequence labelling tasks such as speech recognition (Graves et al., 2013). In addition, to induce character-level features, we draw on the idea of using a convolutional neural network, which has been successfully applied to Spanish and Portuguese NER (dos Santos et al., 2015) and German POS-tagging (Labeau et al., 2015).

2.1 Word-level Features

2.1.1 Word Embeddings

Our best model uses the publicly available 50-dimensional word embeddings released by Collobert et al. (2011b)[1], which were trained on Wikipedia and Reuters RCV-1 corpus.

We also experimented with two other sets of published embeddings, namely Stanford's GloVe embeddings[2] trained on 6 billion words from Wikipedia and web text (Pennington et al., 2014) and Google's word2vec embeddings[3] trained on 100 billion words from Google News (Mikolov et al., 2013).

In addition, as we hypothesized that word embeddings trained on in-domain text may perform better, we also used the publicly available GloVe (Pennington et al., 2014) program and an in-house re-implementation[4] of the word2vec (Mikolov et al., 2013) program to train word embeddings on Wikipedia and Reuters RCV1 dataset as well.[5]

Following Collobert et al. (2011b), all words are lower-cased before passing through the lookup table to convert to their corresponding embeddings. The pre-trained embeddings are allowed to be modified during training[6].

2.1.2 Capitalization Feature

As capitalization information is erased during lookup of the word embedding, we evaluate Collobert's method of using a separate lookup table to add a capitalization feature with the following options: `allCaps`, `upperInitial`, `lowercase`, `mixedCaps`, `noinfo` (Collobert et al., 2011b). This method is compared with the character type features we introduce in Section 2.2.2.

[1]Part of SENNA: `http://ml.nec-labs.com/senna/`

[2]Available at `http://nlp.stanford.edu/projects/glove/`

[3]Available at `https://code.google.com/p/word2vec/`

[4]We used our in-house reimplementation to train word vectors because it uses distributed processing to train much quicker than the publicly-released implementation of word2vec and its performance on the word analogy task was higher than reported by Mikolov et al. (2013).

[5]While Collobert et al. (2011b) used Wikipedia text from 2007, we used Wikipedia text from 2011.

[6]Preliminary experiments showed that modifiable vectors performed better than so-called "frozen vectors."

Category	**Count**
Location	709,772
Miscellaneous	328,575
Organization	231,868
Person	1,074,363
Total	2,344,578

Table 1: Number of entries for each category in the lexicon

2.1.3 Lexicons

Almost all state of the art NER systems make use of lexicons as a form of external knowledge (Ratinov and Roth, 2009; Passos et al., 2014; Durrett and Klein, 2014; Luo et al., 2015).

For each of the four categories (`Person`, `Organization`, `Location`, `Miscellaneous`) defined by the CoNLL 2003 NER shared task, we compiled a list of known named entities from DBpedia (Auer et al., 2007), by extracting all descendants of DBpedia types corresponding to the CoNLL categories[7]. We did not construct separate lexicons for the OntoNotes tagset because correspondences between DBpedia categories and its tags could not be found in many instances. In addition, for each entry we first removed parentheses and all text contained within, then stripped trailing punctuation[8], and finally tokenized it with the Penn Treebank tokenization script for the purpose of partial matching. Table 1 shows the size of each category in our lexicon.

Figure 1 shows an example of how the lexicon features are applied[9]. For each lexicon category, we match every n-gram (up to the length of the longest lexicon entry) against entries in the lexicon. A match is successful when the n-gram matches the prefix or suffix of an entry and is at least half the length of the entry. Because of the high potential for spurious matches, for all categories except Person, we discard partial matches less than 2 tokens in length. When there are multiple overlapping matches within the same category, we prefer exact matches over partial matches, and then longer matches over shorter matches, and finally earlier matches over later matches. All matches are case insensitive.

For each token in the match, the feature is encoded in BIOES annotation (`Begin`, `Inside`, `Outside`, `End`, `Single`), indicating the position of the token in the matched entry. In other words, the `B` label will not appear in a suffix-only partial match, and the `E` label will not appear in a prefix-only partial match.

As shown in Table 10, we found that this more sophisticated method outperforms the method presented by Collobert et al. (2011b), which simply uses exact matching and marks tokens with on/off instead of BIOES annotation.

2.2 Character-level Features

Figure 2 shows the CNN that extracts a fixed-length feature vector from the characters of a single word. For each character in a word, features are encoded using lookup tables and concatenated, then passed through convolution and max layers. The resulting vector is concatenated with other word-level feature vectors described above.

2.2.1 Character Embeddings

We randomly initialized a lookup table with values drawn from an uniform distribution with range $[-0.5, 0.5]$ to output a character embedding of 25 dimensions. The character set includes all unique characters in the CoNLL-2003 dataset (which includes upper and lower case letters, numbers, and punctuations) plus the special tokens `PADDING` and `UNKNOWN`. The `PADDING` token is used for the CNN, and the `UNKNOWN` token is used all other characters (which appear in OntoNotes). The same set of random embeddings were used for all experiments. We chose these settings for convenience and did not experiment with other settings.

2.2.2 Character Type Feature

A lookup table was used to output a 4-dimensional vector representing the type of the character (upper case, lower case, punctuation, other).

[7]The `Miscellaneous` category was populated by entities of the DBpedia categories `Artifact` and `Work`.

[8]The punctuation stripped was period, comma, semi-colon, colon, forward slash, backward slash, and question mark.

[9]As can been seen in this example, the lexicons – in particular `Miscellaneous` – still contain a lot of noise. Better lexicon construction remains an area of future work.

Text	Hayao	Tada	,	commander	of	the	Japanese	North	China	Area	Army
LOC	–	–	–	–	–	B	I	–	S	–	–
MISC	–	–	–	S	B	B	I	S	S	S	S
ORG	–	–	–	–	–	B	I	B	I	I	E
PERS	B	E	–	–	–	–	–	–	S	–	–

Figure 1: Example of how lexicon features are applied. The B, I, E, markings indicate that the token matches the Begin, Inside, and End token of an entry in the lexicon. S indicates that the token matches a single-token entry in the lexicon.

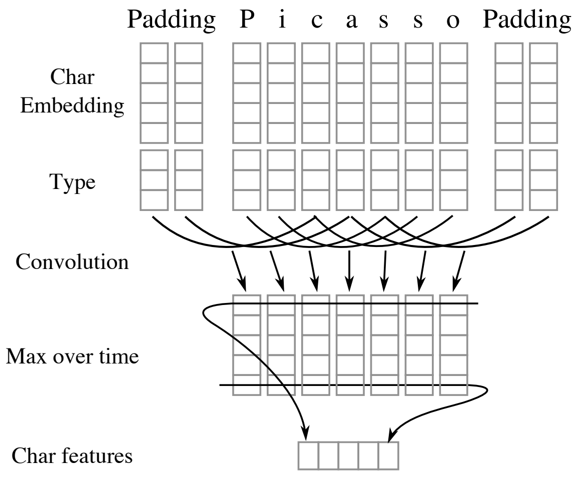

Figure 2: The convolutional neural network extracts character features from each word. The character embedding and the character type feature vector are computed through lookup tables. Then, they are concatenated and passed into the CNN.

2.2.3 Extracting Features Using a Convolutional Neural Network

For each word we employ a convolution and a max layer to extract a new feature vector from the per-character feature vectors described above. Words are padded with a number of special PADDING characters on both sides depending on the window size of the CNN.

Hyper-parameters of the CNN include the window size and the output vector size.

2.3 Sequence-labelling with Bidirectional LSTM

Following the speech-recognition framework outlined by Graves et al. (2013), we employed a stacked[10] bi-directional recurrent neural network with long short-term memory (LSTM) units to transform the extracted word features into a distribution of named entity tag scores. This network configuration will be referred to as BLSTM throughout the remainder of the paper. Figure 3 and Figure 4 illustrates the network in detail.

The extracted features of each word is fed into a forward LSTM network and a backward LSTM network. The output of each recurrent network at each time step is fed through a linear layer and a log-softmax layer to decode into log-probabilities for each tag category. These two vectors are then simply added together to produce the final output.

We tried minor variants of output layer architecture and selected the one that performed the best in preliminary experiments.

2.4 Training and Inference

2.4.1 Implementation

We implement the neural network using the torch7 library (Collobert et al., 2011a). Training and inference are done on a per-sentence level. The initial states of the LSTM are zero vectors. Except for the character and word embeddings whose initialization have been described previously, all lookup tables are randomly initialized with values drawn from the standard normal distribution.

2.4.2 Objective Function and Inference

We train our network to maximize the sentence-level log-likelihood as presented by Collobert et al. (2011b).

First, we define a tag-transition matrix A where $A_{i,j}$ represents the score of jumping from tag i to tag j in successive words, and $A_{0,i}$ as the score for

[10]For each direction (forward and backward), the input is fed into multiple layers of LSTM units connected in sequence (i.e. LSTM units in the second layer takes in the output of the first layer, and so on); the number of layers is a tuned hyper-parameter. Figure 3 shows only one unit for simplicity.

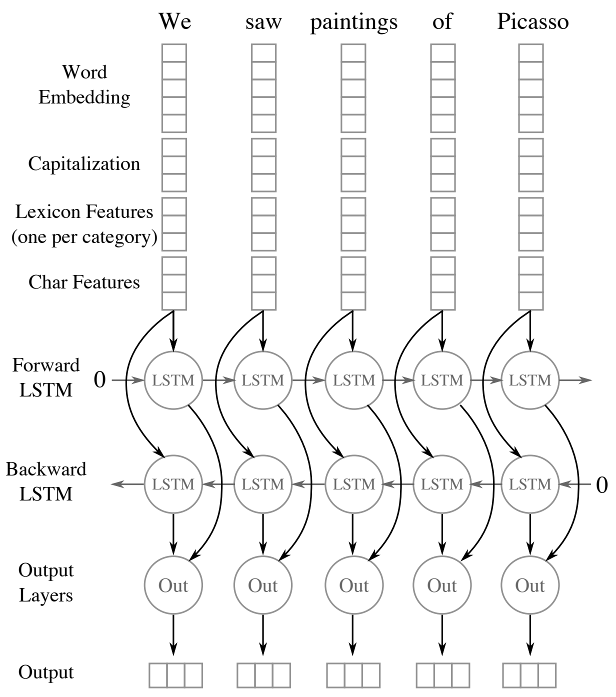

Figure 3: The (unrolled) BLSTM for tagging named entities. Multiple lookup tables compute the word embedding, capitalization, and lexicon feature vectors. The CNN (Figure 2) computes the char feature vector. For each word, these vectors are concatenated and fed to the BLSTM network, and then the output layers (Figure 4).

starting with tag i. This matrix of parameters are also learned. Define θ as the set of parameters for the neural network, and $\theta' = \theta \cup \{A_{i,j}\ \forall i, j\}$ as the set of all parameters to be trained. Given an example sentence, $[x]_1^T$, of length T, and define $[f_\theta]_{i,t}$ as the score outputted by the neural network for the t^{th} word and i^{th} tag, then the score of a *sequence* of tags $[i]_1^T$ is given as the sum of network and transition scores:

$$S([x]_1^T, [i]_1^T, \theta') = \sum_{t=1}^{T} \left(A_{[i]_{t-1},[i]_t} + [f_\theta]_{[i]_t,t} \right)$$

Then, letting $[y]_1^T$ be the true tag sequence, the sentence-level log-likelihood is obtained by normalizing the above score over all possible tag-sequences $[j]_1^T$ using a softmax:

$$\begin{aligned} &\log P([y]_1^T \mid [x]_1^T, \theta') \\ &= S([x]_1^T, [y]_1^T, \theta') - \log \sum_{\forall [j]_1^T} e^{S([x]_1^T, [j]_1^T, \theta')} \end{aligned}$$

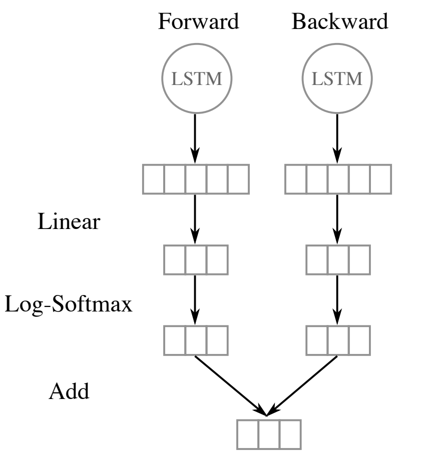

Figure 4: The output layers ("Out" in Figure 3) to decode LSTM output into a score for each tag category, which can be interpreted as log-probabilities of the tag being correct

This objective function and its gradients can be efficiently computed by dynamic programming (Collobert et al., 2011b).

At inference time, given neural network outputs $[f_\theta]_{i,t}$ we use the Viterbi algorithm to find the tag sequence $[i]_1^T$ that maximizes the score $S([x]_1^T, [i]_1^T, \theta')$.

2.4.3 Learning Algorithm

Training is done by mini-batch stochastic gradient descent (SGD) with a fixed learning rate. Each mini-batch consists of multiple sentences with the same number of tokens.

As shown in Table 8 and Table 9, we found that applying dropout to the input and output nodes of each LSTM layer (Pham et al., 2014) was quite effective in reducing overfitting.

We explored other more sophisticated optimization algorithms such as momentum (Nesterov, 1983), AdaDelta (Zeiler, 2012), and RMSProp (Hinton et al., 2012), and in preliminary experiments they did not meaningfully improve upon plain SGD.

2.4.4 Tagging Scheme

The output tags are annotated with BIOES (which stand for `Begin`, `Inside`, `Outside`, `End`, `Single`) as this scheme has been reported to outperform others such as BIO (Ratinov and Roth, 2009).

Dataset	Train	Dev	Test
CoNLL-2003	204,567 (23,505)	51,578 (5,943)	46,666 (5,652)
OntoNotes 5.0 / CoNLL-2012	1,088,503 (81,828)	147,724 (11,066)	152,728 (11,257)

Table 2: Size of each dataset in number of tokens (number of named entities)

3 Evaluation

Evaluation was performed on the well-established CoNLL-2003 NER shared task dataset (Tjong Kim Sang and De Meulder, 2003) and the much larger but less-studied OntoNotes 5.0 dataset (Hovy et al., 2006). Table 2 gives an overview of these two different datasets.

We ran each experiment multiple times and report the average and standard deviation of at least 7 successful trials. Since for unknown reasons, sometimes the model fails to converge, in all experiments we exclude trials where the final F1-score on a subset of training data falls below a certain threshold.

3.1 Dataset Preprocessing

For all datasets, we performed the following preprocessing:

- All sequences of digits 0-9 are replaced by a single "0".
- Before training starts, sentences are grouped by sentence length and separated into mini-batches, which are then shuffled.

In addition, for the OntoNotes dataset, in order to handle the `Date`, `Time`, `Money`, `Percent`, `Quantity`, `Ordinal`, and `Cardinal` named entity tags, we split tokens before and after every digit.

3.2 CoNLL 2003 Dataset

Table 3 shows the chosen hyper-parameters for all experiments involving the CoNLL-2003 dataset. We tuned the hyper-parameters on the development set by random search, then trained the model on both the training and development sets.

We excluded all trials where the final F1 score on the development set was less than 95. There was no ambiguity in selecting the threshold as every trial scored either above 99 or below 90.

Hyper-parameter	CoNLL-2003	OntoNotes
Convolution width	3	3
CNN output size	20	20
LSTM state size	200	200
LSTM layers	2	2
Learning rate	0.0166	0.008
Epochs	100	18
Dropout	0.63	0.63
Mini-batch Size	9	9

Table 3: Final hyper-parameter choice for all experiments

We trained the models for a large number of epochs because while tuning on the development set, we found that for some initializations, the model appears to plateau around 20-30 epochs, and yet slowly improves starting at epoch 50 and beyond.

3.3 OntoNotes 5.0 Dataset

Following Durrett and Klein (2014), we applied our model to the portion of the CoNLL-2012 shared task dataset (Pradhan et al., 2012) with gold-standard named entity annotations; the New Testaments portion was excluded for lacking gold-standard annotations.

For hyper-parameters, due to time constraints it was infeasible to do a fresh random search across the full parameter space. Thus, we simply used the best setting from CoNLL 2003 experiments and tuned the number of epochs and learning rate based on development set performance. Table 3 shows the final hyper-parameters for all experiments involving this dataset.

We excluded all trials where the final F1 score on the last 5,000 sentences of the training set was less than 80; every trial scored either above 80 or below 50.

4 Results and Discussion

Table 4 shows the results for all datasets. Our best model is competitive with other state of the art systems on the CoNLL-2003 dataset, and for the OntoNotes dataset, to the best of our knowledge we

[11]OntoNotes results taken from (Durrett and Klein, 2014)

[12]It was unclear whether or not they evaluated their system on the CoNLL-2012 split of the OntoNotes dataset.

[13]Numbers taken from the original paper (Luo et al., 2015). While the precision, recall, and F1 scores are clearly inconsistent, it is unclear in which way they are incorrect.

Model	CoNLL-2003			OntoNotes 5.0		
	Prec.	**Recall**	**F1**	**Prec.**	**Recall**	**F1**
BLSTM	80.38	73.90	77.00 (± 0.54)	79.68	75.97	77.77 (± 0.37)
BLSTM-CNN	82.97	83.71	83.34 (± 0.36)	82.58	82.49	82.53 (± 0.40)
BLSTM-CNN + emb	90.25	90.88	90.56 (± 0.44)	**86.05**	86.37	86.21 (± 0.20)
BLSTM-CNN + emb + lex	90.41	91.15	90.77 (± 0.35)	85.97	**86.83**	**86.39** (± 0.19)
Collobert et al. (2011b)	-	-	88.67	-	-	-
Collobert et al. (2011b) + lexicon	-	-	89.59	-	-	-
Huang et al. (2015)	-	-	90.10	-	-	-
Ratinov and Roth (2009)[11]	91.20	90.50	90.80	82.00	84.95	83.45
Lin and Wu (2009)	-	-	90.90	-	-	-
Passos et al. (2014)[12]	-	-	90.90	-	-	82.24
Durrett and Klein (2014)	-	-	-	85.22	82.89	84.04
Luo et al. (2015)[13]	**91.50**	**91.40**	**91.20**	-	-	-

Table 4: Results of our models, with various feature sets, compared to other published results. The three sections are, in order, our models, published neural network models, and published non-neural network models. For the features, emb = Collobert word embeddings, lex = lexicon features. For F1-scores, standard deviations are in parentheses.

BLSTM Models		
Features	**CoNLL**	**OntoNotes**
none	77.00 (± 0.54)	77.77 (± 0.37)
emb	87.67 (± 0.28)	82.72 (± 0.23)
emb + caps	90.27 (± 0.22)	86.19 (± 0.25)
emb + caps + lex	**90.48** (± 0.32)	**86.13** (± 0.24)

Table 5: F1-score results of BLSTM models with various additional features; emb = Collobert word embeddings, caps = capitalization feature, lex = lexicon features.

BLSTM-CNN Models		
Features	**CoNLL**	**OntoNotes**
none	83.34 (± 0.36)	82.53 (± 0.40)
emb	90.56 (± 0.44)	86.21 (± 0.20)
emb + char	90.48 (± 0.38)	86.08 (± 0.40)
emb + caps	90.69 (± 0.28)	86.35 (± 0.28)
emb + char + caps	90.50 (± 0.39)	**86.41** (± 0.22)
emb + lex	**90.77** (± 0.35)	86.39 (± 0.19)
emb + char + lex	90.59 (± 0.38)	86.32 (± 0.25)
emb + caps + lex	90.76 (± 0.27)	86.16 (± 0.16)
emb+char+caps+lex	90.66 (± 0.49)	86.30 (± 0.32)

Table 6: F1-score results of BLSTM-CNN models with various additional features; emb = Collobert word embeddings, char = character type feature, caps = capitalization feature, lex = lexicon features.

have surpassed the previous highest reported results on all of precision, recall, and F1-score. In particular, word embeddings and character-level features (produced by the CNN) contributed most of the performance, suggesting that when given enough data, the neural network is able to automatically learn the relevant features for NER without feature engineering.

4.1 Character-level CNNs vs. Character Type and Capitalization Features

Comparison of Table 5 and Table 6 shows that on CoNLL-2003, BLSTM-CNN models significantly[14] outperform the BLSTM models when given the same feature set. This effect is much smaller and not statistically significant on OntoNotes when capitalization features are added. Notably, with the presence of the character-level CNN, neither the character type nor the capitalization feature provide any improvements at all. These results suggest that the character-level CNN can replace hand-crafted character class features.

4.2 Word Embeddings

Table 4 and Table 7 show that we obtain a large, significant[15] improvement when trained word embeddings are used, as opposed to random embeddings, regardless of the additional features used. This is

[14]Wilcoxon rank sum test, $p < 0.05$ when comparing the four BLSTM models with the corresponding BLSTM-CNN models utilizing the same feature set. The Wilcoxon rank sum test was selected for its robustness against small sample sizes when the distribution is unknown.

[15]Wilcoxon rank sum test, $p < 0.001$

Word Embeddings	**CoNLL-2003**	**OntoNotes**
Random 50d	85.69 (± 0.55)	83.79 (± 0.28)
Random 300d	85.87 (± 0.50)	83.37 (± 0.37)
GloVe 6B 50d	89.60 (± 0.49)	86.18 (± 0.22)
GloVe 6B 300d	89.66 (± 0.45)	86.09 (± 0.27)
Google 100B 300d	89.53 (± 0.40)	85.01 (± 0.39)
Collobert 50d	**90.77** (± 0.35)	**86.39** (± 0.19)
Our GloVe 50d	90.34 (± 0.45)	86.10 (± 0.23)
Our Word2Vec 50d	89.84 (± 0.32)	85.79 (± 0.17)

Table 7: F1 score on the datasets when the Collobert vectors are replaced by different choices of word vectors. We tried 50-dimensional and 300-dimensional random vectors (Random 50d, Random 300d), as well as the corresponding GloVe vectors trained on 6 billion words (GloVe 6B 50d, GloVe 6B 300d), and Google's released 300-dimensional vectors trained on 100 billion words from Google News (Google 100B 300d). We also compare them to the 50-dimensional GloVe and word2vec vectors that we trained on Wikipedia and Reuters RCV-1 (Our GloVe 50d, Our Word2Vec 50d).

consistent with results reported previously by Collobert et. al. (2011b).

Table 7 compares the performance of different word embeddings in our best model in Table 4 (BLSTM-CNN + emb + lex). For CoNLL-2003, publicly available GloVe and Google embeddings are about one point behind Collobert's embeddings. For OntoNotes, GloVe embeddings perform close to Collobert embeddings while Google embeddings are again one point behind. In addition, 300 dimensional embeddings present no significant improvement over 50 dimensional embeddings – a result previously reported by Turian et al. (2010).

One possible reason that Collobert embeddings perform better than other publicly available embeddings on CoNLL-2003 is that they are trained on the Reuters RCV-1 corpus, the source of CoNLL-2003 dataset, whereas the other embeddings are not. On the other hand, we suspect that Google's embeddings perform poorly because of vocabulary mismatch – in particular, Google's embeddings were trained in a case-sensitive manner, and embeddings for many common punctuations and symbols were not provided. To test these hypotheses, we performed experiments with new word embeddings trained using GloVe and word2vec, with vocabulary list and corpus similar to Collobert et. al. (2011b).

Dropout	**CoNLL-2003**	
	Train	**Test**
No dropout	**99.97** (± 0.03)	89.48 (± 0.99)
Dropout = 0.63	**99.97** (± 0.01)	**90.77** (± 0.35)

Table 8: F1-score results with and without dropout on CoNLL-2003.

Dropout	**OntoNotes 5.0**	
	Train	**Test**
No dropout	**96.78** (± 4.07)	83.95 (± 0.61)
Dropout = 0.63	90.66 (± 0.49)	**86.39** (± 0.19)

Table 9: F1-score results with and without dropout on OntoNotes 5.0.

As shown in Table 7, our GloVe embeddings improved significantly[16] over publicly available embeddings on CoNLL-2003, and our word2vec embeddings improved significantly[17] over Google's embeddings on OntoNotes.

Due to time constraints we did not perform new hyper-parameter searches with any of the word embeddings. As word embedding quality depends on hyper-parameter choice during their training (Pennington et al., 2014), and also, in our NER neural network, hyper-parameter choice is likely sensitive to the type of word embeddings used, optimizing them all will likely produce better results and provide a fairer comparison of word embedding quality.

4.3 Effect of Dropout

Table 8 and Table 9 compares the result with and without dropout for each dataset. All other hyper-parameters and features remain the same as our best model Table 4. In both datasets, dropout is essential for state of the art performance, and the improvement is statistically significant[18]; without dropout, performance on training data improves while test data performance suffers.

4.4 Lexicon Features

Table 5 and Table 6 show that on the CoNLL-2003 dataset, addition of lexicon features appears to pro-

[16]Wilcoxon rank sum test, $p < 0.01$

[17]Wilcoxon rank sum test, $p < 0.001$

[18]Wilcoxon rank sum test, $p < 0.01$ for both CoNLL-2003 and OntoNotes.

Lexicon Method	CoNLL-2003	OntoNotes
No lexicon	90.56 ($\pm$ 0.44)	86.21 ($\pm$ 0.20)
Collobert	90.37 ($\pm$ 0.33)	86.15 ($\pm$ 0.24)
Our method	**90.77** ($\pm$ 0.35)	**86.39** ($\pm$ 0.19)

Table 10: F1-score when different lexicon matching methods are applied.

vide a small improvement[19] in all models. Unfortunately such an improvement does not occur for the OntoNotes, most likely because our lexicon is not populated for many of the tags in the OntoNotes tagset due to the lack of corresponding DBpedia categories. Another interpretation of these results is that our character-level CNN model is learning much of the same knowledge that is encoded in the NE lexicons[20].

For lexicon matching, we also tried the method outlined by Collobert et al. (2011b), which involves performing exact matching for each category and turning "on" the corresponding lexicon feature when a match is found. Table 10 shows that their more simplistic method provides no improvement at all over not using a lexicon. We hypothesize that our model is able to learn character-level features that perform better than Collobert et al.'s inflexible lexical matching.

5 Related Research

Named entity recognition is a task with a long history in NLP. In this section, we summarize the works that are most relevant to our research, namely those we make direct comparisons with and those that our approach drew inspiration from.

5.1 Named Entity Recognition

Most recent approaches to NER has been characterized by the use of CRF models or other models like SVMs and averaged perceptron models, where performance is heavily dependent on feature engineering. Ratinov and Roth (2009) used non-local features, gazetteer extracted from Wikipedia, and Brown-cluster-like word representations, and achieved an F1 score of 90.80 on CoNLL-2003. Lin and Wu (2009) surpassed them without using a gazetteer, by instead using phrase features obtained by performing k-means clustering over a private database of search engine query logs. Passos et al. (2014) obtained nearly the same performance using only public data by training phrase vectors in their lexicon-infused SkipGram model.

Training an NER system together with related tasks such as entity linking has recently been shown to improve the state of the art. Durrett and Klein (2014) combined coreference resolution, entity linking, and NER into a single CRF model and added cross-task interaction factors. Their system achieved state of the art results on the OntoNotes dataset, but they did not evaluate on the CoNLL-2003 dataset due to lack of coreference annotations. Luo et al. (2015) achieved the current state of the art results on CoNLL-2003 by training a joint model over the NER and entity linking tasks, the pair of tasks whose inter-dependencies contributed the most to the work of Durrett and Klein (2014).

5.2 NER with Neural Networks

While many approaches involve CRF models, there has also been a long history of research applying neural networks to NER. Early attempts were hindered by lack of computational power, scalable learning algorithms, and high quality word embeddings.

Petasis et al. (2000) used a feed-forward neural network with one hidden layer on NER and achieved state-of-the-art results on the MUC6 dataset. Their approach used one-hot encodings for POS tag and gazetteer tags for each word, with no word embeddings.

Hammerton (2003) attempted NER with a single-direction LSTM network and a combination of 64-dimensional word vectors trained using self-organizing maps, as well as context vectors obtained using principle component analysis. However, while our method optimizes log-likelihood and uses softmax, they used a different output encoding and optimized an unspecified objective function. Hammerton's (2003) reported results were only slightly above baseline models.

Much later, with the advent of neural word embeddings, Collobert et al. (2011b) presented SENNA, which employs a feed-forward deep neu-

[19] Not statistically significant

[20] This should not come as a great surprise given the amount of information about proper noun status that is conveyed by capitalization in the English language.

ral network and word embeddings to achieve near state of the art results on POS tagging, chunking, NER, and SRL. We build on their approach, sharing the word embeddings, feature encoding method, and objective functions.

Recently, dos Santos et al. (2015) presented their CharWNN network, which augments the neural network of Collobert et al. (2011b) with character level CNNs, and they reported improved performance on Spanish and Portuguese NER. We have successfully incorporated character-level CNNs into our RNN model[21].

Our method is most similar to the recent Bi-LSTM-CRF model for POS-tagging, chunking, and NER presented by Huang et al. (2015) and the Bi-RNN model for POS-tagging presented by Labeau et al. (2015). Both of them employed the same objective function and similar bidirectional networks, with the former using a Bi-directional LSTM without CNNs and the latter using character-level CNNs with standard RNNs. Our method differs from Huang et al. (2015) in that we employ very little feature engineering and instead take advantage of character level CNNs to automatically extract features from a word. Unlike Labeau et al. (2015), we employed the more powerful LSTM, which we found to outperform standard RNNs in preliminary experimentation as well as being easier to train. In addition, while Labeau et al. (2015) reported near state-of-the-art performance in German POS tagging with no word embeddings at all, we found that even with character-level CNNs, word embeddings are crucial to NER performance.

6 Conclusion

We have shown that our neural network model, which incorporates a bidirectional LSTM and a character-level CNN and which benefits from robust training through dropout, achieves state-of-the-art results in named entity recognition with little feature engineering. Our model improves over previous best reported results on the large OntoNotes dataset, indicating that the model is capable of learning complex relationships from large amounts of data, and achieves competitive performance on the smaller CoNLL 2003 dataset.

Preliminary evaluation of our partial matching lexicon algorithm suggests that performance could be further improved through more flexible application of existing lexicons, but that our character-level CNN models learn much of the information that is encoded in lexicons. Evaluation of existing word vectors suggests that the domain of training data text is as important as the word vector training algorithm.

More effective construction and application of lexicons and word vectors are areas that require more research. In the future, we would also like to extend our model to perform similar tasks such as extended tagset NER and entity linking.

[21]Attempts at character-level RNNs did not perform as well as CNNs despite promising results in language model applications (Karpathy et al., 2015).

Acknowledgments

This research was supported by Honda Research Institute Japan, Co.Ltd. The authors would like to thank Collobert et al. (2011b) for releasing the SENNA source code and word vectors, the torch7 framework contributors, and Andrey Karpathy for the reference LSTM implementation.

References

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. *DBpedia: A nucleus for a web of open data*. Springer.

Phil Blunsom, Edward Grefenstette, Nal Kalchbrenner, et al. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. ACL.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. 2011a. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376.

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

Cıcero dos Santos, Victor Guimaraes, RJ Niterói, and Rio de Janeiro. 2015. Boosting named entity recognition with neural character embeddings. In *Proceedings of NEWS 2015 The Fifth Named Entities Workshop*, page 25.

Greg Durrett and Dan Klein. 2014. A joint model for entity analysis: Coreference, typing, and linking. *Transactions of the Association for Computational Linguistics*, 2:477–490.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with LSTM. *Neural computation*, 12(10):2451–2471.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE.

Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE.

James Hammerton. 2003. Named entity recognition with long short-term memory. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 172–175. Association for Computational Linguistics.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. 2012. Lecture 6e: rmsprop: divide the gradient by a running average of its recent magnitude. In *Neural Networks for Machine Learning*. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. OntoNotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics.

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

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2015. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

Matthieu Labeau, Kevin Löser, and Alexandre Allauzen. 2015. Non-lexical neural architecture for fine-grained pos tagging. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 232–237, Lisbon, Portugal, September. Association for Computational Linguistics.

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.

Gang Luo, Xiaojiang Huang, Chin-Yew Lin, and Zaiqing Nie. 2015. Joint entity recognition and disambiguation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 879–888, Lisbon, Portugal, September. Association for Computational Linguistics.

Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukar Burget, and Jan Cernocky. 2011. Rnnlm-recurrent neural network language modeling toolkit. In *Proc. of the 2011 ASRU Workshop*, pages 196–201.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Yurii Nesterov. 1983. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376.

Alexandre Passos, Vineet Kumar, and Andrew McCallum. 2014. Lexicon infused phrase embeddings for named entity resolution. *Proceedings of CoNLL-2014*, page 78.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543.

G Petasis, S Petridis, G Paliouras, V Karkaletsis, SJ Perantonis, and CD Spyropoulos. 2000. Symbolic and neural learning for named-entity recognition. In *Symposium on Computational Intelligence and Learning, Chios, Greece*, pages 58–66. Citeseer.

Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. 2014. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 285–290. IEEE.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint Conference on EMNLP and CoNLL-Shared Task*, pages 1–40. Association for Computational Linguistics.

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.

David Rumelhart, Geoffrey Hinton, and Ronald Williams. 1986. Learning representations by back-propagating errors. *Nature*, pages 323–533.

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*, pages 142–147. Association for Computational Linguistics.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.