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 approaches. Extensive evaluation shows that, given only tokenized text and publicly available word embeddings, our system is competitive on the CoNLL-2003 dataset and surpasses the previously reported state of the art performance on the OntoNotes 5.0 dataset by 2.13 F1 points. By using two lexicons constructed from publicly-available sources, we establish new state of the art performance with an F1 score of 91.62 on CoNLL-2003 and 86.28 on OntoNotes, surpassing 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 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 amounts 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

Transactions of the Association for Computational Linguistics, vol. 4, pp. 357–370, 2016. Action Editor: Masaaki Nagata.
Submission batch: 11/2015; Revision batch: 3/2016; Published 7/2016.

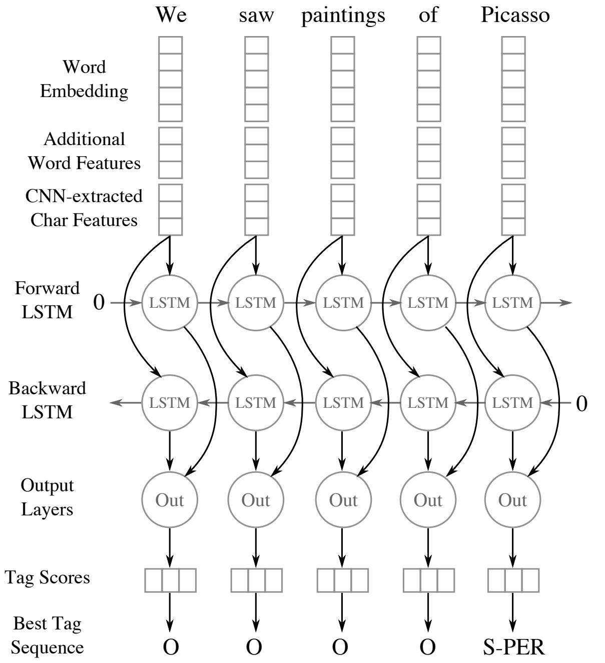

Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

of very small models) and quality word embeddings limited their effectiveness.

Convolutional neural networks (CNN) have also been investigated for modeling character-level information, among other NLP tasks. Santos et al. (2015) and Labeau et al. (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. While we considered using character-level bi-directional LSTMs, which was recently proposed by Ling et al. (2015) for POS-tagging, preliminary evaluation shows that it does not perform significantly better than CNNs while being more computationally expensive to train.

Our main contribution lies in combining these neural network models for the NER task. We present a hybrid model of bi-directional 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 simpler approach of Collobert et al. (2011b). Extensive evaluation shows that our proposed method establishes a new state of the art on both the CoNLL-2003 NER shared task and the OntoNotes 5.0 datasets.

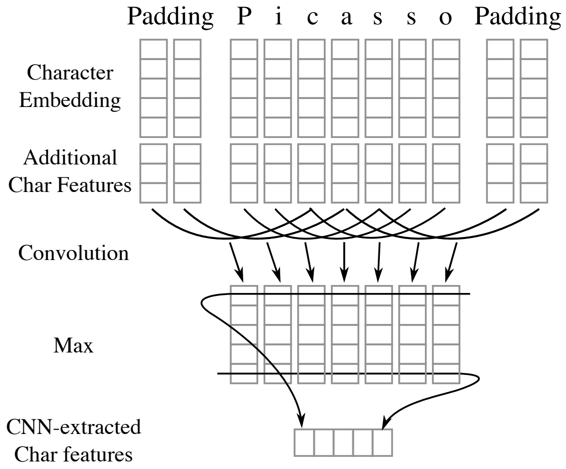

Figure 2: The convolutional neural network extracts character features from each word. The character embedding and (optionally) the character type feature vector are computed through lookup tables. Then, they are concatenated and passed into the CNN.

2 Model

Our neural network is inspired by the work of Collobert et al. (2011b), where lookup tables transform discrete features such as words and characters into continuous vector representations, which are then concatenated and fed into a neural network. Instead of a feed-forward network, we use the bi-directional long-short term memory (BLSTM) network. To induce character-level features, we use a convolutional neural network, which has been successfully applied to Spanish and Portuguese NER (Santos et al., 2015) and German POS-tagging (Labeau et al., 2015).

2.1 Sequence-labelling with BLSTM

Following the speech-recognition framework outlined by Graves et al. (2013), we employed

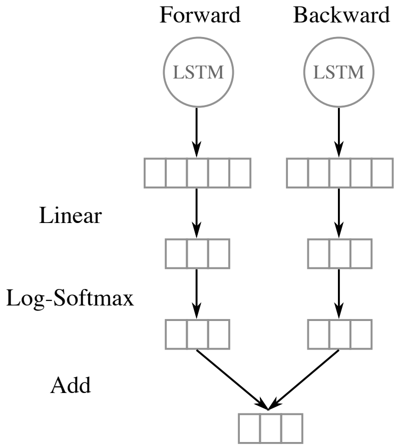

Figure 3: The output layers ("Out" in Figure 1) decode output into a score for each tag category.

a stacked[1] bi-directional recurrent neural network with long short-term memory units to transform word features into named entity tag scores. Figures 1, 2, and 3 illustrate the network in detail.

The extracted features of each word are fed into a forward LSTM network and a backward LSTM network. The output of each network at each time step is decoded by a linear layer and a log-softmax layer 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.

2.2 Extracting Character 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 such as character embeddings (Section 2.3.2) and (optionally) character type (Section 2.5). Words are padded with a number of special `PADDING` characters on both sides depending on the window size of the CNN.

The hyper-parameters of the CNN are the window size and the output vector size.

Category	SENNA	DBpedia
`Location`	36,697	709,772
`Miscellaneous`	4,722	328,575
`Organization`	6,440	231,868
`Person`	123,283	1,074,363
Total	171,142	2,344,578

Table 1: Number of entries for each category in the SENNA lexicon and our DBpedia lexicon.

Dataset	Train	Dev	Test
CoNLL-2003	204,567 (23,499)	51,578 (5,942)	46,666 (5,648)
OntoNotes 5.0 / CoNLL-2012	1,088,503 (81,828)	147,724 (11,066)	152,728 (11,257)

Table 2: Dataset sizes in number of tokens (entities)

2.3 Core Features

2.3.1 Word Embeddings

Our best model uses the publicly available 50-dimensional word embeddings released by Collobert et al. (2011b)[2], which were trained on Wikipedia and the Reuters RCV-1 corpus.

We also experimented with two other sets of published embeddings, namely Stanford's GloVe embeddings[3] trained on 6 billion words from Wikipedia and Web text (Pennington et al., 2014) and Google's word2vec embeddings[4] 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[5] of the word2vec (Mikolov et al., 2013) program to train word embeddings on Wikipedia and Reuters RCV1 datasets as well.[6]

Following Collobert et al. (2011b), all words are lower-cased before passing through the lookup table

[1] 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 take in the output of the first layer, and so on); the number of layers is a tuned hyper-parameter. Figure 1 shows only one unit for simplicity.

[2] `http://ml.nec-labs.com/senna/`

[3] `http://nlp.stanford.edu/projects/glove/`

[4] `https://code.google.com/p/word2vec/`

[5] 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).

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

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 4: 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.

to convert to their corresponding embeddings. The pre-trained embeddings are allowed to be modified during training.[7]

2.3.2 Character Embeddings

We randomly initialized a lookup table with values drawn from a 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[8] plus the special tokens PADDING and UNKNOWN. The PADDING token is used for the CNN, and the UNKNOWN token is used for all other characters (which appear in OntoNotes). The same set of random embeddings was used for all experiments.[9]

2.4 Additional Word-level Features

2.4.1 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 feature (Section 2.5) and character-level CNNs.

2.4.2 Lexicons

Most state of the art NER systems make use of lexicons as a form of external knowledge (Ratinov and Roth, 2009; Passos et al., 2014).

For each of the four categories (Person, Organization, Location, Misc) 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.[14] 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,[15] 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 compared to Collobert's lexicon, which we extracted from their SENNA system.

Figure 4 shows an example of how the lexicon features are applied.[16] 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 in the sentence over later matches. All matches are case insensitive.

For each token in the match, the feature is en-

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

[8] Upper and lower case letters, numbers, and punctuations

[9] We did not experiment with other settings because the English character set is small enough that effective embeddings could be learned directly from the task data.

[10] By increments of 50.

[11] Determined by evaluating dev set performance.

[12] Probability of *discarding* any LSTM output node.

[13] Mini-batch size was excluded from the round 2 particle swarm hyper-parameter search space due to time constraints.

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

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

[16] As can been seen in this example, the lexicons – in particular Miscellaneous – still contain a lot of noise.

Hyper-parameter	CoNLL-2003 (Round 2)		OntoNotes 5.0 (Round 1)	
	Final	Range	Final	Range
Convolution width	**3**	$[3, 7]$	**3**	$[3, 9]$
CNN output size	**53**	$[15, 84]$	**20**	$[15, 100]$
LSTM state size	**275**	$[100, 500]$	**200**	$[100, 400]$[10]
LSTM layers	**1**	$[1, 4]$	**2**	$[2, 4]$
Learning rate	**0.0105**	$[10^{-3}, 10^{-1.8}]$	**0.008**	$[10^{-3.5}, 10^{-1.5}]$
Epochs[11]	**80**	-	**18**	-
Dropout[12]	**0.68**	$[0.25, 0.75]$	**0.63**	$[0, 1]$
Mini-batch size	**9**	-[13]	**9**	$[5, 14]$

Table 3: Hyper-parameter search space and final values used for all experiments

Round	CoNLL-2003	OntoNotes 5.0
1	93.82 (± 0.15)	**84.57** (± 0.27)
2	**94.03** (± 0.23)	84.47 (± 0.29)

Table 4: Development set F1 score performance of the best hyper-parameter settings in each optimization round.

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

As we will see in Section 4.5, we found that this more sophisticated method outperforms the method presented by Collobert et al. (2011b), which treats partial and exact matches equally, allows prefix but not suffix matches, allows very short partial matches, and marks tokens with `YES`/ `NO`.

In addition, since Collobert et al. (2011b) released their lexicon with their SENNA system, we also applied their lexicon to our model for comparison and investigated using both lexicons simultaneously as distinct features. We found that the two lexicons complement each other and improve performance on the CoNLL-2003 dataset.

Our best model uses the SENNA lexicon with exact matching and our DBpedia lexicon with partial matching, with BIOES annotation in both cases.

2.5 Additional Character-level Features

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

2.6 Training and Inference

2.6.1 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 has been described previously, all lookup tables are randomly initialized with values drawn from the standard normal distribution.

2.6.2 Objective Function and Inference

We train our network to maximize the sentence-level log-likelihood from Collobert et al. (2011b).[17]

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 tokens, and $A_{0,i}$ as the score for 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 given parameters θ, then the score of a *sequence* of tags $[i]_1^T$ is given as the sum of network and transition scores:

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

[17] Much later, we discovered that training with cross entropy objective while performing Viterbi decoding to restrict output to valid tag sequences also appears to work just as well.

Model	CoNLL-2003			OntoNotes 5.0		
	Prec.	Recall	F1	Prec.	Recall	F1
FFNN + emb + caps + lex	89.54	89.80	89.67 (± 0.24)	74.28	73.61	73.94 (± 0.43)
BLSTM	80.14	72.81	76.29 (± 0.29)	79.68	75.97	77.77 (± 0.37)
BLSTM-CNN	83.48	83.28	83.38 (± 0.20)	82.58	82.49	82.53 (± 0.40)
BLSTM-CNN + emb	90.75	91.08	90.91 (± 0.20)	85.99	86.36	86.17 (± 0.22)
BLSTM-CNN + emb + lex	91.39	**91.85**	**91.62** (± 0.33)	**86.04**	**86.53**	**86.28** (± 0.26)
Collobert et al. (2011b)	-	-	88.67	-	-	-
Collobert et al. (2011b) + lexicon	-	-	89.59	-	-	-
Huang et al. (2015)	-	-	90.10	-	-	-
Ratinov and Roth (2009)[18]	91.20	90.50	90.80	82.00	84.95	83.45
Lin and Wu (2009)	-	-	90.90	-	-	-
Finkel and Manning (2009)[19]	-	-	-	84.04	80.86	82.42
Suzuki et al. (2011)	-	-	91.02	-	-	-
Passos et al. (2014)[20]	-	-	90.90	-	-	82.24
Durrett and Klein (2014)	-	-	-	85.22	82.89	84.04
Luo et al. (2015)[21]	**91.50**	91.40	91.20	-	-	-

Table 5: 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, caps = capitalization feature, lex = lexicon features from both SENNA and DBpedia lexicons. For F1 scores, standard deviations are in parentheses.

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:

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

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 $S([x]_1^T, [i]_1^T, \theta')$.

2.6.3 Tagging Scheme

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

2.6.4 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. We found applying dropout to the output nodes[22] of each LSTM layer (Pham et al., 2014) was quite effective in reducing overfitting (Section 4.4). 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 improve upon plain SGD.

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; Pradhan et al., 2013). Table 2 gives an overview of these two different datasets.

For each experiment, we report the average and standard deviation of 10 successful trials.

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

[19] Evaluation on OntoNotes 5.0 done by Pradhan et al. (2013)

[20] Not directly comparable as they evaluated on an earlier version of the corpus with a different data split.

[21] 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.

[22] Adding dropout to inputs seems to have an adverse effect.

Features	BLSTM		BLSTM-CNN		BLSTM-CNN + lex	
	CoNLL	OntoNotes	CoNLL	OntoNotes	CoNLL	OntoNotes
none	76.29 (± 0.29)	77.77 (± 0.37)	83.38 (± 0.20)	82.53 (± 0.40)	87.77 (± 0.29)	83.82 (± 0.19)
emb	88.23 (± 0.23)	82.72 (± 0.23)	90.91 (± 0.20)	86.17 (± 0.22)	**91.62** (± 0.33)	86.28 (± 0.26)
emb + caps	90.67 (± 0.16)	86.19 (± 0.25)	90.98 (± 0.18)	86.35 (± 0.28)	91.55 (± 0.19)*	86.28 (± 0.32)*
emb + caps + lex	**91.43** (± 0.17)	**86.21** (± 0.16)	**91.55** (± 0.19)*	86.28 (± 0.32)*	91.55 (± 0.19)*	86.28 (± 0.32)*
emb + char	-	-	90.88 (± 0.48)	86.08 (± 0.40)	91.44 (± 0.23)	**86.34** (± 0.18)
emb + char + caps	-	-	90.88 (± 0.31)	**86.41** (± 0.22)	91.48 (± 0.23)	86.33 (± 0.26)

Table 6: F1 score results of BLSTM and BLSTM-CNN models with various additional features; emb = Collobert word embeddings, char = character type feature, caps = capitalization feature, lex = lexicon features. Note that starred results are repeated for ease of comparison.

3.1 Dataset Preprocessing

For all datasets, we performed the following preprocessing:

- All digit sequences are replaced by a single "0".
- Before training, we group sentences by word length into mini-batches and shuffle them.

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

The CoNLL-2003 dataset (Tjong Kim Sang and De Meulder, 2003) consists of newswire from the Reuters RCV1 corpus tagged with four types of named entities: location, organization, person, and miscellaneous. As the dataset is small compared to OntoNotes, we trained the model on both the training and development sets after performing hyper-parameter optimization on the development set.

3.3 OntoNotes 5.0 Dataset

Pradhan et al. (2013) compiled a core portion of the OntoNotes 5.0 dataset for the CoNLL-2012 shared task and described a standard train/dev/test split, which we use for our evaluation. Following Durrett and Klein (2014), we applied our model to the portion of the dataset with gold-standard named entity annotations; the New Testaments portion was excluded for lacking gold-standard annotations. This dataset is much larger than CoNLL-2003 and consists of text from a wide variety of sources, such as broadcast conversation, broadcast news, newswire, magazine, telephone conversation, and Web text.

3.4 Hyper-parameter Optimization

We performed two rounds of hyper-parameter optimization and selected the best settings based on development set performance[23]. Table 3 shows the final hyper-parameters, and Table 4 shows the dev set performance of the best models in each round.

In the first round, we performed random search and selected the best hyper-parameters over the development set of the CoNLL-2003 data. We evaluated around 500 hyper-parameter settings. Then, we took the same settings and tuned the learning rate and epochs on the OntoNotes development set. [24]

For the second round, we performed independent hyper-parameter searches on each dataset using Optunity's implementation of particle swarm (Claesen et al.,), as there is some evidence that it is more efficient than random search (Clerc and Kennedy, 2002). We evaluated 500 hyper-parameter settings this round as well. As we later found out that training fails occasionally (Section 3.5) as well as large variation from run to run, we ran the top 5 settings from each dataset for 10 trials each and selected the best one based on averaged dev set performance.

For CoNLL-2003, we found that particle swarm produced better hyper-parameters than random search. However, surprisingly for OntoNotes particle swarm was unable to produce better hyper-parameters than those from the ad-hoc approach in round 1. We also tried tuning the CoNLL-2003 hyper-parameters from round 2 for OntoNotes and that was not any better[25] either.

We trained CoNLL-2003 models for a large num-

[23] Hyper-parameter optimization was done with the BLSTM-CNN + emb + lex feature set, as it had the best performance.

[24] Selected based on dev set performance of a few runs.

[25] The result is 84.41 (± 0.33) on the OntoNotes dev set.

Word Embeddings	CoNLL-2003	OntoNotes
Random 50d	87.77 (± 0.29)	83.82 (± 0.19)
Random 300d	87.84 (± 0.23)	83.76 (± 0.37)
GloVe 6B 50d	91.09 (± 0.15)	86.25 (± 0.24)
GloVe 6B 300d	90.71 (± 0.21)	86.26 (± 0.30)
Google 100B 300d	90.60 (± 0.23)	85.34 (± 0.25)
Collobert 50d	**91.62** (± 0.33)	**86.28** (± 0.26)
Our GloVe 50d	91.41 (± 0.21)	86.24 (± 0.35)
Our Skip-gram 50d	90.76 (± 0.23)	85.70 (± 0.29)

Table 7: F1 scores when the Collobert word vectors are replaced. We tried 50- and 300-dimensional random vectors (Random 50d, Random 300d); GloVe's released vectors trained on 6 billion words (GloVe 6B 50d, GloVe 6B 300d); Google's released 300-dimensional vectors trained on 100 billion words from Google News (Google 100B 300d); and 50-dimensional GloVe and word2vec skip-gram vectors that we trained on Wikipedia and Reuters RCV-1 (Our GloVe 50d, Our Skip-gram 50d).

ber of epochs because we observed that the models did not exhibit overtraining and instead continued to slowly improve on the development set long after reaching near 100% accuracy on the training set. In contrast, despite OntoNotes being much larger than CoNLL-2003, training for more than about 18 epochs causes performance on the development set to decline steadily due to overfitting.

3.5 Excluding Failed Trials

On the CoNLL-2003 dataset, while BLSTM models completed training without difficulty, the BLSTM-CNN models fail to converge around 5∼10% of the time depending on feature set. Similarly, on OntoNotes, 1.5% of trials fail. We found that using a lower learning rate reduces failure rate. We also tried clipping gradients and using AdaDelta and both of them were effective at eliminating such failures by themselves. AdaDelta, however, made training more expensive with no gain in model performance.

In any case, for all experiments we excluded trials where the final F1 score on a subset of training data falls below a certain threshold, and continued to run trials until we obtained 10 successful ones.

For CoNLL-2003, we excluded 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 98 or below 90. For OntoNotes, the threshold was a F1 score of 80 on the last 5,000 sentences of the training set; every trial scored either above 80 or below 75.

3.6 Training and Tagging Speed

On an Intel Xeon E5-2697 processor, training takes about 6 hours while tagging the test set takes about 12 seconds for CoNLL-2003. The times are 10 hours and 60 seconds respectively for OntoNotes.

4 Results and Discussion

Table 5 shows the results for all datasets. To the best of our knowledge, our best models have surpassed the previous highest reported F1 scores for both CoNLL-2003 and OntoNotes. In particular, with no external knowledge other than word embeddings, our model is competitive on the CoNLL-2003 dataset and establishes a new state of the art for OntoNotes, suggesting that given enough data, the neural network automatically learns the relevant features for NER without feature engineering.

4.1 Comparison with FFNNs

We re-implemented the FFNN model of Collobert et al. (2011b) as a baseline for comparison. Table 5 shows that while performing reasonably well on CoNLL-2003, FFNNs are clearly inadequate for OntoNotes, which has a larger domain, showing that LSTM models are essential for NER.

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

The comparison of models in Table 6 shows that on CoNLL-2003, BLSTM-CNN models significantly[26] outperform the BLSTM models when given the same feature set. This effect is smaller and not statistically significant on OntoNotes when capitalization features are added. Adding character type and capitalization features to the BLSTM-CNN models degrades performance for CoNLL and mostly improves performance on OntoNotes, suggesting character-level CNNs can replace hand-crafted character features in some cases, but systems with weak lexicons may benefit from character features.

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

Dropout	CoNLL-2003		OntoNotes 5.0	
	Dev	Test	Dev	Test
-	93.72 (± 0.10)	90.76 (± 0.22)	82.02 (± 0.49)	84.06 (± 0.50)
0.10	93.85 (± 0.18)	90.87 (± 0.31)	83.01 (± 0.39)	84.94 (± 0.25)
0.30	94.08 (± 0.17)	91.09 (± 0.18)	83.61 (± 0.32)	85.44 (± 0.33)
0.50	94.19 (± 0.18)	91.14 (± 0.35)	84.35 (± 0.23)	**86.36** (± 0.28)
0.63	-	-	84.47 (± 0.23)	86.29 (± 0.25)
0.68	**94.31** (± 0.15)	**91.23** (± 0.16)	-	-
0.70	**94.31** (± 0.24)	91.17 (± 0.37)	**84.56** (± 0.40)	86.17 (± 0.25)
0.90	94.17 (± 0.17)	90.67 (± 0.17)	81.38 (± 0.19)	82.16 (± 0.18)

Table 8: F1 score results with various dropout values. Models were trained using only the training set for each dataset. All other experiments use dropout = 0.68 for CoNLL-2003 and dropout = 0.63 for OntoNotes 5.0.

4.3 Word Embeddings

Table 5 and Table 7 show that we obtain a large, significant[27] improvement when trained word embeddings are used, as opposed to random embeddings, regardless of the additional features used. This is consistent with Collobert et. al. (2011b)'s results.

Table 7 compares the performance of different word embeddings in our best model in Table 5 (BLSTM-CNN + emb + lex). For CoNLL-2003, the 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 the CoNLL-2003 dataset, whereas the other embeddings are not[28]. 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). As shown in Table 7, our GloVe embeddings improved significantly[29] over publicly available embeddings on CoNLL-2003, and our word2vec skip-gram embeddings improved significantly[30] 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.

4.4 Effect of Dropout

Table 8 compares the result of various dropout values for each dataset. The models are trained using only the training set for each dataset to isolate the effect of dropout on both dev and test sets. All other hyper-parameters and features remain the same as our best model in Table 5. In both datasets and on both dev and test sets, dropout is essential for state of the art performance, and the improvement is statistically significant[31]. Dropout is optimized on the dev set, as described in Section 3.4. Hence, the chosen

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

[28] To make a direct comparison to Collobert et al. (2011b), we do not exclude the CoNLL-2003 NER task test data from the word vector training data. While it is possible that this difference could be responsible for the disparate performance of word vectors, the CoNLL-2003 training data comprises only 20k out of 800 million words, or 0.00002% of the total data; in an unsupervised training scheme, the effects are likely negligible.

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

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

[31] Wilcoxon rank sum test, no dropout vs. best setting: $p < 0.001$ for the CoNLL-2003 test set, $p < 0.0001$ for the OntoNotes 5.0 test set, $p < 0.0005$ for all others.

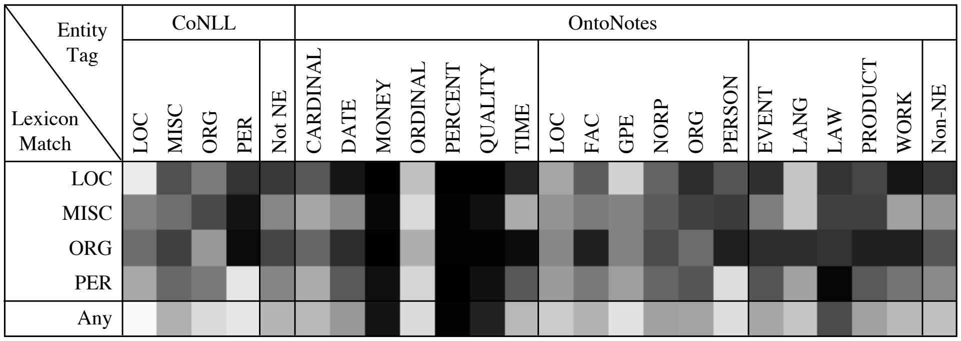

Figure 5: Fraction of named entities of each tag category matched completely by entries in each lexicon category of the SENNA/DBpedia combined lexicon. White = higher fraction.

value may not be the best-performing in Table 8.

4.5 Lexicon Features

Table 6 shows that on the CoNLL-2003 dataset, using features from both the SENNA lexicon and our proposed DBpedia lexicon provides a significant[32] improvement and allows our model to clearly surpass the previous state of the art.

Unfortunately the difference is minuscule for OntoNotes, most likely because our lexicon does not match DBpedia categories well. Figure 5 shows that on CoNLL-2003, lexicon coverage is reasonable and matches the tags set for everything except the catch-all `MISC` category. For example, `LOC` entries in lexicon match mostly `LOC` named entities and vice versa. However, on OntoNotes, the matches are noisy and correspondence between lexicon match and tag category is quite ambiguous. For example, all lexicon categories have spurious matches in unrelated named entities like `CARDINAL`, and `LOC`, `GPE`, and `LANGUAGE` entities all get a lot of matches from the `LOC` category in the lexicon. In addition, named entities in categories like `NORP`, `ORG`, `LAW`, `PRODUCT` receive little coverage. The lower coverage, noise, and ambiguity all contribute to the disappointing performance. This suggests that the DBpedia lexicon construction method needs to be improved. A reasonable place to start would be the DBpedia category to OntoNotes NE tag mappings.

In order to isolate the contribution of each lexicon and matching method, we compare different sources and matching methods on a BLSTM-CNN model with randomly initialized word embeddings and no other features or sources of external knowledge. Table 9 shows the results. In this weakened model, both lexicons contribute significant[33] improvements over the baseline.

Compared to the SENNA lexicon, our DBpedia lexicon is noisier but has broader coverage, which explains why when applying it using the same method as Collobert et al. (2011b), it performs worse on CoNLL-2003 but better on OntoNotes – a dataset containing many more obscure named entities. However, we suspect that the method of Collobert et al. (2011b) is not noise resistant and therefore unsuitable for our lexicon because it fails to distinguish exact and partial matches[34] and does not set a minimum length for partial matching.[35] Instead, when we apply our superior partial matching algorithm and BIOES encoding with our DBpedia lexicon, we gain a significant[36] improvement, allowing our lexicon to perform similarly to the SENNA lexicon. Unfortunately, as we could not reliably remove partial entries from the SENNA lexicon, we were unable to investigate whether or not our lexicon matching method would help in that lexicon.

In addition, using both lexicons together as distinct features provides a further improvement[37] on CoNLL-2003, which we suspect is because the lexi-

[32] Wilcoxon rank sum test, $p < 0.001$.

[33] Wilcoxon rank sum test, $p < 0.05$ for SENNA-Exact-BIOES, $p < 0.005$ for all others.

[34] We achieve this by using BIOES encoding and prioritizing exact matches over partial matches.

[35] Matching only the first word of a long entry is not very useful; this is not a problem in the SENNA lexicon because 99% of its entries contain only 3 tokens or less.

[36] Wilcoxon rank sum test, $p < 0.001$.

[37] Wilcoxon rank sum test, $p < 0.001$.

Lexicon	Matching	Encoding	CoNLL-2003	OntoNotes
No lexicon	-	-	83.38 (± 0.20)	82.53 (± 0.40)
SENNA	Exact	YN	86.21 (± 0.39)	83.24 (± 0.33)
	Exact	BIOES	86.14 (± 0.48)	83.01 (± 0.52)
DBpedia	Exact	YN	84.93 (± 0.30)	83.15 (± 0.26)
	Exact	BIOES	85.02 (± 0.23)	83.39 (± 0.39)
	Partial	YN	85.72 (± 0.45)	83.25 (± 0.33)
	Partial	BIOES	86.18 (± 0.56)	**83.97** (± 0.38)
	Collobert's method		85.01 (± 0.31)	83.24 (± 0.26)
Both	Best combination		**87.77** (± 0.29)	83.82 (± 0.19)

Table 9: Comparison of lexicon and matching/encoding methods over the BLSTM-CNN model employing random embeddings and no other features. When using both lexicons, the best combination of matching and encoding is Exact-BIOES for SENNA and Partial-BIOES for DBpedia. Note that the SENNA lexicon already contains "partial entries" so exact matching in that case is really just a more primitive form of partial matching.

cons are complementary; the SENNA lexicon is relatively clean and tailored to newswire, whereas the DBpedia lexicon is noisier but has high coverage.

4.6 Analysis of OntoNotes Performance

Table 10 shows the per-genre breakdown of the OntoNotes results. As expected, our model performs best on clean text like broadcast news (BN) and newswire (NW), and worst on noisy text like telephone conversation (TC) and Web text (WB). Our model also substantially improves over previous work on all genres except TC, where the small size of the training data likely hinders learning. Finally, the performance characteristics of our model appear to be quite different than the previous CRF models (Finkel and Manning, 2009; Durrett and Klein, 2014), likely because we apply a completely different machine learning method.

5 Related Research

Named entity recognition is a task with a long history. In this section, we summarize the works we compare with and that influenced our approach.

5.1 Named Entity Recognition

Most recent approaches to NER have been characterized by the use of CRF, SVM, and perceptron models, where performance is heavily dependent on feature engineering. Ratinov and Roth (2009) used non-local features, a 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 skip-gram model. In order to combat the problem of sparse features, Suzuki et al. (2011) employed large-scale unlabelled data to perform feature reduction and achieved an F1 score of 91.02 on CoNLL-2003, which is the current state of the art for systems without external knowledge.

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 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 interdependencies 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 involving neural networks. Early attempts were hindered by

[38] We downloaded their publicly released software and model to perform the per-genre evaluation.

Model	BC	BN	MZ	NW	TC	WB
Test set size (# tokens)	32,576	23,557	18,260	51,667	11,015	19,348
Test set size (# entities)	1,697	2,184	1,163	4,696	380	1,137
Finkel and Manning (2009)	78.66	87.29	82.45	85.50	67.27	72.56
Durrett and Klein (2014)[38]	78.88	87.39	82.46	87.60	**72.68**	76.17
BLSTM-CNN	81.26	86.87	79.94	85.27	67.82	72.11
BLSTM-CNN + emb	85.05	**89.93**	84.31	88.35	72.44	77.90
BLSTM-CNN + emb + lex	**85.23**	**89.93**	**84.45**	**88.39**	72.39	**78.38**

Table 10: Per genre F1 scores on OntoNotes. BC = broadcast conversation, BN = broadcast news, MZ = magazine, NW = newswire, TC = telephone conversation, WB = blogs and newsgroups

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 only 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 word vectors trained using self-organizing maps and 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 deep FFNN 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, 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 model.

There have been various other similar architecture proposed for various sequential labeling NLP tasks. Huang et al. (2015) used a BLSTM for the POS-tagging, chunking, and NER tasks, but they employed heavy feature engineering instead of using a CNN to automatically extract character-level features. Labeau et al. (2015) used a BRNN with character-level CNNs to perform German POS-tagging; our model differs in that we use the more powerful LSTM unit, which we found to perform better than RNNs in preliminary experiments, and that we employ word embeddings, which is much more important in NER than in POS tagging. Ling et al. (2015) used both word- and character-level BLSTMs to establish the current state of the art for English POS tagging. While using BLSTMs instead of CNNs allows extraction of more sophisticated character-level features, we found in preliminary experiments that for NER it did not perform significantly better than CNNs and was substantially more computationally expensive to train.

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 two major datasets for NER, suggesting that the model is capable of learning complex relationships from large amounts of data.

Preliminary evaluation of our partial matching lexicon algorithm suggests that performance could be further improved through more flexible application of existing lexicons. Evaluation of existing word embeddings suggests that the domain of training data is as important as the training algorithm.

More effective construction and application of lexicons and word embeddings 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.

Acknowledgments

This research was supported by Honda Research Institute Japan Co., Ltd. The authors would like to thank Collobert et al. (2011b) for releasing SENNA with its word vectors and lexicon, 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. In *The semantic web*, pages 722–735. 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*. 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. Association for Computational Linguistics.

Marc Claesen, Jaak Simm, Dusan Popovic, Yves Moreau, and Bart De Moor. Easy hyperparameter search using Optunity. In *Proceedings of the International Workshop on Technical Computing for Machine Learning and Mathematical Engineering*.

Maurice Clerc and James Kennedy. 2002. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73.

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 *Proceedings of 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.

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.

Jenny Rose Finkel and Christopher D Manning. 2009. Joint parsing and named entity recognition. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 326–334. Association for Computational Linguistics.

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 *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649.

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*, 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.

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. 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*, pages 1030–1038. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530. 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. Association for Computational Linguistics.

Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukar Burget, and Jan Cernocky. 2011. RNNLM-recurrent neural network language modeling toolkit. In *Proceedings 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 *Proceedings of the Twenty-seventh Annual Conference on 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. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 78–86. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 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 *Proceedings of the Symposium on Computational Intelligence and Learning*, pages 58–66. Citeseer.

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

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2013. Towards robust linguistic analysis using OntoNotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152. 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.

Cıcero Santos, Victor Guimaraes, RJ Niterói, and Rio de Janeiro. 2015. Boosting named entity recognition with neural character embeddings. In *Proceedings of the Fifth Named Entities Workshop*, pages 25–33.

Jun Suzuki, Hideki Isozaki, and Masaaki Nagata. 2011. Learning condensed feature representations from large unsupervised data sets for supervised learning. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers*, pages 636–641. Association for Computational Linguistics.

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*, 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.