Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation

Wang Ling Tiago Luís Luís Marujo Ramón Fernandez Astudillo
Silvio Amir Chris Dyer Alan W Black Isabel Trancoso

L^{2}F Spoken Systems Lab, INESC-ID, Lisbon, Portugal
Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, USA
Instituto Superior Técnico, Lisbon, Portugal
{lingwang,lmarujo,cdyer,awb}@cs.cmu.edu
{ramon.astudillo,samir,tmcl,isabel.trancoso}@inesc-id.pt

Abstract

We introduce a model for constructing vector representations of words by composing characters using bidirectional LSTMs. Relative to traditional word representation models that have independent vectors for each word type, our model requires only a single vector per character type and a fixed set of parameters for the compositional model. Despite the compactness of this model and, more importantly, the arbitrary nature of the form–function relationship in language, our "composed" word representations yield state-of-the-art results in language modeling and part-of-speech tagging. Benefits over traditional baselines are particularly pronounced in morphologically rich languages (e.g., Turkish).

1 Introduction

Good representations of words are important for good generalization in natural language processing applications. Of central importance are vector space models that capture functional (i.e., semantic and syntactic) similarity in terms of geometric locality. However, when word vectors are learned—a practice that is becoming increasingly common—most models assume that each word type has its own vector representation that can vary independently of other model components. This paper argues that this independence assumption is inherently problematic, in particular in morphologically rich languages (e.g., Turkish). In such languages, a more reasonable assumption would be that orthographic (formal) similarity is evidence for functional similarity.

However, it is manifestly clear that similarity in form is neither a necessary nor sufficient condition for similarity in function: small orthographic differences may correspond to large semantic or syntactic differences (*butter* vs. *batter*), and large orthographic differences may obscure nearly perfect functional correspondence (*rich* vs. *affluent*). Thus, any orthographically aware model must be able to capture *non-compositional* effects in addition to more regular effects due to, e.g., morphological processes. To model the complex form–function relationship, we turn to long short-term memories (LSTMs), which are designed to be able to capture complex non-linear and non-local dynamics in sequences (Hochreiter and Schmidhuber, 1997). We use bidirectional LSTMs to "read" the character sequences that constitute each word and combine them into a vector representation of the word. This model assumes that each character type is associated with a vector, and the LSTM parameters encode both idiosyncratic lexical and regular morphological knowledge.

To evaluate our model, we use a vector-based model for part-of-speech (POS) tagging and for language modeling, and we report experiments on these tasks in several languages comparing to baselines that use more traditional, orthographically-unaware parameterizations. These experiments show: (i) our character-based model is able to generate similar representations for words that are semantically and syntactically similar, even for words are orthographically distant (e.g., *October* and *January*); our model achieves improvements over word lookup tables using only a fraction of the number of parameters in two tasks; (iii) our model obtains state-of-the-art performance on POS tagging (including establishing a new best performance in English); and

(iv) performance improvements are especially dramatic in morphologically rich languages.

The paper is organized as follows: Section 2 presents our character-based model to generate word embeddings. Experiments on Language Modeling and POS tagging are described in Sections 4 and 5. We present related work in Section 6; and we conclude in Section 7.

2 Word Vectors and Wordless Word Vectors

It is commonplace to represent words as vectors. In contrast to naïve models in which all word types in a vocabulary V are equally different from each other, vector space models capture the intuition that words may be different or similar along a variety of dimensions. Learning vector representations of words by treating them as optimizable parameters in various kinds of language models has been found to be a remarkably effective means for generating vector representations that perform well in other tasks (Collobert et al., 2011; Kalchbrenner and Blunsom, 2013; Liu et al., 2014; Chen and Manning, 2014). Formally, such models define a matrix $\mathbf{P} \in \mathbb{R}^{d\times|V|}$, which contains d parameters for each word in the vocabulary V. For a given word type $w \in V$, a column is selected by right-multiplying $\mathbf{P}$ by a one-hot vector of length $|V|$, which we write $\mathbf{1}_w$, that is zero in every dimension except for the element corresponding to w. Thus, $\mathbf{P}$ is often referred to as word lookup table and we shall denote by $\mathbf{e}^W_w \in \mathbb{R}^d$ the embedding obtained from a word lookup table for w as $\mathbf{e}^W_w = \mathbf{P}\cdot\mathbf{1}_w$. This allows tasks with low amounts of annotated data to be trained jointly with other tasks with large amounts of data and leverage the similarities in these tasks. A common practice to this end is to initialize the word lookup table with the parameters trained on an unsupervised task. Some examples of these include the skip-n-gram and CBOW models of Mikolov et al. (2013).

2.1 Problem: Independent Parameters

There are two practical problems with word lookup tables. Firstly, while they can be pretrained with large amounts of data to learn semantic and syntactic similarities between words, each vector is independent. That is, even though models based on word lookup tables are often observed to learn that *cats*, *kings* and *queens* exist in roughly the same linear correspondences to each other as *cat*, *king* and *queen* do, the model does not represent the fact that adding an s at the end of the word is evidence for this transformation. This means that word lookup tables cannot generate representations for previously unseen words, such as *Frenchification*, even if the components, *French* and *-ification*, are observed in other contexts.

Second, even if copious data is available, it is impractical to actually store vectors for all word types. As each word type gets a set of parameters d, the total number of parameters is $d\times|V|$, where $|V|$ is the size of the vocabulary. Even in relatively morphological poor English, the number of word types tends to scale to the order of hundreds of thousands, and in noisier domains, such as online data, the number of word types raises considerably. For instance, in the English wikipedia dump with 60 million sentences, there are approximately 20 million different lowercased and tokenized word types, each of which would need its own vector. Intuitively, it is not sensible to use the same number of parameters for each word type.

Finally, it is important to remark that it is uncontroversial among cognitive scientists that our lexicon is structured into related forms—i.e., their parameters are not independent. The well-known "past tense debate" between connectionists and proponents of symbolic accounts concerns disagreements about how humans represent knowledge of inflectional processes (e.g., the formation of the English past tense), not whether such knowledge exists (Marslen-Wilson and Tyler, 1998).

2.2 Solution: Compositional Models

Our solution to these problems is to construct a vector representation of a word by composing smaller pieces into a representation of the larger form. This idea has been explored in prior work by composing *morphemes* into representations of words (Luong et al., 2013; Botha and Blunsom, 2014; Soricut and Och, 2015). Morphemes are an ideal primitive for such a model since they are—by definition—the minimal meaning-bearing (or syntax-bearing) units of language. The drawback to such approaches is they depend on a morphological analyzer.

In contrast, we would like to compose representations of *characters* into representations of words. However, the relationship between words forms and their meanings is non-trivial (de Saussure, 1916). While some compositional relationships exist, e.g., morphological processes such as adding *-ing* or *-ly* to a stem have relatively regular effects, many words with lexical similarities convey different meanings, such as, the word pairs *lesson* $\Longleftrightarrow$ *lessen* and *coarse* $\Longleftrightarrow$ *course*.

3 C2W Model

Our compositional character to word (C2W) model is based on bidirectional LSTMs (Graves and Schmidhuber, 2005), which are able to learn complex non-local dependencies in sequence models. An illustration is shown in Figure 1. The input of the C2W model (illustrated on bottom) is a single word type w, and we wish to obtain is a d-dimensional vector used to represent w. This model shares the same input and output of a word lookup table (illustrated on top), allowing it to easily replace then in any network.

As input, we define an alphabet of characters C. For English, this vocabulary would contain an entry for each uppercase and lowercase letter as well as numbers and punctuation. The input word w is decomposed into a sequence of characters $c_1, \ldots, c_m$, where m is the length of w. Each c_i is defined as a one hot vector $\mathbf{1}_{c_i}$, with one on the index of c_i in vocabulary M. We define a projection layer $\mathbf{P}_C \in \mathbb{R}^{d_C \times |C|}$, where d_C is the number of parameters for each character in the character set C. This of course just a character lookup table, and is used to capture similarities between characters in a language (e.g., vowels *vs.* consonants). Thus, we write the projection of each input character c_i as $\mathbf{e}_{c_i} = \mathbf{P}_C \cdot \mathbf{1}_{c_i}$.

Given the input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$, a LSTM computes the state sequence $\mathbf{h}_1, \ldots, \mathbf{h}_{m+1}$ by iteratively applying the following updates:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i)\\
\mathbf{f}_t &= \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f)\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} +\\
&\qquad \mathbf{i}_t \odot \tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c)\\
\mathbf{o}_t &= \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o)\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

where σ is the component-wise logistic sigmoid function, and $\odot$ is the component-wise (Hadamard) product. LSTMs define an extra cell memory $\mathbf{c}_t$, which is combined linearly at each timestamp t. The information that is propagated from $\mathbf{c}_{t-1}$ to $\mathbf{c}_t$ is controlled by the three gates $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$, which determine the what to include from the input $\mathbf{x}_t$, the what to forget from $\mathbf{c}_{t-1}$ and what is relevant to the current state $\mathbf{h}_t$. We write $\mathcal{W}$ to refer to all parameters the LSTM ($\mathbf{W}_{ix}$, $\mathbf{W}_{fx}$, $\mathbf{b}_f$, …). Thus, given a sequence of character representations $\mathbf{e}^C_{c_1}, \ldots, \mathbf{e}^C_{c_m}$ as input, the forward LSTM, yields the state sequence $\mathbf{s}^f_0, \ldots, \mathbf{s}^f_m$, while the backward LSTM receives as input the reverse sequence, and yields states $\mathbf{s}^b_m, \ldots, \mathbf{s}^b_0$. Both LSTMs use a different set of parameters $\mathcal{W}^f$ and $\mathcal{W}^b$. The representation of the word w is obtained by combining the forward and backward states:

$$\mathbf{e}^C_w = \mathbf{D}^f\mathbf{s}^f_m + \mathbf{D}^b\mathbf{s}^b_0 + \mathbf{b}_d,$$

where $\mathbf{D}^f$, $\mathbf{D}^b$ and $\mathbf{b}_d$ are parameters that deter-

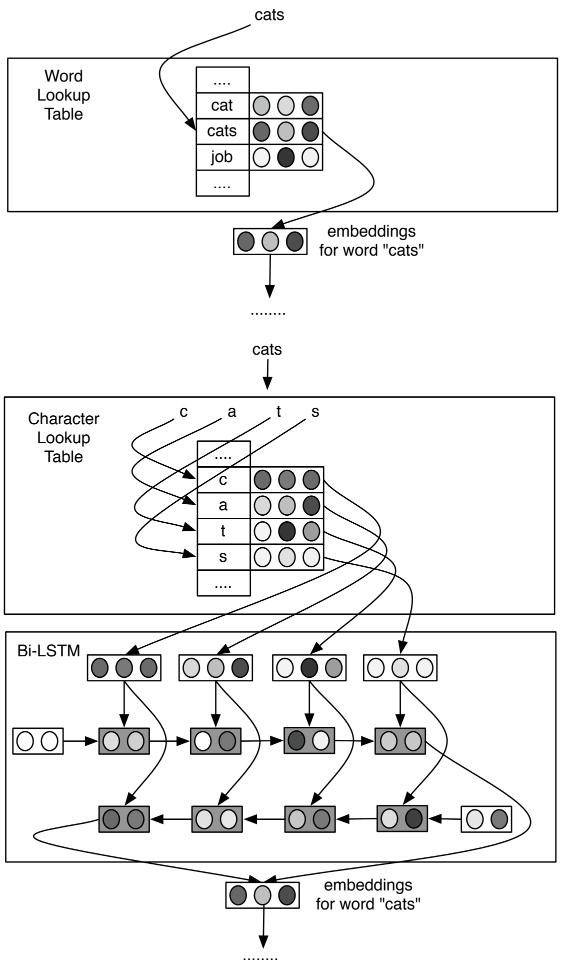

Figure 1: Illustration of the word lookup tables (top) and the lexical Composition Model (bottom). Square boxes represent vectors of neuron activations. Shaded boxes indicate that a non-linearity.

mine how the states are combined.

Caching for Efficiency. Relative to $\mathbf{e}^W_w$, computing $\mathbf{e}^C_w$ is computational expensive, as it requires two LSTMs traversals of length m. However, $\mathbf{e}^C_w$ only depends on the character sequence of that word, which means that unless the parameters are updated, it is possible to cache the value of $\mathbf{e}^C_w$ for each different w's that will be used repeatedly. Thus, the model can keep a list of the most frequently occurring word types in memory and run the compositional model only for rare words. Obviously, caching all words would yield the same performance as using a word lookup table $\mathbf{e}^W_w$, but also using the same amount of memory. Consequently, the number of word types used in cache can be adjusted to satisfy memory vs. performance requirements of a particular application.

At training time, when parameters are changing, repeated words within the same batch only need to be computed once, and the gradient at the output can be accumulated within the batch so that only one update needs to be done per word type. For this reason, it is preferable to define larger batches.

4 Experiments: Language Modeling

Our proposed model is similar to models used to compute composed representations of sentences from words (Cho et al., 2014; Li et al., 2015). However, the relationship between the meanings of individual words and the composite meaning of a phrase or sentence is arguably more regular than the relationship of representations of characters and the meaning of a word. Is our model capable of learning such an irregular relationship? We now explore this question empirically.

Language modeling is a task with many applications in NLP. An effective LM requires syntactic aspects of language to be modeled, such as word orderings (e.g., "John is smart" *vs.* "John smart is"), but also semantic aspects (e.g., "John ate fish" *vs.* "fish ate John"). Thus, if our C2W model only captures regular aspects of words, such as, prefixes and suffixes, the model will yield worse results compared to word lookup tables.

4.1 Language Model

Language modeling amounts to learning a function that computes the log probability, $\log p(\boldsymbol{w})$, of a sentence $\boldsymbol{w} = (w_1, \ldots, w_n)$. This quantity can be decomposed according to the chain rule into the sum of the conditional log probabilities $\sum_{i=1}^{n} \log p(w_i \mid w_1, \ldots, w_{i-1})$. Our language model computes $\log p(w_i \mid w_1, \ldots, w_{i-1})$ by composing representations of words $w_1, \ldots, w_{i-1}$ using an recurrent LSTM model (Mikolov et al., 2010; Sundermeyer et al., 2012).

The model is illustrated in Figure 2, where we observe on the first level that each word w_i is projected into their word representations. This can be done by using word lookup tables $\mathbf{e}^W_{w_i}$, in which case, we will have a regular recurrent language model. To use our C2W model, we can simply replace the word lookup table with the model $f(w_i) = \mathbf{e}^C_{w_i}$. Each LSTM block $\mathbf{s}_i$, is used to predict word w_{i+1}. This is performed by projecting the s_i into a vector of size of the vocabulary V and performing a softmax.

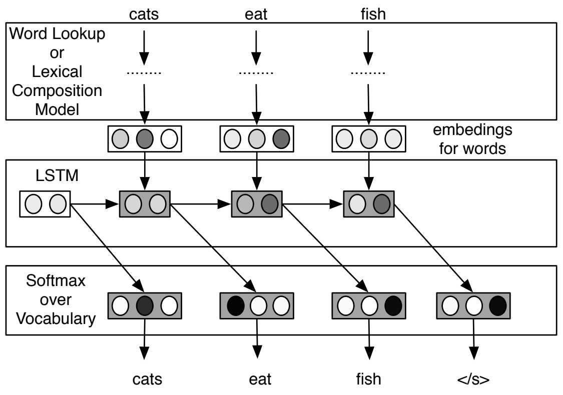

Figure 2: Illustration of our neural network for Language Modeling.

The softmax is still simply a $d \times V$ table, which encodes the likelihood of every word type in a given context, which is a closed-vocabulary model. Thus, at test time out-of-vocabulary (OOV) words cannot be addressed. A strategy that is generally applied is to prune the vocabulary V by replacing word types with lower frequencies as an OOV token. At test time, the probability of words not in vocabulary is estimated as the OOV token. Thus, depending on the number of word types that are pruned, the global perplexities may decrease, since there are fewer outcomes in the softmax, which makes the absolute value of perplexity not informative when comparing models of different vocabulary sizes. Yet, the relative perplexity between different models indicates which models can better predict words based on their contexts.

To address OOV words in the baseline setup, these are replaced by an unknown token, and also associated with a set of embeddings. During training, word types that occur once are replaced with the unknown token stochastically with 0.5 probability. The same process is applied at the character level for the C2W model.

4.2 Experiments

Datasets We look at the language model performance on English, Portuguese, Catalan, German and Turkish, which have a broad range of morphological typologies. While all these languages contain inflections, in agglutinative languages affixes tend to be unchanged, while in fusional languages they are not. For each language, Wikipedia articles were randomly extracted until 1 million words are obtained and these were used for training. For development and testing, we extracted an additional set of 20,000 words.

Setup We define the size of the word representation d to 50. In the C2W model requires setting the dimensionality of characters d_C and current states d_{CS}. We set $d_C = 50$ and $d_{CS} = 150$. Each LSTM state used in the language model sequence s_i is set to 150 for both states and cell memories. Training is performed with mini-batch gradient descent with 100 sentences. The learning rate and momentum were set to 0.2 and 0.95. The softmax over words is always performed on lowercased words. We restrict the output vocabulary to the most frequent 5000 words. Remaining word types will be replaced by an unknown token, which must also be predicted. The word representation layer is still performed over all word types (i.e., completely open vocabulary). When using word lookup tables, the input words are also lowercased, as this setup produces the best results. In the C2W, case information is preserved.

Evaluation is performed by computing the perplexities over the test data, and the parameters that yield the highest perplexity over the development data are used.

Perplexities Perplexities over the testset are reported on Table 4. From these results, we can see that in general, it is clear that C2W always outperforms word lookup tables (row “Word”), and that improvements are especially pronounced in Turkish, which is a highly morphological language, where word meanings differ radically depending on the suffixes used (*evde* → *in the house vs. evden* → *from the house*).

	Fusional			Agglutinative	
Perplexity	EN	PT	CA	DE	TR
5-gram KN	70.72	58.73	39.83	59.07	52.87
Word	59.38	46.17	35.34	43.02	44.01
C2W	**57.39**	**40.92**	**34.92**	**41.94**	**32.88**
#Parameters					
Word	4.3M	4.2M	4.3M	6.3M	5.7M
C2W	**180K**	**178K**	**182K**	**183K**	**174K**

Table 1: Language Modeling Results

Number of Parameters As for the number of parameters (illustrated for block “#Parameters”), the number of parameters in word lookup tables is $V \times d$. If a language contains 80,000 word types (a conservative estimate in morphologically rich languages), 4 million parameters would be necessary. On the other hand, the compositional model consists of 8 matrices of dimensions $d_{CS} \times d_C + 2d_{CS}$. Additionally, there is also the matrix that combines the forward and backward states of size $d \times 2d_{CS}$. Thus, the number of parameters is roughly 150,000 parameters—substantially fewer. This model also needs a character lookup table with d_C parameters for each entry. For English, there are 618 characters, for an additional 30,900 parameters. So the total number of parameters for English is roughly 180,000 parameters (2 to 3 parameters per word type), which is an order of magnitude lower than word lookup tables.

Performance As for efficiency, both representations can label sentences at a rate of approximately 300 words per second during training. While this is surprising, due to the fact that the C2W model requires a composition over characters, the main bottleneck of the system is the softmax over the vocabulary. Furthermore, caching is used to avoid composing the same word type twice in the same batch. This shows that the C2W model, is relatively fast compared operations such as a softmax.

Representations of (nonce) words While is is promising that the model is not simply learning lexical features, what is most interesting is that the model can propose embeddings for nonce words, in stark contrast to the situation observed with lookup table models. We show the 5-most-similar in-vocabulary words (measured with cosine similarity) as computed by our character model on two

increased	*John*	*Noahshire*	*phding*
reduced	Richard	Nottinghamshire	mixing
improved	George	Bucharest	modelling
expected	James	Saxony	styling
decreased	Robert	Johannesburg	blaming
targeted	Edward	Gloucestershire	christening

Table 2: Most-similar in-vocabular words under the C2W model; the two query words on the left are in the training vocabulary, those on the right are nonce (invented) words.

in-vocabulary words and two nonce words[1].This makes our model generalize significantly better than lookup tables that generally use unknown tokens for OOV words. Furthermore, this ability to generalize is much more similar to that of human beings, who are able to infer meanings for new words based on its form.

5 Experiments: Part-of-speech Tagging

As a second illustration of the utility of our model, we turn to POS tagging. As morphology is a strong indicator for syntax in many languages, a much effort has been spent engineering features (Nakagawa et al., 2001; Mueller et al., 2013). We now show that some of these features can be learnt automatically using our model.

5.1 Bi-LSTM Tagging Model

Our tagging model is likewise novel, but very straightforward. It builds a Bi-LSTM over words as illustrated in Figure 3. The input of the model is a sequence of features $f(w_1), \ldots, f(w_n)$. Once again, word vectors can either be generated using the C2W model $f(w_i) = \mathbf{e}^C_{w_i}$, or word lookup tables $f(w_i) = \mathbf{e}^W_{w_i}$. We also test the usage of hand-engineered features, in which case $f_1(w_i), \ldots, f_n(w_i)$. Then, the sequential features $f(w_1), \ldots, f(w_n)$ are fed into a bidirectional LSTM model, obtaining the forward states $\mathbf{s}^f_0, \ldots, \mathbf{s}^f_n$ and the backward states $\mathbf{s}^b_{N+1}, \ldots, \mathbf{s}^b_0$. Thus, state $\mathbf{s}^f_i$ contains the information of all words from 0 to i and $\mathbf{s}^b_i$ from n to i. The forward and backward states are combined, for each index from 1 to n, as follows:

$$\mathbf{l}_i = \tanh(\mathbf{L}^f\mathbf{s}^f_i + \mathbf{L}^b\mathbf{s}^b_i + \mathbf{b}_l),$$

where $\mathbf{L}^f$, $\mathbf{L}^b$ and $\mathbf{b}_l$ are parameters defining how the forward and backward states are combined. The size of the forward $\mathbf{s}^f$ and backward states $\mathbf{s}^b$ and the combined state $\mathbf{l}$ are hyperparameters of the model, denoted as d^f_{WS}, d^b_{WS} and d_{WS}, respectively. Finally, the output labels for index i are obtained as a softmax over the POS tagset, by projecting the combined state $\mathbf{l}_i$.

[1]software submitted as supplementary material

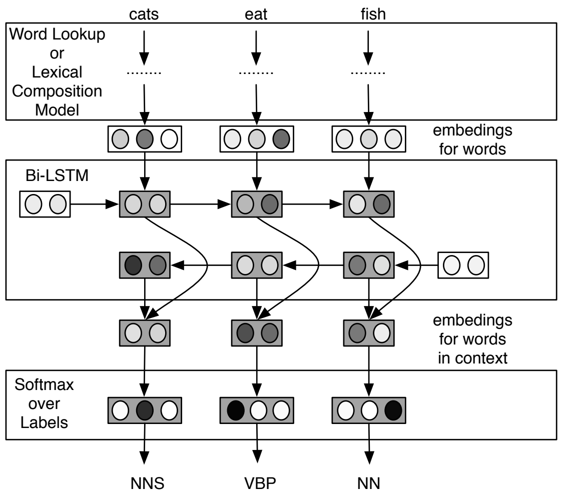
cats
eat
fish
Word Lookup or Lexical Composition Model
embedings for words
Bi-LSTM
embedings for words in context
Softmax over Labels
NNS
VBP
NN

Figure 3: Illustration of our neural network for POS tagging.

5.2 Experiments

Datasets For English, we conduct experiments on the Wall Street Journal of the Penn Treebank dataset (Marcus et al., 1993), using the standard splits (sections 1–18 for train, 19–21 for tuning and 22–24 for testing). We also perform tests on 4 other languages, which we obtained from the CoNLL shared tasks (Martí et al., 2007; Brants et al., 2002; Afonso et al., 2002; Atalay et al., 2003). While the PTB dataset provides standard train, tuning and test splits, there are no tuning sets in the datasets in other languages, so we withdraw the last 100 sentences from the training dataset and use them for tuning.

Setup The POS model requires two sets of hyperparameters. Firstly, words must be converted into continuous representations and the same hyperparametrization as in language modeling (Section 4) is used. Additionally, we also compare to the convolutional model of Santos and Zadrozny (2014), which also requires the dimensionality for characters and the word representation size, which are set to 50 and 150, respectively. Secondly, words representations are combined to encode context. Our POS tagger has three hyperparameters d^f_{WS}, d^b_{WS} and d_{WS}, which correspond to the sizes of LSTM states, and are all set to 50. As for the learning algorithm, use the same setup (learning rate, momentum and mini-batch sizes) as used in language modeling.

Once again, we replace OOV words with an unknown token, in the setup that uses word lookup tables, and the same with OOV characters in the C2W model. In setups using pre-trained word embeddings, we consider a word an OOV if it was not seen in the labelled training data as well as in the unlabeled data used for pre-training.

Compositional Model Comparison A comparison of different recurrent neural networks for the C2W model is presented in Table 3. We used our proposed tagger tagger in all experiments and results are reported for the English Penn Treebank. Results on label accuracy test set is shown in the column "acc". The number of parameters in the word composition model is shown in the column "parameters". Finally, the number of words processed at test time per second are shown in column "words/sec".

We observe that approaches using RNN yield worse results than their LSTM counterparts with a difference of approximately 2%. This suggests that while regular RNNs can learn shorter character sequence dependencies, they are not ideal to learn longer dependencies. LSTMs, on the other hand, seem to effectively obtain relatively higher results, on par with using word look up tables (row "Word Lookup"), even when using forward (row "Forward LSTM") and backward (row "Backward LSTM") LSTMs individually. The best results are obtained using the bidirectional LSTM (row "Bi-LSTM"), which achieves an accuracy of 97.29% on the test set, surpassing the word lookup table. The convolution model (Santos and Zadrozny, 2014) obtained slightly lower results (row "Convolutional (S&Z)"), we think this is because the convolutional model uses a max-pooling layer over series of window convolutions. As order is only perserved within windows, longer distance dependences are unobserved.

There are approximately 40k lowercased word types in the training data in the PTB dataset. Thus, a word lookup table with 50 dimensions per type contains approximately 2 million parameters. In the C2W models, the number of characters types (including uppercase and lowercase) is approximately 80. Thus, the character look up table consists of only 4k parameters, which is negligible compared to the number of parameters in the compositional model, which is once again 150k parameters. One could argue that results in the Bi-LSTM model are higher than those achieved by other models as it contains more parameters, so we set the state size $d_{CS} = 50$ (row "Bi-LSTM $d_{CS} = 50$") and obtained similar results.

	acc	parameters	words/sec
Word Lookup	96.97	2000k	6K
Convolutional (S&Z)	96.80	42.5k	4K
Forward RNN	95.66	17.5k	4K
Backward RNN	95.52	17.5k	4K
Bi-RNN	95.93	40k	3K
Forward LSTM	97.12	80k	3K
Backward LSTM	97.08	80k	3K
Bi-LSTM $d_{CS} = 50$	97.22	70k	3K
Bi-LSTM	**97.36**	150k	2K

Table 3: POS accuracy results for the English PTB using word representation models.

In terms of computational speed, we can observe that there is a more significant slowdown when applying the C2W models compared to language modeling. This is because there is no longer a softmax over the whole word vocabulary as the main bottleneck of the network. However, we can observe that while the Bi-LSTM system is 3 times slower, it is does not significantly hurt the performance of the system.

Results on Multiple Languages Results on 5 languages are shown in Table 4. In general, we can observe that the model using word lookup tables (row "Word") performs consistently worse than the C2W model (row "C2W"). We also compare our results with Stanford's POS tagger, with the default set of features, found in Table 4. Results using these tagger are comparable or better than state-of-the-art systems. We can observe that in most cases we can slightly outperform the scores obtained using their tagger. This is a promising result, considering that we use the same training data and do not handcraft any features. Furthermore, we can observe that for Turkish, our results are significantly higher (>4%).

Comparison with Benchmarks Most state-of-the-art POS tagging systems are obtained by either learning or handcrafting good lexical features (Manning, 2011; Sun, 2014) or using ad-

System	Fusional			Agglutinative	
	EN	PT	CA	DE	TR
Word	96.97	95.67	98.09	97.51	83.43
C2W	**97.36**	97.47	**98.92**	**98.08**	**91.59**
Stanford	97.32	**97.54**	98.76	97.92	87.31

Table 4: POS accuracies on different languages

ditional raw data to learn features in an unsupervised fashion. Generally, optimal results are obtained by performing both. Table 5 shows the current Benchmarks in this task for the English PTB. Accuracies on the test set is reported on column "acc". Columns "+feat" and "+data" define whether hand-crafted features are used and whether additional data was used. We can see that even without feature engineering or unsupervised pretraining, our C2W model (row "C2W") is on par with the current state-of-the-art system (row "structReg"). However, if we add hand-crafted features, we can obtain further improvements on this dataset (row "C2W + features").

However, there are many words that do not contain morphological cues to their part-of-speech. For instance, the word *snake* does not contain any morphological cues that determine its tag. In these cases, if they are not found labelled in the training data, the model would be dependent on context to determine their tags, which could lead to errors in ambiguous contexts. Unsupervised training methods such as the Skip-n-gram model (Mikolov et al., 2013) can be used to pretrain the word representations on unannotated corpora. If such pretraining places *cat*, *dog* and *snake* near each other in vector space, and the supervised POS data contains evidence that *cat* and *dog* are nouns, our model will be likely to label *snake* with the same tag.

We train embeddings using English wikipedia with the dataset used in (Ling et al., 2015), and the Structured Skip-n-gram model. Results using pre-trained word lookup tables and the C2W with the pre-trained word lookup tables as additional parameters are shown in rows "word(sskip)" and "C2W + word(sskip)". We can observe that both systems can obtain improvements over their random initializations (rows "word" and (C2W)).

Finally, we also found that when using the C2W model in conjunction pre-trained word embeddings, that adding a non-linearity to the representations extracted from the C2W model $\mathbf{e}_w^C$ improves the results over using a simple linear transformation (row "C2W(tanh)+word (sskip)"). This setup, obtains 0.28 points over the current state-of-the-art system(row "SCCN").

	+feat	+data	acc
word	no	no	96.70
C2W	no	no	**97.36**
word+features	yes	no	97.34
C2W+features	yes	no	**97.57**
Stanford 2.0 (Manning, 2011)	yes	no	97.32
structReg (Sun, 2014)	yes	no	97.36
word (sskip)	no	yes	97.42
C2W+word (sskip)	no	yes	97.54
C2W(tanh)+word (sskip)	no	yes	**97.78**
Morče (Spoustová et al., 2009)	yes	yes	97.44
SCCN (Søgaard, 2011)	yes	yes	97.50

Table 5: POS accuracy result comparison with state-of-the-art systems for the English PTB.

5.3 Discussion

It is important to refer here that these results do not imply that our model always outperforms existing benchmarks, in fact in most experiments, results are typically fairly similar to existing systems. Even in Turkish, using morphological analysers in order to extract additional features could also accomplish similar results. The goal of our work is not to overcome existing benchmarks, but show that much of the feature engineering done in the benchmarks can be learnt automatically from the task specific data. More importantly, we wish to show large dimensionality word look tables can be compacted into a lookup table using characters and a compositional model allowing the model scale better with the size of the training data. This is a desirable property of the model as data becomes more abundant in many NLP tasks.

6 Related Work

Our work, which learns representations without relying on word lookup tables has not been explored to our knowledge. In essence, our model attempts to learn lexical features automatically while compacting the model by reducing the redundancy found in word lookup tables. Individually, these problems have been the focus of research in many areas.

Lexical information has been used to augment word lookup tables. Word representation learning can be thought of as a process that takes a string as input representing a word and outputs a set of values that represent a word in vector space. Using word lookup tables is one possible approach to accomplish this. Many methods have been used to augment this model to learn lexical features with an additional model that is jointly maximized with the word lookup table. This is generally accomplished by either performing a component-wise addition of the embeddings produced by word lookup tables (Chen et al., 2015), and that generated by the additional lexical model, or simply concatenating both representations (Santos and Zadrozny, 2014). Many models have been proposed, the work in (Collobert et al., 2011) refers that additional features sets F_i can be added to the one-hot representation and multiple lookup tables $\mathbf{I}_{F_i}$ can be learnt to project each of the feature sets to the same low-dimensional vector $\mathbf{e}^W_w$. For instance, the work in (Botha and Blunsom, 2014) shows that using morphological analyzers to generate morphological features, such as stems, prefixes and suffixes can be used to learn better representations for words. A problem with this approach is the fact that the model can only learn from what has been defined as feature sets. The models proposed in (Santos and Zadrozny, 2014; Chen et al., 2015) allow the model to arbitrary extract meaningful lexical features from words by defining compositional models over characters. The work in (Chen et al., 2015) defines a simple compositional model by summing over all characters in a given word, while the work in (Santos and Zadrozny, 2014) defines a convolutional network, which combines windows of characters and a max-pooling layer to find important morphological features. The main drawback of these methods is that character order is often neglected, that is, when summing over all character embeddings, words such as *dog* and *god* would have the same representation according to the lexical model. Convolutional model are less susceptible to these problems as they combine windows of characters at each convolution, where the order within the window is preserved. However, the order between extracted windows is not, so the problem still persists for longer words, such as those found in agglutinative languages. Yet, these approaches work in conjunction with a word lookup table, as they compensate for this inability. Aside from neural approaches, character-based models have been applied to address multiple lexically oriented tasks, such as transliteration (Kang and Choi, 2000) and twitter normalization (Xu et al., 2013; Ling et al., 2013).

Compacting models has been a focus of research in tasks, such as language modeling and machine translation, as extremely large models can be built with the large amounts of training data that are available in these tasks. In language modeling, it is frequent to prune higher order n-grams that do not encode any additional information (Seymore and Rosenfeld, 1996; Stolcke, 1998; Moore and Quirk, 2009). The same be applied in machine translation (Ling et al., 2012; Zens et al., 2012) by removing longer translation pairs that can be replicated using smaller ones. In essence our model learns regularities at the subword level that can be leveraged for building more compact word representations.

Finally, our work has been applied to dependency parsing and found similar improvements over word models in morphologically rich languages (Ballesteros et al., 2015).

7 Conclusion

We propose a C2W model that builds word embeddings for words without an explicit word lookup table. Thus, it benefits from being sensitive to lexical aspects within words, as it takes characters as atomic units to derive the embeddings for the word. On POS tagging, our models using characters alone can still achieve comparable or better results than state-of-the-art systems, without the need to manually engineer such lexical features. Although both language modeling and POS tagging both benefit strongly from morphological cues, the success of our models in languages with impoverished morphological cues shows that it is able to learn non-compositional aspects of how letters fit together.

The code for the C2W model and our language model and POS tagger implementations is available from `https://github.com/wlin12/JNN`.

Acknowledgements

The PhD thesis of Wang Ling is supported by FCT grant SFRH/BD/51157/2010. This research was supported in part by the U.S. Army Research Laboratory, the U.S. Army Research Office under contract/grant number W911NF-10-1-0533 and NSF IIS-1054319 and FCT through the pluri-annual contract UID/CEC/50021/2013 and grant number SFRH/BPD/68428/2010.

References

Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. 2002. "Floresta sintá(c)tica": a treebank for Portuguese. In *Proc. LREC*.

Nart B. Atalay, Kemal Oflazer, and Bilge Say. 2003. The annotation process in the Turkish treebank. In *In Proc. the 4th International Workshop on Linguistically Interpreted Corpora (LINC)*.

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

Jan A. Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *Proc. ICML*.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proc. EMNLP*.

Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. 2015. Joint learning of character and word embeddings.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proc. EMNLP*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *JMLR*.

Ferdinand de Saussure. 1916. *Course in General Linguistics*.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8).

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proc. EMNLP*.

Byung-Ju Kang and Key-Sun Choi. 2000. Automatic transliteration and back-transliteration by decision tree learning. In *LREC*.

Jiwei Li, Dan Jurafsky, and Eduard H. Hovy. 2015. When are tree structures necessary for deep learning of representations? *CoRR*, abs/1503.00185.

Wang Ling, João Graça, Isabel Trancoso, and Alan Black. 2012. Entropy-based pruning for phrase-based machine translation. In *Proc. EMNLP*.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. 2013. Paraphrasing 4 microblog normalization. In *Proc. EMNLP*.

Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proc. NAACL*.

Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. 2014. A recursive recurrent neural network for statistical machine translation. In *Proc. ACL*.

Thang Luong, Richard Socher, and Christopher Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proc. CoNLL*.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proc. CICLing*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Comput. Linguist.*

William Marslen-Wilson and Lorraine K. Tyler. 1998. Rules, representations, and the English past tense. *Trends in Cognitive Science*, 2(11).

M. Antonia Martí, Mariona Taulé, Lluís Márquez, and Manuel Bertran. 2007. CESS-ECE: A multilingual and multilevel annotated corpus. In *Proc. LREC*.

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

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

Robert C. Moore and Chris Quirk. 2009. Less is more: significance-based n-gram selection for smaller, better language models. In *Proc. EMNLP*.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proc. EMNLP*.

Tetsuji Nakagawa, Taku Kudoh, and Yuji Matsumoto. 2001. Unknown word guessing and part-of-speech tagging using support vector machines. In *In Proc. the Sixth Natural Language Processing Pacific Rim Symposium*.

Cicero D. Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proc. ICML*.

Kristie Seymore and Ronald Rosenfeld. 1996. Scalable backoff language models. In *Proc. ICSLP*.

Anders Søgaard. 2011. Semisupervised condensed nearest neighbor for part-of-speech tagging. In *Proc. ACL*.

Radu Soricut and Franz Och. 2015. Unsupervised morphology induction using word embeddings. In *Proc. NAACL*.

Drahomíra Spoustová, Jan Hajič, Jan Raab, and Miroslav Spousta. 2009. Semi-supervised training for the averaged perceptron POS tagger. In *Proc. EACL*.

Andreas Stolcke. 1998. Entropy-based pruning of backoff language models. In *In Proc. DARPA Broadcast News Transcription and Understanding Workshop*.

Xu Sun. 2014. Structure regularization for structured prediction: Theories and experiments. *CoRR*, abs/1411.6243.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Proc. Interspeech*.

Wei Xu, Alan Ritter, and Ralph Grishman. 2013. Gathering and generating paraphrases from twitter with application to normalization. In *Proceedings of the Sixth Workshop on Building and Using Comparable Corpora*.

Richard Zens, Daisy Stanton, and Peng Xu. 2012. A systematic comparison of phrase table pruning techniques. In *Proc. EMNLP*.