Improved Word and Symbol Embedding for Part-of-Speech Tagging

Nicholas Altieri, Sherdil Niyaz, Samee Ibraheem, and **John DeNero**
{naltieri,sniyaz,sibraheem,denero}@berkeley.edu

Abstract

State-of-the-art neural part-of-speech (POS) taggers trained only on labeled data from the Penn Treebank have comparable performance to a structure perceptron tagger with hand-engineered features. This paper explores three modeling techniques for a neural POS tagger that address potential learning challenges at the boundaries of the tagger's discrete and continuous representations of data. First, a model that predicts each tag independently, based on a joint representation of the input sequence, only has access to the model's hidden continuous representation of the sentence, but not discrete distributions over neighboring tags. We show that also conditioning on a learned multinomial distribution over the discrete tag space for neighboring positions improves performance. Second, using only one embedding vector for each input symbol leads to high variability in final tag accuracy, perhaps due to the challenge of jointly optimizing the embeddings for so many symbols. We show that embedding each symbol twice, in combination with dropout on the embedding layers, also improves performance. Finally, the choice of how each word is decomposed into sub-words affects the way in which continuous parameters are allocated to a discrete sequence of symbols. We describe a new data-driven technique for sub-word segmentation designed to respect morpheme boundaries. However, our experiments indicate that this change does not improve final tag accuracy, despite a large increase in intrinsic segmentation quality when compared with human segmentation annotations. Overall, these three approaches highlight the learning challenges that arise when a model embeds discrete symbols into continuous spaces. Combining these techniques reduces test set errors by 3.8%.

Introduction

Deep learning has achieved state-of-the-art performance in many natural language processing (NLP) tasks by embedding words and tags into continuous vector spaces. An important technique for generalizing to unseen vocabularies is to represent each word as a variable-length sequence of symbols from a small fixed inventory of *sub-words*. This paper explores the performance of a convolutional deep part-of-speech (POS) tagger trained on the Penn Treebank, in order to identify and address challenges that arise when sub-words and tags are embedded as vectors.

A bidirectional LSTM (Hochreiter and Schmidhuber, 1997) represents the current state-of-the-art approach to POS tagging (Ling et al., 2015; Wang et al., 2015). However, in order to achieve maximal performance, unlabeled training data from a large external text corpus is required. Using only labeled data from the Penn Treebank, the best reported accuracy of a deep tagger is 97.36%, using a bidirectional LSTM over characters to encode words, and another bidirectional LSTM over words to encode sentence context for tagging (Ling et al., 2015). The best reported accuracy of a linear tagger trained with structured perceptron to combine hand-engineered features, trained only on labeled data from the Penn Treebank, is a nearly identical 97.35% (Huang et al., 2012). The fact that a linear model using hand-engineered features can match the performance of a nonlinear model capable of feature induction indicates that there may be some remaining learning challenges when embedding discrete symbols into continuous spaces.

Rather than attempt to establish a new state-of-the-art through known methods such as deeper models or ensembles, we aim to provide insight into whether certain targeted improvements provide reliable performance gains in POS tagging, with the hope of informing research on other language tasks. Toward this end, we explore three candidate extensions to a convolutional tagger designed to address potential learning challenges at the boundaries of the model's discrete and continuous representations of text and tags.

First, a model that predicts each tag independently using a softmax output layer may not effectively learn tag-sequence patterns. The output layer of a deep POS tagger conditions on an internal representation of the word and its context, but does not condition on the predicted tags of neighboring words. By contrast, state-of-the-art structured perceptron taggers benefit from tag-sequence features. In principle, a deep model's hidden representation should be sufficiently expressive to represent these features, but it may be challenging to learn these patterns in practice. To investigate this issue, we evaluate a model architecture that applies the same output layer twice: once to generate preliminary probabilities for each tag in each position, then again to generate final probabilities conditioned on those preliminary probabilities. In this way, the model has explicit access to its own predicted distributions over discrete tags, rather than only its embedded representations. This architecture change provides a small improvement, eliminating 1.47% of test set errors on average.

The second challenge is that jointly learning embeddings for a large inventory of sub-words may be a difficult numerical optimization problem. In an attempt to improve training, we evaluate the following small architecture change. Each input symbol, a sub-word in our experiments, is associated with two different embedding vectors instead of one. During training, dropout is applied to each dimension of each vector independently, then the vectors are summed. At test time, the vectors are summed without dropout, so the model is equivalent at test time to having only one embedding for each sub-word. This simple change reduces performance variability and provides a small improvement, eliminating 0.8% of test set errors on average. Combining these two techniques reduces errors by 3.8% on average.

The final challenge concerns how words are encoded as sub-word symbols before being embedded in a vector space. We investigate how a variable-length encoding technique called byte-pair encoding, used to segment words into sub-words, may impact the model's ability to identify morphological patterns. For example, if *earning* and *gaining* are segmented as *earn- ing* and *gain- ing* respectively, then both word representations will share parameters since they share a sub-word. However, if *earning* is instead segmented as *ear- ning* (a common result from byte-pair encoding), then these words won't share an embedding, and the relationship between the two words must be learned. We evaluate a data-driven variant of byte-pair encoding that is designed to impose sub-word segmentation boundaries between morphemes. While we show a clear increase in intrinsic segmentation quality, final tag accuracy does not appear to improve. Overall, our experiments do not conclusively associate increased segmentation quality with better tagger performance.

This investigation of techniques related to embedding for deep POS taggers produces a competitive model with labeled-training-only test set accuracy of 96.68% on the Penn Treebank, similar to the 96.61% accuracy of the best bidirectional LSTM over words and case markers reported in (Wang et al., 2015) and the 96.70% accuracy of the best bidirectional LSTM over words reported in (Ling et al., 2015).

Related Work

State-of-the-Art POS Taggers

The bidirectional LSTM tagger of (Wang et al., 2015) embeds two discrete input sequences: lower-cased words and indicator features describing the original capitalization pattern of each word. The authors also describe using a sequence of two-letter suffixes as an additional input. The model benefits substantially from large hidden layers and from word embeddings trained on a large unlabeled text corpus of 536 million words. The feed-forward network of (Collobert et al., 2011) used similar input and also benefited from unlabeled text data. The highest performing model trained only on labeled data is reported in (Ling et al., 2015). This work uses a bidirectional LSTM over characters to form word representations, and then another bidirectional LSTM over the words to embed sentence context and predict tag. One drawback of this method is that embedding characters rather than words causes the model to be substantially slower than a word-based alternative, especially during training.

The best-performing linear POS taggers trained only on labeled data combine hand-engineered features over tag sequences and morphological patterns. The structured perceptron tagger of (Huang et al., 2012) uses the features originally developed for the maximum entropy tagger of (Ratnaparkhi, 1996). These features include tag bigrams and tag trigrams. Tagging by independent classifiers can achieve comparable performance, but only when trained on large unlabeled corpora (Moore, 2014). Syntactic parsers also provide excellent part-of-speech tagging performance, but are typically evaluated on a different test set than POS taggers.

Word Representation Methods

A substantial amount of recent research has investigated different methods of encoding text as a sequence of discrete symbols. One simplistic approach is to embed each common word as its own vector and all uncommon words using a single UNK vector. A promising alternative is to represent each word as a variable-length sequence of symbols from a fixed-size inventory, for example using language-model-based sub-word segmentation (Schuster and Nakajima, 2012), Huffman codes (Chitnis and DeNero, 2015), or byte-pair encoding (Sennrich et al., 2015). The byte-pair encoding (BPE) method that splits words into frequent sub-words has proven particularly effective for machine translation. This approach uses only character n-gram frequencies in order to segment words.

In addition to BPE, mixed word-character models have also yielded performance gains over treating words as single tokens. For example, one effective approach is to represent common words as individual tokens and rare or unknown words as a sequence of characters (Luong and Manning, 2016). An approach that has proven particularly effective for language modeling and part-of-speech tagging is to treat all words as a sequences of characters (Ling et al., 2015).

Model

Our model has two components: the first generates a representation for each word from its embedded sub-word symbols, and the second predicts a tag from the representations of the word to be tagged and its context words. The word and its context are combined using a convolution, rather an an LSTM, because the faster training speed of this architecture allows us to perform a more thorough experimental evaluation.

Word Representation

Generating vector embeddings for words in the corpus is straightforward when each word is represented as a single symbol, as multiple approaches exist for doing so (Pennington et al., 2014). However, we focus in this work on representing each individual word as a sequence of sub-words, for example chosen by byte-pair encoding (Sennrich et al., 2015).

In order to generate the vector embedding for a word with sub-words $s_1, ..., s_n$, we first embed each sub-word s_j as a vector e_j, then combine $e_1, ..., e_n$ with an LSTM. The final hidden state of the forward pass of the LSTM is a vector v representing the word. This architecture allows the model to embed words with arbitrarily many sub-words.

Convolutional Model

Once a vector embedding v_i has been generated for each word in a sentence, we apply a convolution operation to generate convolution vector c_i that combines $v_{i-k}, \ldots, v_i, \ldots, v_{i+k}$ for the purpose of predicting the tag for word i. This convolution operation involves the vector embedding for w_i, as well as the vector embeddings for all words within the convolution window around w_i. We denote this operation as $C_w(w_i)$.

After computing the convolution vectors, we sum the vector embedding for each word with its corresponding convolution vector, $s_i = v_i + c_i$. Finally, we apply a dense layer to the resulting vectors with a softmax activation to produce a multinomial distribution over possible tags for each word, $t_i = D(s_i)$.

Extensions

We consider three extensions to this model, each designed to target a potential learning challenge.

Conditioning on Neighbor Tag Predictions

When predicting a part-of-speech tag for a word, we incorporate predictions of neighboring tags as inputs into the final output layer. That is, given the vector embedding v_i of a word w_i and its convolutional output c_i from the convolutional network over its context, we first apply a dense layer D to $s_i = v_i + c_i$ in order to get preliminary estimates of the tag distribution t'_i for each w_i. Then, in order to refine our estimates, we apply a convolutional layer C_t over the tag predictions t'_i's to get an output r_i that serves as a residual adjustment to s_i. Finally, we apply D again, with the same parameters as before, to $s_i + r_i$ in order to generate final tag distributions.

In summary, we incorporate neighboring tag information as follows:

$c \leftarrow C_w(v)$	# Convolve over words
$s \leftarrow v + c$	# Sum the embeddings
$t' \leftarrow D(s)$	# Initial tag predictions
$r \leftarrow C_t(t')$	# Convolve over tags
$t \leftarrow D(s + r)$	# Final tag predictions

This architecture adds new parameters C_t to perform a convolution over the intermediate multinomial tag distributions t'. We describe t' as a multinomial over tags, not only because of the softmax applied by D, but also because the parameters of layer D are shared, both to predict t' and t. The model is trained to be a distribution over tags using a cross-entropy loss over the final output t predicted by D, and so we may expect that the intermediate output t' also predicted by D will give a similar distribution over tags.

Duplicate Embedding with Dropout

We embed each sub-word using two separate vectors that are randomly initialized independently: $e_j^{(1)}$ and $e_j^{(2)}$. The embedding vector used as input to the LSTM over sub-words is the sum of these duplicated embeddings.

$$e_j = e_j^{(1)} + e_j^{(2)}$$

This modification only differs from a single embedding vector per sub-word because dropout is applied independently to each embedding vector during training. We use a dropout probability of 0.2 in experiments. Therefore, each dimension of e_j is only dropped out completely with probability 0.04. More often, a dimension of e_j during training is computed from only one of the addends because the other is dropped out, an event that occurs with probability 0.32. The most common case is that the dimension of e_j is the sum of the addends, an event with probability 0.64. The resulting embedding e_j is affected by dropout whenever $e_j^{(1)}$ and $e_j^{(2)}$ differ, but remains mostly unaffected by dropout if the contents of both embedding vectors converge to the same values.

Morphological Pre-Segmentation

Our final extension involves segmenting words in the corpus in a manner that is designed to respect morpheme boundaries, as opposed to byte-pair encoding (BPE), which is computed only from character n-gram frequencies. Described most generally, our approach enforces segmentation boundaries in the corpus before BPE is applied, and then applies BPE to these pre-segmented words. Therefore, the result is an encoding of the corpus with a fixed number of symbols chosen in advance. Therefore, the final model has the same number of parameters using our morphological pre-segmentation technique as it would when applied to a corpus segmented directly with BPE.

Our pre-segmentation algorithm uses the output of another data-driven corpus analysis technique, unsupervised induction of morphological transformations using word embeddings (Soricut and Och, 2015). This technique identifies triples $(w_{old},\ w_{new},\ type)$, where $type \in \{prefix, suffix\}$. The words w_{old} and w_{new} are related words that differ only in their prefix or suffix.

We only use transformations in which w_{old} is a word longer than w_{new}. We also assume the existence of trained vector embeddings for full words in the corpus. For each input rule, we define an example transformation vector $\vec{v} = \vec{v}_{new} - \vec{v}_{old}$, where $\vec{v}_{new}$ and $\vec{v}_{old}$ are the trained vector embeddings for w_{new} and w_{old} respectively.

We also compute s_f and s_t for each such example transformation. If a certain transformation has a $type$ of "prefix", s_f is the smallest substring of w_{old} that, when removed from the front of w_{old}, makes the remaining string a substring of w_{new}. This is symmetric for the suffix case. The first step of our algorithm iterates through all example transformations, computing and storing the tuple $(s_f,\ s_t,\ \vec{v},\ type)$ for each.

Once all such 4-tuples have been computed from the trained embeddings and the input transformations, pre-segmentations are computed on all words in the corpus: for each word w_i, we test against all 4-tuples, with optimizations and heuristics to ensure tractability. Consider w_i being tested against $(s_f,\ s_t,\ \vec{v},\ type)$. If $type$ is of "prefix", s_f is replaced from the beginning of w_i (if possible) with s_t to create a new word w_c. If w_c is in the corpus, we define a candidate transformation vector:

$$\vec{v}_t = \vec{v}_c - \vec{v}_i$$

where $\vec{v}_i$ and $\vec{v}_c$ are the vector embeddings for w_i and w_c respectively. If $\vec{v}_t \cdot \vec{v}$ exceeds a threshold γ, we create a pre-segmentation $p_i = [s_f, w_i - s_f]$ for w_i, where $w_i - s_f$ denotes the remaining string when s_f is removed from the front or end of w_i, depending on the type of transformation. This use of direction vectors as measures of transform similarity has previously been shown to be effective (Soricut and Och, 2015).

We now have a list of pre-segmentations $p_1, p_2, ...p_n$. Each p_i, is interpreted as a "hard boundary" in the word w_i it applies to. We apply BPE to the corpus, but impose an additional constraint that these hard boundaries should always exist between subwords, regardless of how other parts of w_i are merged through the creation of new symbols.

As a final optimization, we allow ignoring these pre-segmentations in the k most frequent words. These words may be important enough to warrant vector representations on their own, instead of being broken into sub-words.

Results

We evaluate these extensions by performing repeated experiments on the Penn Treebank 3. We used the standard section split from prior work established in (Collins, 2002): Sections 0-18 for training, 19-21 for validation, and 22-24 for testing. All results appear in Table 1.

For each condition, we trained the tagger parameters from 8 different random initializations. We report average test-set accuracy for each condition over these 8 training runs. Validation set accuracy was measured every epoch, and training was stopped early in each run whenever validation set accuracy decreased. In all experiments, words were segmented into 8192 sub-words. Each sub-word was embedded into a 64-dimensional vector space, and all hidden layers also had dimension 64. The Adam optimizer was used to minimize cross-entropy loss.

Conditioning on Neighboring Tag Predictions

The extension of explicitly conditioning on neighboring tag distributions (*Tag Twice*) resulted in a modest gain over the baseline architecture. The improvement from the baseline condition (row 1) to the tag-twice condition (row 2) with baseline sub-word embeddings and segmentation was not statistically significant according to a one-tailed permutation test to evaluate whether test set accuracy was reliably higher under the *Tag Twice* condition (p=0.09). However, the *Tag Twice* technique did provide a statistically significant improvement on top of duplicate embedding (row 4 vs row 3; p=0.02), as well as when applied to the data set that was pre-segmented (row 6 vs row 5; p=0.05). With only 8 samples for each condition, statistical significance may be difficult to establish, even if the distribution of outcomes for two conditions are in fact different. Therefore, with two of three comparisons showing significance, it is reasonable to conclude that conditioning on the discrete multinomial distribution of neighbor tag predictions does improve part-of-speech tagging.

Tagging	Embedding	Segmentation	Accuracy
Baseline	Baseline	BPE	96.56%
Tag Twice	Baseline	BPE	96.61%
Baseline	Embed Twice	BPE	96.59%
Tag Twice	Embed Twice	BPE	**96.68%**
Baseline	Baseline	Morphology	96.54%
Tag Twice	Baseline	Morphology	96.61%
Baseline	Embed Twice	Morphology	96.57%
Tag Twice	Embed Twice	Morphology	96.61%

Table 1: Average test-set accuracy over 8 training runs for each condition.

Duplicate Embedding with Dropout

The extension of embedding each input symbol twice and applying dropout independently to both embedding vectors (*Embed Twice*) also resulted in modest but consistent gains in accuracy. The improvement over the baseline condition was not statistically significant (row 3 vs row 1; p=0.26), but the improvement when tagging twice in both conditions was statistically significant (row 4 vs row 2; p=0.03). The standard deviation of accuracies within a condition was 12% smaller for the *Embed Twice* condition than the baseline (row 3 vs row 1), indicating that this extension might reduce variability in the outcome accuracy.

Morphological Pre-Segmentation

As part of testing our morphological pre-segmentations, we ran an intrinsic segmentation quality experiment. The gold standard segmentations were taken from the Morpho Challenge 2005 and Morpho Challenge 2010 (Kurimo et al, 2010).

We ran our pre-segmenter with $\gamma = 0.6$ in all experiments. We chose to exclude the top $k = 1000$ words from pre-segmentation. Vector embeddings were trained for the corpus using the GloVe algorithm (Pennington et al., 2014). The example transformations used as input were trained on a separate corpus using the approach of Soricut and Och (2015). In all sub-word experiments, 8000 merges were used for BPE after pre-segmentations were imposed.

When compared against BPE, our morphological approach to sub-words displayed a substantial improvement in F-measure against a gold standard segmentation favoring morphological splits (Figure 1).

However, morphological pre-segmentation did not result in better POS performance, despite the increase in intrinsic segmentation quality. Tagging after morphological pre-

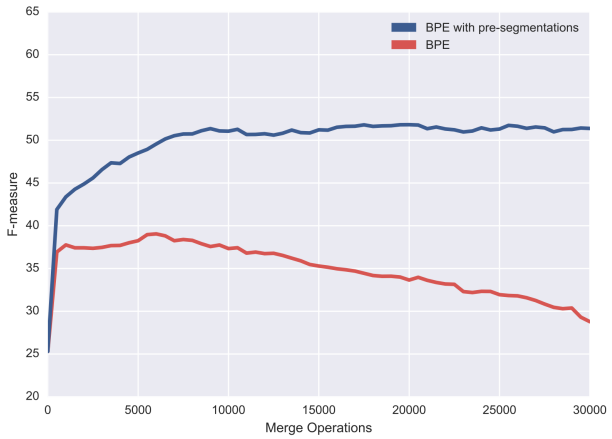

Figure 1: F-measure of BPE method with and without morphological pre-segmentations.

segmentation was slightly but consistently less accurate than tagging with the segmentations from byte-pair-encoding without any pre-segmentation.

Combined Improvements

The improvement that resulted from the combination of *Tag Twice* and *Embed Twice* for BPE segmentations was highly statistically significantly (row 4 vs row 1; p¡0.01). Neither extension alone provided such a clear advantage over the baseline, indicating that both extensions contributed to the performance gain.

We found that these methods improved general POS tagging performance, decreasing the number of errors for 33 out of 42 part-of-speech tags for which errors were observed. Table 2 provides some examples for which mistakes made by the baseline model were generally fixed by these two modifications.

Conclusion and Discussion

Our convolutional POS tagging model that predicts tags from sub-words is a typical example of a natural language processing model with discrete inputs and outputs. By addressing issues related to embedding these discrete symbols into vector spaces, we identified sources of modest improvement that reduce test set errors by 3.8%. This allowed us to provide a competitive model with average accuracy improved from 96.56% to 96.68%.

Notably, two approaches that operated on the model level fared well in our experiments compared to an approach that instead modified the input to the model: in this case, by using superior word segmentations. It is indeed surprising that our experiments did not associate intrinsic segmentation accuracy with increased tagger performance.

We hope that these combined results provide some guidance for future work in extending and refining NLP models that mix discrete and continuous representations of text and symbols.

References

Chitnis and DeNero, 2015. Variable-Length Word Encodings for Neural Translation Models. In *EMNLP 2015.*

Hochreiter and Schmidhuber, 1997. Long Short-Term Memory In *ACM 1997.*

Huang et al, 2012. Structured Perceptron with Inexact Search. In *ACL 2012.*

Kurimo et al, 2010. Morpho Challenge competition 2005–2010: evaluations and results. In *ACM 2010.*

Ling et al., 2015. Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. In *EMNLP 2015.*

Luong and Manning, 2016. Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models. In *ACL 2016.*

Moore, 2015. An Improved Tag Dictionary for Faster Part-of-Speech Tagging. In *EMNLP 2015.*

Pennington et al., 2014. GloVe: Global Vectors for Word Representation. In *ACL 2014.*

Schuster and Nakajima, 2012. Japanese and Korean Voice Search. In *IEEE 2012.*

Sennrich et al., 2015. Neural Machine Translation of Rare Words with Subword Units. In *ACL 2016.*

Soricut and Och, 2015. Unsupervised Morphology Induction Using Word Embeddings. In *ACL 2015.*

Toutanov et al., 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *ACL 2003.*

Wang et al., 2015. Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network. In *Arxiv.*

Wu et al., 2016. Googles Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. In *Arxiv.*

Sentence	Word	Baseline Correct Tags	Tag Twice + Embed Twice Correct Tags
The division had only minor damage at its Sunnyvale headquarters and plant in Palo Altos, and no delays in deliveries are expected.	deliveries	0	5
The firm brought in to strengthen the structure could be liable as well.	liable	1	6
Los Angeles County Supervisor Kenneth Hahn yesterday vowed to fight the introduction of double-decking in the area.	Kenneth	5	8

Table 2: Examples where the model using tag twice and dropout makes fewer errors than the baseline model. Number of correct classifications out of 8 runs is shown. In the first case, the plural noun (NNS) was misclassified as an adjective (JJ). For the second sentence, the adjective was misclassified as a preposition or subordinating conjunction (IN). In the third example, a singular proper noun (NNP) was misclassified as either an adjective or a verb, gerund, or present participle.