Mapping Unseen Words to Task-Trained Embedding Spaces

Pranava Swaroop Madhyastha[*] **Mohit Bansal**[†] **Kevin Gimpel**[†] **Karen Livescu**[†]
[*]Universitat Politècnica de Catalunya
pranava@cs.upc.edu
[†]Toyota Technological Institute at Chicago
{mbansal,kgimpel,klivescu}@ttic.edu

Abstract

We consider the supervised training setting in which we learn task-specific word embeddings. We assume that we start with initial embeddings learned from unlabelled data and update them to learn task-specific embeddings for words in the supervised training data. However, for new words in the test set, we must use either their initial embeddings or a single unknown embedding, which often leads to errors. We address this by learning a neural network to map from initial embeddings to the task-specific embedding space, via a multi-loss objective function. The technique is general, but here we demonstrate its use for improved dependency parsing (especially for sentences with out-of-vocabulary words), as well as for downstream improvements on sentiment analysis.

1 Introduction

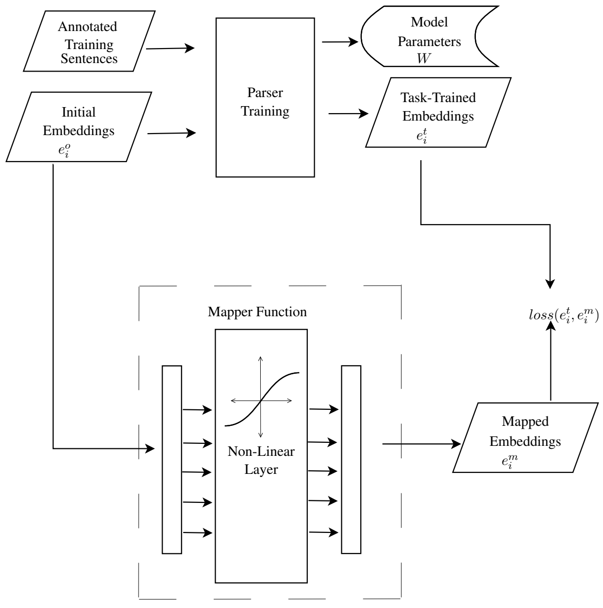
Performance on NLP tasks drops significantly when moving from training sets to held-out data (Petrov et al., 2010). One cause of this drop is words that do not appear in the training data but appear in test data, whether in the same domain or in a new domain. We refer to such out-of-training-vocabulary (OOTV) words as *unseen* words. NLP systems often make errors on unseen words and, in structured tasks like dependency parsing, this can trigger a cascade of errors in the sentence.

Word embeddings can counter the effects of limited training data (Necsulescu et al., 2015; Turian et al., 2010; Collobert et al., 2011). While the effectiveness of pretrained embeddings can be heavily task-dependent (Bansal et al., 2014), there is a great deal of work on updating embeddings during supervised training to make them more task-specific (Kalchbrenner et al., 2014; Qu et al., 2015; Chen and Manning, 2014). These task-trained embeddings have shown encouraging results but raise some concerns: (1) the updated embeddings of infrequent words are prone to overfitting, and (2) many words in the test data are not contained in the training data at all. In the latter case, at test time, systems either use a single, generic embedding for all unseen words or use their initial embeddings (typically derived from unlabelled data) (Søgaard and Johannsen, 2012; Collobert et al., 2011). Neither choice is ideal: A single unknown embedding conflates many words, while the initial embeddings may be in a space that is not comparable to the trained embedding space.

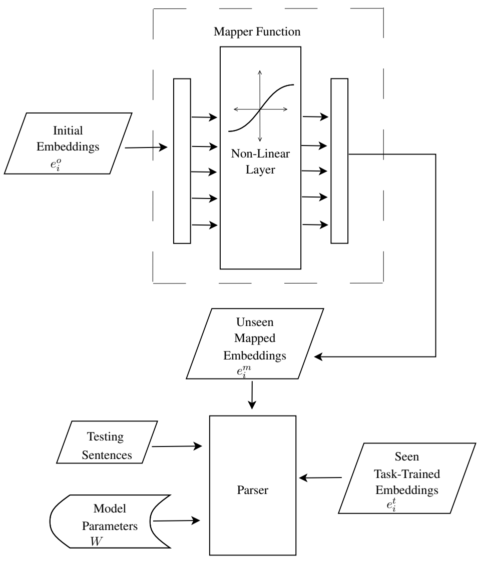
In this paper, we address both concerns by learning to map from the initial embedding space to the task-trained space. We train a neural network mapping function that takes initial word embeddings and maps them to task-specific embeddings that are trained for the given task, via a multi-loss objective function. We tune the mapper's hyperparameters to optimize performance on each domain of interest, thereby achieving some of the benefits of domain adaptation. We demonstrate significant improvements in dependency parsing across several domains and for the downstream task of dependency-based sentiment analysis using the model of Tai et al. (2015).

2 Mapping Unseen Representations

Let $\mathcal{V} = \{w_1, \ldots, w_V\}$ be the vocabulary of word types in a large, unannotated corpus. Let e_i^o denote the initial (original) embedding of word w_i computed from this corpus. The initial embeddings are typically learned in an unsupervised


Annotated Training Sentences
Initial Embeddings e_i^o
Parser Training
Model Parameters W
Task-Trained Embeddings e_i^t
Mapper Function
Non-Linear Layer
$loss(e_i^t, e_i^m)$
Mapped Embeddings e_i^m

(a) Mapper Training


Mapper Function
Initial Embeddings e_i^o
Non-Linear Layer
Unseen Mapped Embeddings e_i^m
Testing Sentences
Model Parameters W
Parser
Seen Task-Trained Embeddings e_i^t

(b) Test-time: Parsing with Mapped Embeddings

Figure 1: System Pipeline

way, but for our purposes they can be any initial embeddings. Let $\mathcal{T} \subseteq \mathcal{V}$ be the subset of words that appear in the annotated training data for some supervised task-specific training. We define **unseen** words as those in the set $\mathcal{V} \setminus \mathcal{T}$. While our approach is general, for concreteness, we consider the task of dependency parsing, so the annotated data consists of sentences paired with dependency trees. We assume a dependency parser that learns task-specific word embeddings e_i^t for word $w_i \in \mathcal{T}$, starting from the original embedding e_i^o. In this work, we use the Stanford neural dependency parser (Chen and Manning, 2014).

The goal of the mapper is as follows. We are given a training set of N pairs of initial and task-trained embeddings $\mathcal{D} = \{(e_1^o, e_1^t), \ldots, (e_N^o, e_N^t)\}$, and we want to learn a function G that maps each initial embedding e_i^o to be as close as possible to its corresponding output embedding e_i^t. We denote the mapped embedding e_i^m, i.e., $e_i^m = G(e_i^o)$.

Figure 1a describes the training procedure of the mapper. We use a supervised parser which is trained on an annotated dataset and initialized with pre-trained word embeddings e_i^o. The parser uses back-propagation to update these embeddings during training, producing task-trained embeddings e_i^t for all $w_i \in \mathcal{T}$. After we train the parser, the mapping function G is trained to map an initial word embedding e_i^o to its parser-trained embedding e_i^t. At test (or development) time, we use the trained mapper G to transform the original embeddings of unseen test words to the parser-trained space (see Figure 1b). When parsing held-out data, we use the same parser model parameters (W) as shown in Figure 1b. The only difference is that now some of the word embeddings (i.e., for unseen words) have changed to mapped ones.

2.1 Mapper Architecture

Our proposed mapper is a multi-layer feedforward neural network that takes an initial word embedding as input and outputs a mapped representation of the same dimensionality. In particular, we use a single hidden layer with a hard$\tanh$ nonlinearity, so the function G is defined as:

$$G(e_i^o) = W_2(\text{hard}\tanh(W_1 e_i^o + b_1)) + b_2 \quad (1)$$

where W_1 and W_2 are parameter matrices and b_1 and b_2 are bias vectors.

The 'hard$\tanh$' non-linearity is the standard 'hard' version of hyperbolic tangent:

$$\text{hard}\tanh(z) = \begin{cases} -1 & \text{if } z < -1 \\ z & \text{if } -1 \le z \le 1 \\ 1 & \text{if } z > 1 \end{cases}$$

In preliminary experiments we compared with other non-linear functions (sigmoid, $\tanh$, and ReLU), as well as with zero and more than one non-linear layers. We found that fewer or more non-linear layers did not improve performance.

2.2 Loss Function

We use a weighted, multi-loss regression approach, optimizing a weighted sum of mean squared error and mean absolute error:

$$loss(y, \hat{y}) = \alpha \sum_{j=1}^{n} |y_j - \hat{y_j}| + (1 - \alpha) \sum_{j=1}^{n} |y_j - \hat{y_j}|^2 \quad (2)$$

where $y = e_i^t$ (the ground truth) and $\hat{y} = e_i^m$ (the prediction) are n-dimensional vectors. This multi-loss approach seeks to make both the conditional *mean* of the predicted representation close to the task-trained representation (via the squared loss) and the conditional *median* of the predicted representation close to the task-trained one (via the mean absolute loss). A weighted multi-criterion objective allows us to avoid making strong assumptions about the optimal transformation to be learned. We tune the hyperparameter α on domain-specific held-out data. We try to minimize the assumptions in our formulation of the loss, and let the tuning determine the particular mapper configuration that works best for each domain. Strict squared loss or an absolute loss are just special forms of this loss function.

For optimization, we use batch limited memory BFGS (L-BFGS) (Liu and Nocedal, 1989). In preliminary experiments comparing with stochastic optimization, we found L-BFGS to be more stable to train and easier to check for convergence (as has recently been found in other settings as well (Ngiam et al., 2011)).

2.3 Regularization

We use elastic net regularization (Liu and Nocedal, 1989), which linearly combines ℓ_1 and ℓ_2 penalties on the parameters to control the capacity of the mapper function. This equates to minimizing:

$$F(\theta) = L(\theta) + \lambda_1 \|\theta\|_1 + \frac{\lambda_2}{2} \|\theta\|_2^2$$

where θ is the full set of mapper parameters and $L(\theta)$ is the loss function (Eq. 2 summed over mapper training examples). We tune the hyperparameters of the regularizer and the loss function separately for each task, using a task-specific development set. This gives us additional flexibility to map the embeddings for the domain of interest, especially when the parser training data comes from a particular domain (e.g., newswire) and we want to use the parser on a new domain (e.g., email). We also tried dropout-based regularization (Srivastava et al., 2014) for the non-linear layer but did not see any significant improvement.

2.4 Mapper-Parser Thresholds

Certain words in the parser training data $\mathcal{T}$ are very infrequent, which may lead to inferior task-specific embeddings e_i^t learned by the parser. We want our mapper function to be learned on high-quality task-trained embeddings. After learning a strong mapping function, we can use it to remap the inferior task-trained embeddings.

We thus consider several frequency thresholds that control which word embeddings to use to train the mapper and which to map at test time. Below are the specific thresholds that we consider:

Mapper-training Threshold (τ_t) The mapper is trained only on embedding pairs for words seen at least τ_t times in the training data $\mathcal{T}$.

Mapping Threshold (τ_m) For test-time inference, the mapper will map any word whose count in $\mathcal{T}$ is less than τ_m. That is, we discard parser-trained embeddings e_i^t of these infrequent words and use our mapper to map the initial embeddings e_i^o instead.

Parser Threshold (τ_p) While training the parser, for words that appear fewer than τ_p times in $\mathcal{T}$, the parser replaces them with the 'unknown' embedding. Thus, no parser-trained embeddings will be learned for these words.

In our experiments, we explore a small set of values from this large space of possible threshold combinations (detailed below). We consider only relatively small values for the mapper-training (τ_t) and parser thresholds (τ_p) because as we increase them, the number of training examples for the mapper decreases, making it harder to learn an accurate mapping function[1].

3 Related Work

There are several categories of related approaches, including those that learn a single

[1]Note that the training of the mapper tends to be very quick because training examples are word types rather than word tokens. When we increase τ_t, the number of training examples reduces further. Hence, since we do not have many examples, we want the mapping procedure to have as much flexibility as possible, so we use multiple losses and regularization strategies, and then tune their relative strengths.

embedding for unseen words (Søgaard and Johannsen, 2012; Chen and Manning, 2014; Collobert et al., 2011), those that use character-level information (Luong et al., 2013; Botha and Blunsom, 2014; Ling et al., 2015; Ballesteros et al., 2015), those using morphological and n-gram information (Candito and Crabbé, 2009; Habash, 2009; Marton et al., 2010; Seddah et al., 2010; Attia et al., 2010; Bansal and Klein, 2011; Keller and Lapata, 2003), and hybrid approaches (Dyer et al., 2015; Jean et al., 2015; Luong et al., 2015; Chitnis and DeNero, 2015). The representation for the unknown token is either learned specifically or computed from a selection of rare words, for example by averaging their embedding vectors.

Other work has also found improvements by combining pre-trained, fixed embeddings with task-trained embeddings (Kim, 2014; Paulus et al., 2014). Also relevant are approaches developed specifically to handle large target vocabularies (including many rare words) in neural machine translation systems (Jean et al., 2015; Luong et al., 2015; Chitnis and DeNero, 2015).

Closely related to our approach is that of Tafforeau et al. (2015). They induce embeddings for unseen words by combining the embeddings of the k nearest neighbors. In Sec. 4, we show that our approach outperforms theirs. Also related is the approach taken by Kiros et al. (2015). They learn a linear mapping of the initial embedding space via unregularized linear regression. Our approach differs by considering nonlinear mapping functions, comparing different losses and mapping thresholds, and learning separately tuned mappers for each domain of interest. Moreover, we focus on empirically evaluating the effect of the mapping of unseen words, showing statistically significant improvements on both parsing and a downstream task (sentiment analysis).

4 Experimental Setup

4.1 Dependency Parser

We use the feed-forward neural network dependency parser of Chen and Manning (2014). In all our experiments (unless stated otherwise), we use the default arc-standard parsing configuration and hyperparameter settings. For evaluation, we compute the percentage of words that get the correct head, reporting both unlabelled attachment score (UAS) and labelled attachment score (LAS). LAS additionally requires the predicted dependency label to be correct. To measure statistical significance, we use a bootstrap test (Efron and Tibshirani, 1986) with 100K samples.

4.2 Pre-Trained Word Embeddings

We use the 100-dimensional GloVe word embeddings from Pennington et al. (2014). These were trained on Wikipedia 2014 and the Gigaword v5 corpus and have a vocabulary size of approximately 400,000.[2]

4.3 Datasets

We consider a number of datasets with varying rates of OOTV words. We define the OOTV rate (or, equivalently, the unseen rate) of a dataset as the percentage of the vocabulary (types) of words occurring in the set that were not seen in training.

Wall Street Journal (WSJ) and OntoNotes-WSJ We conduct experiments on the Wall Street Journal portion of the English Penn Treebank dataset (Marcus et al., 1993). We follow the standard splits: sections 2-21 for training, section 22 for validation, and section 23 for testing. We convert the original phrase structure trees into dependency trees using Stanford Basic Dependencies (De Marneffe and Manning, 2008) in the Stanford Dependency Parser. The POS tags are obtained using the Stanford POS tagger (Toutanova et al., 2003) in a 10-fold jackknifing setup on the training data (achieving an accuracy of 96.96%). The OOTV rate in the development and test sets is approximately 2-3%.

We also conduct experiments on the OntoNotes 4.0 dataset (which we denote OntoNotes-WSJ). This dataset contains the same sentences as the WSJ corpus (and we use the same data splits), but has significantly different annotations. The OntoNotes-WSJ training data is used for the Web Treebank test experiments. We perform the same pre-processing steps as for the WSJ dataset.

Web Treebank We expect our mapper to be most effective when parsing held-out data with many unseen words. This often happens when the held-out data is drawn from a different distribution than the training data. For example, when training a parser on newswire and testing on web data,

[2] `http://www-nlp.stanford.edu/data/glove.6B.100d.txt.gz`; We have also experimented with the downloadable 50-dimensional SENNA embeddings from Collobert et al. (2011) and with `word2vec` (Mikolov et al., 2013) embeddings that we trained ourselves; in preliminary experiments the GloVe embeddings performed best, so we use them for all experiments below.

many errors occur due to differing patterns of syntactic usage and unseen words (Foster et al., 2011; Petrov and McDonald, 2012; Kong et al., 2014; Wang et al., 2014).

We explore this setting by training our parser on OntoNotes-WSJ and testing on the Web Treebank (Petrov and McDonald, 2012), which includes five domains: answers, email, newsgroups, reviews, and weblogs. Each domain contains approximately 2000-4000 manually annotated syntactic parse trees in the OntoNotes 4.0 style. In this case, we are adapting the parser which is trained on OntoNotes corpora using the small development set for each of the sub-domains (the size of the Web Treebank dev corpora is only around 1000-2000 trees so we use it for validation instead of including it in training). As before, we convert the phrase structure trees to dependency trees using Stanford Basic Dependencies. The parser and the mapper hyperparameters were tuned separately on the development set for each domain. The unseen rate is typically 6-10% in the domains of the Web Treebank. We used the Stanford tagger (Toutanova et al., 2003), which was trained on the OntoNotes training corpus, for part-of-speech tagging the Web Treebank corpora. The tagger used bidirectional architecture and it included word shape and distributional similarity features. We train a separate mapper for each domain, tuning mapper hyperparameters separately for each domain using the development sets. In this way, we obtain some of the benefits of domain adaptation for each target domain.

Downstream Task: Sentiment Analysis with Dependency Tree LSTMs We also perform experiments to analyze the effects of embedding mapping on a downstream task, in this case sentiment analysis using the Stanford Sentiment Treebank (Socher et al., 2013). We use the dependency tree long short-term memory network (Tree-LSTM) proposed by Tai et al. (2015), simply replacing their default dependency parser with our version that maps unseen words. The dependency parser is trained on the WSJ corpus and mapped using the WSJ development set. We use the same mapper that was optimized for the WSJ development set, without further hyperparameter tuning for the mapper. For the Tree-LSTM model, we use the same hyperparameter tuning as described in Tai et al. (2015). We use the standard train/development/test splits of 6820/872/1821 sentences for the binary classification task and 8544/1101/2210 for the fine-grained task.

4.4 Mapper Settings and Hyperparameters

The initial embeddings given to the mapper are the same as the initial embeddings given to the parser. These are the 100-dimensional GloVe embeddings mentioned above. The output dimensionality of the mapper is also fixed to 100. All model parameters of the mappers are initialized to zero. We set the dimensionality of the non-linear layer to 400 across all experiments. The model parameters are optimized by maximizing the weighted multiple-loss objective using L-BFGS with elastic-net regularization (Section 2). The hyperparameters include the relative weight of the two objective terms (α) and the regularization constants (λ_1, λ_2). For α, we search over values in $\{0, 0.1, 0.2, \ldots, 1\}$. For each of λ_1 and λ_2, we consider values in $\{10^{-1}, 10^{-2}, \ldots, 10^{-9}, 0\}$. The hyperparameters are tuned via grid search to maximize the UAS on the development set.

5 Results and Analysis

5.1 Results on WSJ, OntoNotes, and Switchboard

The upper half of Table 1 shows our main test results on WSJ, OntoNotes, and Switchboard, the low-OOTV rate datasets. Due to the small initial OOTV rates (<3%), we only see modest gains of 0.3-0.4% in UAS, with statistical significance at $p < 0.05$ for WSJ and OntoNotes and $p < 0.07$ for Switchboard. The initial OOTV rates are cut in half by our mapper, with the remaining unknown words largely being numerical strings and misspellings.[3] When only considering test sentences containing OOTV words (the row labeled "OOTV subset"), the gains are significantly larger (0.5-0.8% UAS at $p < 0.05$).

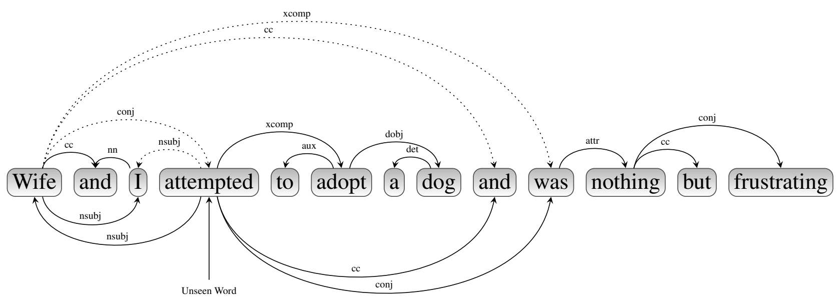
5.2 Results on Web Treebank

The lower half of Table 1 shows our main test results on the Web Treebank's five domains, the high-OOTV rate datasets. As expected, the mapper has a much larger impact when parsing these out-of-domain datasets with high OOTV word

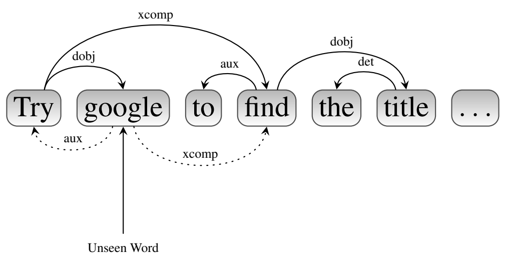
[3]We could potentially train the initial embeddings on a larger corpus or use heuristics to convert unknown numbers and misspellings to forms contained in our initial embeddings.

	Lower OOTV word rate			Higher OOTV word rate					
	WSJ	OntoNotes	Avg.	Answers	Emails	Newsgroups	Reviews	Weblogs	Avg.
UAS	91.85→92.21	90.17→90.49	90.38→90.70	82.67→83.21	81.76→82.42	84.68→85.13	84.25→84.99	87.73→88.43	84.22→84.84
LAS	89.49→89.73	87.68→87.92	87.92→88.14	78.98→79.59	77.93→78.56	81.88→82.71	81.26→81.92	85.68→86.29	81.01→81.81
OOTV %	2.72→1.45	2.72→1.4	–	8.53→1.22	10.56→3.01	10.34→1.04	6.84→0.73	8.45→0.38	–
OOTV UAS	89.88→90.51	89.27→89.81	89.12→89.78	80.88→81.75	79.29→81.02	82.54→83.71	81.17→82.22	86.43→87.31	82.06→83.20
#Sents	337	329	–	671	644	579	632	541	–

Table 1: Results of dependency parsing on various treebanks. Entries of the form A→B give results for parsing without mapped embeddings (A) and with mapped embeddings (B). "OOTV %" entries A→B indicate that A% of the test set vocabulary was unseen in the parser training, and B% remain unknown after mapping the embeddings. "OOTV UAS" refers to UAS measured on the subset of the test set sentences that contain at least one OOTV word, and "#Sents" gives the number of sentences in this subset.


Wife and I attempted to adopt a dog and was nothing but frustrating
xcomp
cc
conj
nn
nsubj
aux
dobj
det
attr
Unseen Word

(a) We obtain correct attachments and correct tree after the mapper maps the unseen word 'attempted'.



(b) The mapper incorrectly maps 'google', resulting in wrong attachments and wrong tree.

Figure 2: Examples where the mapper helps and hurts: In the above examples the top arcs are before mapping and bottom ones are after mapping; dotted lines refer to incorrect attachment.

rates.[4]

The OOTV rate reduction is much larger than for the WSJ-style datasets, and the parsing improvements (UAS and LAS) are statistically significant at $p < 0.05$. On subsets containing at least one OOTV word (that also has an initial embedding), we see an average gain of 1.14% UAS (see row labeled "OOTV subset"). In this case, all improvements are statistically significant at $p < 0.02$. We observe that the relative reduction in OOTV% for the Web Treebanks is larger than for the WSJ, OntoNotes, or Switchboard datasets. In particular, we are able to reduce the OOTV% by 71-95% relative. We also see the intuitive trend that larger relative reductions in OOTV rate correlate with larger accuracy improvements.

[4]As stated above, we train the parser on the OntoNotes dataset, but tune mapper hyperparameters to maximize parsing performance on each development section of the Web Treebank's five domains. We then map the OOTV word vectors on each test set domain using the learned mapper for that domain.

5.3 Downstream Results

We now report results using the Dependency Tree-LSTM of Tai et al. (2015) for sentiment analysis on the Stanford Sentiment Treebank. We consider both the binary (positive/negative) and fine-grained classification tasks ({very negative, negative, neutral, positive, and very positive}). We use the implementation provided by Tai et al. (2015), changing only the dependency parses that are fed to their model. The sentiment dataset contains approximately 25% OOTV words in the training set vocabulary, 5% in the development set vocabulary, and 9% in the test set vocabulary. We map unseen words using the mapper tuned on the WSJ development set. We use the same Dependency Tree-LSTM experimental settings as Tai et al. Re-

Fine-Grained	Binary
48.4→49.5	85.7→ 86.1

Table 2: Improvements on Stanford Sentiment Treebank test set using our parser with the Dependency Tree-LSTM.

Baseline	$\boldsymbol{t_1}$	$\boldsymbol{t_3}$	$\boldsymbol{t_5}$	$\boldsymbol{t_\infty}$
84.11	84.89	84.97	84.81	84.14

Table 3: Average Web Treebank development UAS at different threshold settings.

sults are shown in Table 2. We improve upon the original accuracies in both binary and fine-grained classification. [5] We also reduce the OOTV rate from 25% in the training set vocabulary to about 6%, and from 9% in the test set vocabulary down to 4%.

5.4 Effect of Thresholds

We also experimented with different values for the thresholds described in Section 2. For the mapping threshold τ_m, mapper-training threshold τ_t, and parser threshold τ_p, we consider the following four settings:

$$\begin{aligned}\boldsymbol{t_1} &: \tau_m = \tau_t = \tau_p = 1\\ \boldsymbol{t_3} &: \tau_m = \tau_t = \tau_p = 3\\ \boldsymbol{t_5} &: \tau_m = \tau_t = \tau_p = 5\\ \boldsymbol{t_\infty} &: \tau_m = \infty, \tau_p = \tau_t = 5\end{aligned}$$

Using $\tau_m = \infty$ corresponds to mapping all words at test time, even words that we have seen many times in the training data and learned task-specific embeddings for.

We report the average development set UAS over all Web Treebank domains in Table 3. We see that $\boldsymbol{t_3}$ performs best, though settings $\boldsymbol{t_1}$ and $\boldsymbol{t_5}$ also improve over the baseline. At threshold $\boldsymbol{t_3}$ we have approximately 20,000 examples for training the mapper, while at threshold $\boldsymbol{t_5}$ we have only about 10,000 examples. We see a performance drop at $\boldsymbol{t_\infty}$, so it appears better to directly use the task-specific embeddings for words that appear frequently in the training data. In other results reported in this paper, we used $\boldsymbol{t_3}$ for the Web Treebank test sets and $\boldsymbol{t_1}$ for the rest.

5.5 Effect of Weighted Multi-Loss Objective

We analyzed the results when varying α, which balances between the two components of the mapper's multi-loss objective function. We found that, for all domains except Answers, the best results are obtained with some α between 0 and 1. The optimal values outperformed the cases with $\alpha = 0$ and $\alpha = 1$ by 0.1-0.3% UAS absolute. However, on the Answers domain, the best performance was achieved with $\alpha = 0$; i.e., the mapper preferred mean squared error. For other domains, the optimal α tended to be within the range $[0.3, 0.7]$.

5.6 Comparison with Related Work

We compare to the approach presented by Tafforeau et al. (2015). They propose to refine embeddings for unseen words based on the relative shifts of their k nearest neighbors in the original embeddings space. Specifically, they define "artificial refinement" as:

$$\phi_r(t) = \phi_o(t) + \sum_{k=1}^{K} \alpha_k(\phi_r(n_k) - \phi_o(n_k)) \quad (3)$$

where $\phi_r(.)$ is the vector in the refined embedding space and $\phi_o(.)$ is the vector in the original embedding space. They define α_k to be proportional to the cosine similarity between the target unseen word (t) and neighbor (n_k):

$$\alpha_k = s(t, n_k) = \frac{\phi_o(t).\phi_o(n_k)}{|\phi(t)||\phi_o(n_k)|}$$

	Avg. UAS	Avg. LAS
Baseline	84.11	81.02
k-NN	84.54	81.38
Our Mapped	84.97	81.79

Table 4: Comparison to k-nearest neighbor matching of Tafforeau et al. (2015).

Table 4 shows the average performance of the models over the development sets of the Web Treebank. On average, our mapper outperforms the k-NN approach ($k = 3$).

5.7 Dependency Parsing Examples

In Figure 2, we show two sentences: an instance where the mapper helps and another where the mapper hurts the parsing performance.[6] In the first sentence (Figure 2a), the parsing model has not seen the word 'attempted' during training. Note that the sentence contains 3 verbs: 'attempted', 'adopt', and 'was'. Even with the POS tags, the

[5]Note that we report accuracies and improvements on the dependency parse based system of Tai et al. (2015) because the neural parser that we use is dependency-based.

[6]Sentences in Figure 2 are taken from the development portion of the Answers domain from the Web Treebank.

parser was unable to get the correct dependency attachment. After mapping, the parser correctly makes ‘attempted’ the root and gets the correct arcs and the correct tree. The 3 nearest neighbors of ‘attempted’ in the mapped embedding space are ‘attempting’, ‘tried’, and ‘attempt’. We also see here that a single unseen word can lead to multiple errors in the parse.

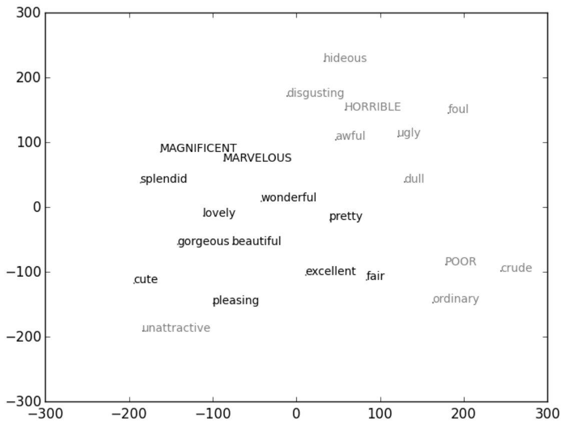
In the second example (Figure 2b), the default model assigns the correct arcs using the POS information even though it has not seen the word ‘google’. However, using the mapped representation for ‘google’, the parser makes errors. The 3-nearest neighbors for ‘google’ in the mapped space are ‘damned’, ‘look’, and ‘hash’. We hypothesize that the mapper has mapped this noun instance of ‘google’ to be closer to verbs instead of nouns, which would explain the incorrect attachment.

5.8 Analyzing Mapped Representations

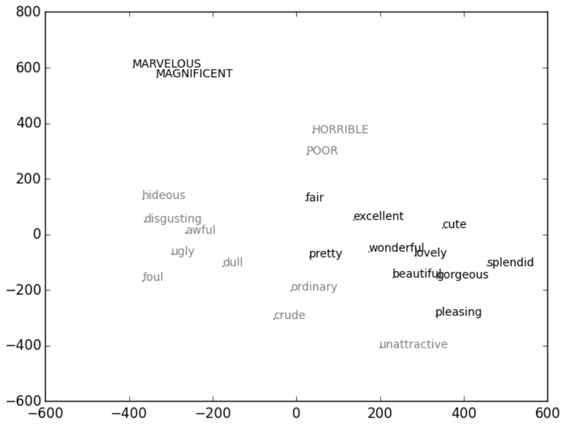
To understand the mapped embedding space, we use t-SNE (Van der Maaten and Hinton, 2008) to visualize a small subset of embeddings. In Figure 3, we plot the initial embeddings, the parser-trained embeddings, and finally the mapped embeddings. We include four unseen words (shown in caps): ‘horrible’, ‘poor’, ‘marvelous’, and ‘magnificent’. In Figure 3a and Figure 3b, the embeddings for the unseen words are identical (even though t-SNE places them in different places when producing its projection). In Figure 3c, we observe that the mapper has placed the unseen words within appropriate areas of the space with respect to similarity with the seen words. We contrast this with Figure 3b, in which the unseen words appear to be within a different region of the space from all seen words.

6 Conclusion

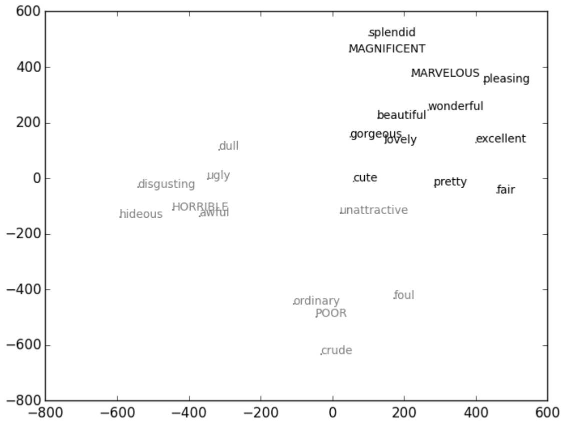
We have described a simple method to resolve unseen words when training supervised models that learn task-specific word embeddings: a feed-forward neural network that maps initial embeddings to the task-specific embedding space. We demonstrated significant improvements in dependency parsing accuracy across several domains, as well as improvements on a downstream task. Our approach is simple, effective, and applicable to many other settings, both inside and outside NLP.



(a) Initial Representational Space



(b) Learned Representational Space



(c) Mapped Representational Space

Figure 3: t-SNE plots on initial, parser trained, and mapped representations.

Acknowledgments

We would like to thank the anonymous reviewers for their useful comments. This research was supported by a Google Faculty Research Award to Mohit Bansal, Karen Livescu and Kevin Gimpel.

References

Mohammed Attia, Jennifer Foster, Deirdre Hogan, Joseph Le Roux, Lamia Tounsi, and Josef Van Genabith. 2010. Handling unknown words in statistical latent-variable parsing models for arabic, english and french. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 67–75. Association for Computational Linguistics.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, Lisbon, Portugal, September. Association for Computational Linguistics.

Mohit Bansal and Dan Klein. 2011. Web-scale features for full-scale parsing. In *Proceedings of ACL*.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of ACL*.

Jan A. Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *International Conference on Machine Learning (ICML)*.

Marie Candito and Benoît Crabbé. 2009. Improving generative statistical parsing with semi-supervised word clustering. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 138–141. Association for Computational Linguistics.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Rohan Chitnis and John DeNero. 2015. Variable-length word encodings for neural translation models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2088–2093, Lisbon, Portugal, September. Association for Computational Linguistics.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:24932537.

Marie-Catherine De Marneffe and Christopher D Manning. 2008. Stanford typed dependencies manual. Technical report, Stanford University.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July. Association for Computational Linguistics.

B. Efron and R. Tibshirani. 1986. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statist. Sci.*, 1(1):54–75, 02.

Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef Van Genabith. 2011. # hardtoparse: Pos tagging and parsing the twitterverse. In *AAAI 2011 Workshop on Analyzing Microtext*, pages 20–25.

Nizar Habash. 2009. Remoov: A tool for online handling of out-of-vocabulary words in machine translation. In *Proceedings of the 2nd International Conference on Arabic Language Resources and Tools (MEDAR), Cairo, Egypt*.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10, Beijing, China, July. Association for Computational Linguistics.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland, June. Association for Computational Linguistics.

Frank Keller and Mirella Lapata. 2003. Using the web to obtain frequencies for unseen bigrams. *Computational linguistics*, 29(3):459–484.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October. Association for Computational Linguistics.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2015. Skip-thought vectors. *arXiv preprint arXiv:1506.06726*.

Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah A. Smith. 2014. A dependency parser for tweets. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1001–1012, Doha, Qatar, October. Association for Computational Linguistics.

Wang Ling, Tiago Luís, Luís Marujo, Rámon Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proc. of EMNLP*.

D. C. Liu and J. Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45(3, (Ser. B)):503–528.

Thang Luong, Richard Socher, and Christopher Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, Sofia, Bulgaria, August. Association for Computational Linguistics.

Thang Luong, Ilya Sutskever, Quoc Le, Oriol Vinyals, and Wojciech Zaremba. 2015. Addressing the rare word problem in neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 11–19, Beijing, China, July. Association for Computational Linguistics.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Yuval Marton, Nizar Habash, and Owen Rambow. 2010. Improving arabic dependency parsing with lexical and inflectional morphological features. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 13–21, Los Angeles, CA, USA, June. Association for Computational Linguistics.

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

Silvia Necsulescu, Sara Mendes, David Jurgens, Núria Bel, and Roberto Navigli. 2015. Reading between the lines: Overcoming data sparsity for accurate classification of lexical relationships. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, pages 182–192, Denver, Colorado, June. Association for Computational Linguistics.

Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V. Le, and Andrew Y. Ng. 2011. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272.

Romain Paulus, Richard Socher, and Christopher D Manning. 2014. Global belief recursive neural networks. In *Advances in Neural Information Processing Systems*, pages 2888–2896.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October. Association for Computational Linguistics.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL).

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 705–713. Association for Computational Linguistics.

Lizhen Qu, Gabriela Ferraro, Liyuan Zhou, Weiwei Hou, Nathan Schneider, and Timothy Baldwin. 2015. Big data small data, in domain out-of domain, known word unknown word: The impact of word representation on sequence labelling tasks. *arXiv preprint arXiv:1504.05319*.

Djamé Seddah, Grzegorz Chrupała, Ozlem Cetinoglu, Josef van Genabith, and Marie Candito. 2010. Lemmatization and lexicalized statistical parsing of morphologically-rich languages: the case of french. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 85–93, Los Angeles, CA, USA, June. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October. Association for Computational Linguistics.

Anders Søgaard and Anders Johannsen. 2012. Robust learning in random subspaces: Equipping NLP for OOV effects. In *Proceedings of COLING 2012: Posters*, Mumbai, India, December.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Jeremie Tafforeau, Thierry Artieres, Benoit Favre, and Frederic Bechet. 2015. Adapting lexical representation and oov handling from written to spoken language with word embedding. In *Interspeech*.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semisupervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, page 384394. Association for Computational Linguistics.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85.

William Yang Wang, Lingpeng Kong, Kathryn Mazaitis, and William W Cohen. 2014. Dependency parsing for weibo: An efficient probabilistic logic programming approach. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1152–1158, Doha, Qatar, October. Association for Computational Linguistics.