Dependency Link Embeddings: Continuous Representations of Syntactic Substructures

Mohit Bansal
Toyota Technological Institute at Chicago, IL 60637, USA
mbansal@ttic.edu

Abstract

We present a simple method to learn continuous representations of dependency substructures (links), with the motivation of directly working with higher-order, structured embeddings and their hidden relationships, and also to avoid the millions of sparse, template-based word-cluster features in dependency parsing. These link embeddings allow a significantly smaller and simpler set of *unary* features for dependency parsing, while maintaining improvements similar to state-of-the-art, n-ary word-cluster features, and also stacking over them. Moreover, these link vectors (made publicly available) are directly portable as off-the-shelf, dense, syntactic features in various NLP tasks. As one example, we incorporate them into constituent parse reranking, where their small feature set again matches the performance of standard non-local, manually-defined features, and also stacks over them.

1 Introduction

Word representations and more recently, word embeddings, learned from large amounts of text have been quite successful as features in various NLP tasks (Koo et al., 2008; Turian et al., 2010; Collobert et al., 2011; Dhillon et al., 2012; Al-Rfou' et al., 2013; Bansal et al., 2014; Guo et al., 2014; Pennington et al., 2014; Yu and Dredze, 2014; Faruqui et al., 2014; Wang et al., 2015). While these word representations do capture useful, dense relationships among known and unknown words, one still has to work with sparse conjunctions of features on the multiple words involved in the substructure that a task factors on, e.g., head-argument links in dependency parsing. Therefore, most statistical dependency parsers still suffer from millions of such conjoined, template-based, n-ary features on word clusters or embeddings (Koo et al., 2008; Bansal et al., 2014). Some recent work has addressed this issue, via low-rank tensor mappings (Lei et al., 2014), feature embeddings (Chen et al., 2014), or neural network parsers (Chen and Manning, 2014).

Secondly, it would also be useful to learn dense representations directly for the higher-order substructures (that structured NLP tasks factor on) so as to explicitly capture the useful, hidden relationships among these substructures, instead of relying on the sparse word-conjoined relationships.

In this work, we propose to address both these issues by learning simple dependency *link embeddings* on 'head—argument' pairs (as a single concatenated unit), which allows us to work directly with linguistically-intuitive, higher-order substructures, and also fire significantly fewer and simpler features in dependency parsing, as opposed to word cluster and embedding features in previous work (Koo et al., 2008; Bansal et al., 2014), while still maintaining their strong accuracies.

Trained using appropriate dependency-based context in `word2vec`, the fast neural language model of Mikolov et al. (2013a), these link vectors allow a substantially smaller set of *unary* link features (as opposed to n-ary, conjoined features) which provide savings in parsing time and memory. Moreover, unlike conjoined features, link embeddings allow a tractable set of accurate per-dimension features, making the feature set even smaller and the feature-generation process orders of magnitude faster (than

hierarchical clustering features).

At the same time, these link embedding features maintain dependency parsing improvements similar to the complex, template-based features on word clusters and embeddings by previous work (Koo et al., 2008; Bansal et al., 2014) (up to 9% relative error reduction), and also stack statistically significantly over them (up to an additional 5% relative error reduction).

Another advantage of this approach (versus previous work on feature embeddings or special neural networks for parsing) is that these link embeddings can be imported as off-the-shelf, dense, syntactic features into various other NLP tasks, similar to word embedding features, but now with richer, structured information, and in tasks where plain word embeddings have not proven useful . As an example, we incorporate them into a constituent parse reranker and see improvements that again match state-of-the-art, manually-defined, non-local reranking features and stack over them statistically significantly. We make our link embeddings publicly available[1] and hope that they will prove useful in various other NLP tasks in future work, e.g., as dense, syntactic features in sentence classification or as linguistically-intuitive, initial units in vector-space composition.

2 Dependency Link Embeddings

To train the link embeddings, we use the speedy, skip-gram neural language model of Mikolov et al. (2013a; 2013b) via their toolkit `word2vec`.[2] We use the original skip-gram model and simply change the context tuple data on which the model is trained, similar to Bansal et al. (2014) and Levy and Goldberg (2014). The goal is to learn similar embeddings for links with similar syntactic contextual properties like label, signed distance, ancestors, etc.

To this end, we first parse the BLLIP corpus (minus the PTB portion)[3] using the baseline MSTParser (McDonald et al., 2005b). Next, for each predicted link, we create a tuple, consisting of the parent-child pair p—c (concatenated as a single unit, same as p_c) and its various properties such as the link's dependency relation label l, the grandparent dependency relation label gl, and the signed, binned distance d:

N.Y.–Yonkers, Md.–Columbia, N.Y.–Bronx, Va.–Reston, Ky.–Lexington, Mich.–Kalamazoo, Calif.–Calabasas, ...
boost–revenue, tap–markets, take–losses, launch–fight, reduce–holdings, terminate–contract, identify–bidders, ...
boosting–bid, meeting–schedules, obtaining–order, having–losses, completing–review, governing–industry, ...
says–mean, adds–may, explains–have, contend–has, recalls–had, figures–is, asserted–is, notes–would, ...
would–Based, is–Besides, was–Like, is–From, are–Despite, said–Besides, says–Despite, reported–As, ...
began–Meanwhile, was–Since, are–Often, would–Now, had–During, were–Over, was–Late, have–Until, ...
Catsimatidis–Mr., Swete–Mr., Case–Mr., Montoya–Mr., Byerlein–Mr., Heard–Mr., Leny–Mr., Graham–Mrs., ...
only–1.5, about–170, nearly–eight, approximately–10, almost–15, some–80, Only–two, about–23, roughly–50, ...

Table 1: Example clusters of the link embeddings.

$$\text{“}d_{\text{<D>}} \quad gl_{\text{<GL>}} \quad p\text{—}c \quad l_{\text{<L>}} \quad d_{\text{<D>}}\text{”}, \qquad (1)$$

We then run the skip-gram model on the the above context tuples (Eq. 1) with a window-size of 2, dimension-size of 100, and a min-count cutoff of 4 to give us a vocabulary of around 92K.[4] We also tried other context settings, e.g., where we add more lexicalized, link-based context to the tuple such as the neighboring grandparent-parent link gp—p:

$$\text{“}gl_{\text{<GL>}} \quad gp\text{—}p \quad p\text{—}c \quad d_{\text{<D>}} \quad l_{\text{<L>}}\text{”}, \qquad (2)$$

but the setting in Eq. 1 performs slightly better (based on the development set).

Clusters: Table 1 shows example clusters obtained by clustering link embeddings via MATLAB's `linkage + cluster` commands, with 1000 clusters.[5] We can see that these link embeddings are able to capture useful groups and subtle distinctions directly at the link level (without having to work with all pairs of word types), e.g., based on syntactic properties like capitalization, verb form, position in sentence; and based on topics like location, time, finance, etc.

[1] `ttic.edu/bansal`

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

[3] Same dataset as what was used to train the BROWN clusters in Koo et al. (2008), for comparability.

[4] We add subscripts to all context tokens so as to treat them differently and remove them from the vocabulary after training.

[5] `http://www.mathworks.com/help/stats/linkage.html`, `http://www.mathworks.com/help/stats/cluster.html`

3 Dependency Parsing Experiments

In this section, we will first discuss how we use the link embeddings as features in dependency parsing. Next, we will present empirical results on feature space reduction and on parsing performance on both in-domain and out-of-domain datasets.

3.1 Features

The BROWN cluster features are based on Bansal et al. (2014), who follow Koo et al. (2008) to add 1st and 2nd order features to MSTParser based on prefixes (of length 4, 6, 8, and 12) of the 0-1 hierarchical clustering bit-strings (via the bigram class-based language model of Brown et al. (1992)) of the head and argument, siblings, intermediate words, etc. See McDonald et al. (2005a) and Koo et al. (2008) for the exact feature templates.

For link embeddings, we tried two feature types:

Bucket features: For each dimension of the link vector, we fire a simple indicator feature, where the feature name consists of the dimension index d and the bucketed vector value b at that index (using a bucket of 0.25), i.e., simply $d \wedge b$, as compared to the large list of n-ary feature templates in previous work, which include various conjunctions, in-between and surrounding word information, etc. (see McDonald et al. (2005a) and Koo et al. (2008)). We have another feature that additionally includes the signed, bucketed distance of the particular link in the given sentence.

Also note the difference of our unary bucket features from the binary bucket features of Bansal et al. (2014), who had to work with pairwise, conjoined features of the head and the argument. Hence, they used features on conjunctions of the two bucket values from the head and argument word vectors, firing one pairwise feature per dimension, because firing features on all dimension pairs (corresponding to an outer product) led to an infeasible number of features. The result discussion of these feature differences in presented in §3.2.

Bit-string features: We first hierarchically cluster the link vectors via MATLAB's `linkage` function with {*method=ward, metric=euclidean*} to get 0-1 bit-strings (similar to BROWN). Next, we again fire a small set of *unary* indicator features that simply consist of the link's bit-string prefix, the prefix-length, and another feature that adds the signed, bucketed distance of that link in the sentence.[6]

System	Number of features
Baseline	5M
BROWN	13M
Bansal et al. (2014)	30M
Bucket	**15K**
Bit-string	1M

Table 2: Number of features.

System	Dev	Test
Baseline	92.4	91.9
+ BROWN	93.2	92.7
+ Bucket	93.0	92.3
+ Bit-string	92.9	92.6
+ BROWN + Bucket	93.4	93.0
+ BROWN + Bit-string	**93.4**	**93.1**

Table 3: UAS results on WSJ.

3.2 Setup and Results

For all experiments (unless otherwise noted), we follow the 2nd-order MSTParser setup of Bansal et al. (2014), in terms of data splits, parameters, preprocessing, and feature thresholding. *Statistical significance* is reported based on the bootstrap test (Efron and Tibshirani, 1994) with 1 million samples.

First, we compare the **number of features** in Table 2. Our dense, unary, link-embedding based Bucket and Bit-string features are substantially fewer than the sparse, n-ary, template-based features used in the MSTParser baseline, in BROWN, and in the word embedding SKIP_{DEP} result of Bansal et al. (2014). This in turn also improves our parsing speed and memory. Moreover, regarding the **preprocessing time** taken to generate these various feature types, our Bucket features, which just need the fast `word2vec` training, take 2-3 orders of magnitude lesser time than the BROWN features (*15 mins. versus 2.5 days*)[7]; this is also advantageous when

[6] We again used prefixes of length 4, 6, 8, 12, same as the BROWN feature setting. For unknown links' features, we replace the bucket or bit-string prefix with a special 'UNK' string.

[7] Based on a modern 3.50 GHz desktop and 1 thread. The Bit-string features additionally need hierarchical clustering, but are still at least twice as fast as BROWN features.

System	Test Average
Baseline	83.5
+ BROWN	84.2
+ Bucket	84.0
+ Bit-string	83.8
+ BROWN + Bucket	**84.6**
+ BROWN + Bit-string	84.4

Table 4: UAS results on Web treebanks.

	Dev		Test	
Parsing Model	F1	EX	F1	EX
Baseline (1-best)	90.6	39.4	90.2	37.3
Baseline (log $p(t\|w)$)	90.4	38.9	89.9	37.3
+ Config	91.8	43.8	91.1	40.6
+ Bit-string	91.1	40.3	90.9	40.6
+ Config + Bit-string	**92.0**	**43.9**	**91.4**	**42.0**

Table 5: F1 results of constituent reranker on WSJ.

training and parsing with representations of new domains or languages.

Table 3 shows the main UAS (unlabeled attachment score) results on **WSJ**, where each '+ X' row denotes adding type X features to the MSTParser baseline. All the final test improvements, i.e., Bucket (92.3) and Bit-string (92.6) w.r.t. Baseline (91.9), and BROWN + Bucket (93.0) and BROWN + Bit-string (93.1) w.r.t. BROWN (92.7), are *statistically significant* at $p < 0.01$. Moreover, the Bit-string result (92.6) is the same, i.e., has no statistically significant difference from the BROWN result (92.7), and also from the Bansal et al. (2014) SKIP_{DEP} result (92.7). Therefore, the main contribution of these link embeddings is that their significantly simpler, smaller, and faster set of unary features can match the performance of complex, template-based BROWN features (and of the dependency-based word embedding features of Bansal et al. (2014)), and also stack over them. We also get similar trends of improvements on the labeled attachment score (LAS) metric.[8]

Moreover, unlike Bansal et al. (2014), our Bucket features achieve statistically significant improvements, most likely because they fired D pairwise, conjoined features, one per dimension d, consisting of the two bucket values from the head and argument word vectors. This would disallow the classifier to learn useful linear combinations of the various dimensions. Firing D^2 features on all dimension pairs (corresponding to an outer product) would lead to an infeasible number of features. On the other hand, we have a single vector for head+argument, allowing us to fire just D features (one per dimension) and still learn useful dimension combinations in linear space.

We also report out-of-domain performance, in Table 4, on the **Web** treebank (Petrov and McDonald, 2012) test sets, directly using the WSJ-trained models. Again, both our Bucket and Bit-string link-embedding features achieve decent improvements over Baseline and they stack over BROWN, while using much fewer features. Moreover, one can hopefully achieve bigger gains by training link embeddings on Web or Wikipedia data (since BLLIP is news-domain).

4 Off-the-shelf: Constituent Parsing

Finally, these link embeddings are also portable as off-the-shelf, dense, syntactic features into other NLP tasks, either to incorporate missing syntactic information, or to replace sparse (n-ary lexicalized or template-based) parsing features, or where word embedding features are not appropriate and one needs higher-order embeddings, e.g., in constituent parsing (see Andreas and Klein (2014)).

Therefore, as a first example, we import our link embedding features into a constituent parse reranker. We follow Bansal and Klein (2011), reranking 50-best lists of the Berkeley parser (Petrov et al., 2006). We first extract dependency links in each candidate constituent tree based on the head-modifier rules of Collins (2000). Next, we simply fire our Bit-string features on each link, where the feature again consists of just the prefix bit-string, the prefix length, and the signed, bucketed link distance.[9]

Table 5 shows these reranking results, where 1-best and log $p(t|w)$ are the two Berkeley parser baselines, and where Config is the state-of-the-art, non-local, configurational feature set of Huang (2008),

[8]Note that one can achieve even stronger results by tuning separate prefix lengths for the Bit-string versus the BROWN + Bit-string cases.

[9]Based on development set tuning, we use prefixes 4, 6, 8, and then gaps of 4 up to the full-length for '+ Bit-string' and prefixes 4, 6, 8, 12, 16, and full-length for '+ Config + Bit-string'.

which in turn is a simplified merge of Charniak and Johnson (2005) and Collins (2000) (here configurational). Again, all our test improvements are *statistically significant* at $p < 0.01$: Bit-string (90.9) over both the baselines (90.2, 89.9); and Config + Bit-string (91.4) over Config (91.1). Moreover, the Bit-string result (90.9) is the same (i.e., no statistically significant difference) as the Config result (91.1). Therefore, we can again match the improvements of complex, manually-defined, non-local reranking features with a much smaller set of simple, dense, off-the-shelf, link-embedding features, and also complement them statistically significantly.

5 Related Work

As mentioned earlier, there has been a lot of useful, previous work on using word embeddings for NLP tasks such as similarity, tagging, NER, sentiment analysis, and parsing (Turian et al., 2010; Collobert et al., 2011; Dhillon et al., 2012; Huang et al., 2012; Al-Rfou' et al., 2013; Hisamoto et al., 2013; Andreas and Klein, 2014; Bansal et al., 2014; Guo et al., 2014; Pennington et al., 2014; Wang et al., 2015), inter alia.

In related work, Bansal et al. (2014) also use dependency context to tailor word embeddings to dependency parsing. However, their embedding features are still based on the sparse set of n-ary, word-based templates from previous work (McDonald et al., 2005a; Koo et al., 2008). Our structured link embeddings achieve similar improvements as theirs (and better in the case of direct, per-dimension bucket features) with a substantially smaller and simpler (unary) set of features that are aimed to directly capture hidden relationships between the substructures that dependency parsing factors on. Moreover, we hope that similar to word embeddings, these link embeddings will also prove useful when imported into various other NLP tasks as dense, continuous features, but now with additional syntactic information.

There has also been some recent, useful work on reducing the sparsity of features in dependency parsing, e.g., via low-rank tensors (Lei et al., 2014) and via neural network parsers that learn tag and label embeddings (Chen and Manning, 2014). In related work, Chen et al. (2014) learn dense feature embeddings for dependency parsing; however, they still work with the large number of manually-defined feature templates from previous work and train embeddings for all those templates, with an aim to discover hidden, shared information among the large set of sparse features. We get similar improvements with a much smaller and simpler set of unary link features; also, our link embeddings are more portable to other NLP tasks than template-based embeddings specific to dependency parsing.

Other work includes learning distributed structured output via dense label vectors (Srikumar and Manning, 2014), learning bilexical operator embeddings (Madhyastha et al., 2014), and learning joint word embeddings and composition functions based on predicate-argument compositionality (Hashimoto et al., 2014).

Our main goal is to directly learn embeddings on linguistically-intuitive units like dependency links, so that they can be used as non-sparse, unary features in dependency parsing, and also as off-the-shelf, dense, syntactic features in other NLP tasks (versus more intrinsic approaches based on feature embeddings or neural network parsers, which are harder to export).

6 Conclusion and Future Work

We presented dependency link embeddings, which provide a small, simple set of unary features for dependency parsing, while maintaining statistically significant improvements, similar and complementary to sparse, n-ary, word-cluster features. These link vectors are also portable as off-the-shelf syntactic features in other NLP tasks; we import them into constituent parse reranking, where they again match and stack over state-of-the-art, non-local reranking features. We release our link embeddings (available at `ttic.edu/bansal`) and hope that these will prove useful in various other NLP tasks, e.g., as dense, syntactic features in sentence classification or as linguistically-intuitive, initial units in vector-space composition.

In future work, it will be useful to try obtaining stronger parsing accuracies via newer, better representation learning tools, e.g., GloVe (Pennington et al., 2014), and by training on larger quantities

of automatically-parsed data. It will also be useful to perform intrinsic evaluation of these link embeddings on appropriate syntactic datasets and metrics, and extrinsic evaluation via various other NLP tasks such as sentence classification. Finally, it will be interesting to try parsers or frameworks where we can directly employ the embeddings as features, instead of bucketing or clustering them.

Acknowledgments

I would like to thank Kevin Gimpel, Ryota Tomioka, and the anonymous reviewers for their useful comments.

References

Rami Al-Rfou', Bryan Perozzi, and Steven Skiena. 2013. Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of CoNLL*.

Jacob Andreas and Dan Klein. 2014. How much do word embeddings encode about syntax? In *Proceedings of ACL*.

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*.

Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based N-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. of ACL*.

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

Wenliang Chen, Yue Zhang, and Min Zhang. 2014. Feature embedding for dependency parsing. In *Proceedings of COLING*.

M. Collins. 2000. Discriminative reranking for natural language parsing. In *Proc. of ICML*.

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

Paramveer Dhillon, Jordan Rodu, Dean P. Foster, and Lyle H. Ungar. 2012. Two Step CCA: A new spectral method for estimating vector models of words. In *Proceedings of ICML*.

Bradley Efron and Robert J. Tibshirani. 1994. *An introduction to the bootstrap*, volume 57. CRC press.

Manaal Faruqui, Jesse Dodge, Sujay K Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. 2014. Retrofitting word vectors to semantic lexicons. In *Proceedings of Deep Learning and Representation Learning Workshop, NIPS*.

Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. 2014. Revisiting embedding features for simple semi-supervised learning. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 110–120.

Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2014. Jointly learning word representations and composition functions using predicate-argument structures. In *Proceedings of EMNLP*.

Sorami Hisamoto, Kevin Duh, and Yuji Matsumoto. 2013. An empirical investigation of word representations for parsing the web. In *Proceedings of ANLP*.

Eric H. Huang, Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2012. Improving Word Representations via Global Context and Multiple Word Prototypes. In *Proceedings of ACL*.

L. Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. of ACL*.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL*.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of ACL*.

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of ACL*.

Pranava Swaroop Madhyastha, Xavier Carreras, and Ariadna Quattoni. 2014. Learning task-specific bilexical embeddings. In *Proceedings of COLING*.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proc. of ACL*.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT-EMNLP*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the Empiricial Methods*

in Natural Language Processing (EMNLP 2014), volume 12.

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

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proc. of COLING-ACL*.

Vivek Srikumar and Christopher D Manning. 2014. Learning distributional representations for structured output prediction. In *Proceedings of NIPS*.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*.

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

Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 545–550.